SPRINGER REFERENCE

Rafael Martí
Panos M. Pardalos
Mauricio G. C. Resende
Editors

Handbook of Heuristics

Springer

Handbook of Heuristics

Rafael Martí • Panos M. Pardalos
Mauricio G. C. Resende
Editors

Handbook of Heuristics

With 285 Figures and 120 Tables

Springer

Editors
Rafael Martí
Statistics and Operations Research Department
University of Valencia
Valencia, Spain

Panos M. Pardalos
Department of Industrial and Systems Engineering
University of Florida
Gainesville, FL, USA

Mauricio G. C. Resende
Amazon.com, Inc. and University of Washington
Seattle, WA, USA

ISBN 978-3-319-07123-7 ISBN 978-3-319-07124-4 (eBook)
ISBN 978-3-319-07125-1 (print and electronic bundle)
https://doi.org/10.1007/978-3-319-07124-4

Library of Congress Control Number: 2018940258

© Springer International Publishing AG, part of Springer Nature 2018
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Heuristics have become a very popular family of solution methods for optimization problems because they are capable of finding *acceptable* solutions in a *reasonable* amount of time. The word heuristic means *serving to discover* or *find out*. It is an irregular formation from the Greek term *heuretikos*, which means *inventive*. It is related to the Greek word *heuriskein*, meaning *to find* and the word *Eureka*, which comes from the ancient Greek *eurika*, which means *I have found it*.

In the last decades, algorithmic advances as well as hardware and software improvements have provided an excellent environment on which to build heuristic-based decision support systems based on new and effective methodologies. To the layman, heuristics may be thought of as *rules of thumb*, but despite its imprecision, the field of heuristics is very rich and refers to experience-based techniques for problem-solving, learning, and discovery. Any given heuristic solution is not guaranteed to be optimal, but heuristic methods are used to speed up the process of finding satisfactory solutions where optimal solutions are impractical. Despite the dynamic state of heuristic optimization, we feel that the time is ripe to bring together in one handbook the major algorithmic and methodological advances in this field. Leading experts in heuristic optimization have contributed to this handbook.

The main goal of this handbook is to provide the basic principles and fundamental ideas that will enable students and practitioners to create valuable applications based on heuristic technologies. Specifically, the *Handbook of Heuristics* is aimed at engineers, scientists, operations researchers, and other applications specialists who are looking for the most appropriate and recent optimization tools based on heuristic and metaheuristic methodologies to solve particular problems. The handbook provides a broad spectrum of advances in heuristic optimization with a focus on its algorithmic and computational aspects.

The handbook consists of five main parts: search strategies, local search, metaheuristics, analysis and implementations, and applications. In the part devoted to search strategies, we cover methodological aspects, from multi-objective and restart mechanisms to *matheuristics*, the exciting field in which mathematical programming is combined with heuristics. As it is well known to practitioners of heuristic optimization, local search constitutes the core of many methodologies. For this reason, we devote an entire part of the handbook to local search. In the metaheuristics part of the handbook, fourteen important methodologies in the major

field of heuristic optimization are described. We also focus on the analysis and implementation of these methods in this part of the book. Finally, the purpose of the part on applications is to provide the practitioner with a description of some relevant optimization issues in a number of specific application areas, such as scheduling, vehicle routing, network optimization, supply chain, diversity models, as well as some examples of applied optimization in specific areas.

Finally, we would like to take the opportunity to thank Springer for inviting us to edit this handbook, their staff for helping put the handbook into production, the authors for their contributions, and the anonymous referees for their valuable comments and suggestions.

Valencia, Spain — Rafael Martí
Gainesville, FL, USA — Panos M. Pardalos
Seattle, WA, USA — Mauricio G. C. Resende
May 2018

Contents

Volume 1

Part I Search Strategies **1**

1 Adaptive and Multilevel Metaheuristics 3
Marc Sevaux, Kenneth Sörensen, and Nelishia Pillay

2 Biased Random-Key Genetic Progamming 23
José Fernando Gonçalves and Mauricio G. C. Resende

3 Data Mining in Stochastic Local Search 39
Simone de Lima Martins, Isabel Rosseti, and Alexandre Plastino

4 Evolution Strategies 89
Michael Emmerich, Ofer M. Shir, and Hao Wang

5 Matheuristics 121
Martina Fischetti and Matteo Fischetti

6 Multi-start Methods 155
Rafael Martí, Jose A. Lozano, Alexander Mendiburu, and Leticia Hernando

7 Multi-objective Optimization 177
Carlos A. Coello Coello

8 Restart Strategies 205
Oleg V. Shylo and Oleg A. Prokopyev

Part II Local Search **221**

9 Constraint-Based Local Search 223
Laurent Michel and Pascal Van Hentenryck

10 Guided Local Search 261
Abdullah Alsheddy, Christos Voudouris, Edward P. K. Tsang, and Ahmad Alhindi

11 Theory of Local Search 299
W. Michiels, E. H. L. Aarts, and J. Korst

12 Variable Neighborhood Descent 341
Abraham Duarte, Nenad Mladenović, Jesús Sánchez-Oro, and Raca Todosijević

Part III Metaheuristics **369**

13 Ant Colony Optimization: A Component-Wise Overview 371
Manuel López-Ibáñez, Thomas Stützle, and Marco Dorigo

14 Evolutionary Algorithms 409
David Corne and Michael A. Lones

15 Genetic Algorithms 431
Carlos García-Martínez, Francisco J. Rodriguez, and Manuel Lozano

16 GRASP 465
Paola Festa and Mauricio G. C. Resende

17 Hyper-heuristics 489
Michael G. Epitropakis and Edmund K. Burke

18 Iterated Greedy 547
Thomas Stützle and Rubén Ruiz

19 Iterated Local Search 579
Thomas Stützle and Rubén Ruiz

20 Memetic Algorithms 607
Carlos Cotta, Luke Mathieson, and Pablo Moscato

21 Particle Swarm Methods 639
Konstantinos E. Parsopoulos

22 POPMUSIC 687
Éric D. Taillard and Stefan Voß

23 Random-Key Genetic Algorithms 703
José Fernando Gonçalves and Mauricio G. C. Resende

24 Scatter Search 717
Rafael Martí, Ángel Corberán, and Juanjo Peiró

25 Tabu Search 741
Manuel Laguna

26 Variable Neighborhood Search 759
Pierre Hansen and Nenad Mladenović

Volume 2

Part IV Analysis and Implementation **789**

27 A History of Metaheuristics 791
Kenneth Sörensen, Marc Sevaux, and Fred Glover

28 Parallel Metaheuristic Search 809
Teodor Gabriel Crainic

29 Theoretical Analysis of Stochastic Search Algorithms 849
Per Kristian Lehre and Pietro S. Oliveto

Part V Applications **885**

30 City Logistics 887
Jaume Barceló, Hanna Grzybowska, and Jesús Arturo Orozco

31 Cutting and Packing 931
Ramón Alvarez-Valdes, Maria Antónia Carravilla, and José Fernando Oliveira

32 Diversity and Equity Models 979
Fernando Sandoya, Anna Martínez-Gavara, Ricardo Aceves, Abraham Duarte, and Rafael Martí

33 Evolutionary Algorithms for the Inverse Protein Folding Problem 999
Sune S. Nielsen, Grégoire Danoy, Wiktor Jurkowski, Roland Krause, Reinhard Schneider, El-Ghazali Talbi, and Pascal Bouvry

34 Linear Layout Problems 1025
Eduardo G. Pardo, Rafael Martí, and Abraham Duarte

35 Maritime Container Terminal Problems 1051
Christopher Expósito-Izquierdo, Eduardo Lalla-Ruiz, Jesica de Armas, Belén Melián-Batista, and J. Marcos Moreno-Vega

36 Metaheuristics for Medical Image Registration 1079
Andrea Valsecchi, Enrique Bermejo, Sergio Damas, and Oscar Cordón

37 Metaheuristics for Natural Gas Pipeline Networks 1103
Roger Z. Ríos-Mercado

38 Network Optimization 1123
Luciana S. Buriol

39 Optimization Problems, Models, and Heuristics in Wireless Sensor Networks 1141
Vinicius Morais, Fernanda S. H. Souza, and Geraldo R. Mateus

40 Particle Swarm Optimization for the Vehicle Routing Problem: A Survey and a Comparative Analysis 1163
Yannis Marinakis, Magdalene Marinaki, and Athanasios Migdalas

41 Scheduling Heuristics 1197
Rubén Ruiz

42 Selected String Problems 1221
Christian Blum and Paola Festa

43 Supply Chain Management 1241
Helena Ramalhinho Lourenço and Martín Gómez Ravetti

44 The Maximum Clique and Vertex Coloring 1259
Oleksandra Yezerska and Sergiy Butenko

45 The Multi-plant Lot Sizing Problem with Multiple Periods and Items 1291
Mariá C. V. Nascimento, Horacio H. Yanasse, and Desiree M. Carvalho

46 Trees and Forests 1307
Andréa Cynthia Santos, Christophe Duhamel, and Rafael Andrade

47 World's Best Universities and Personalized Rankings 1335
Mario Inostroza-Ponta, Natalie Jane de Vries, and Pablo Moscato

Index 1373

About the Editors

Rafael Martí
Statistics and Operations Research Department
University of Valencia
Valencia, Spain

Rafael Martí is Professor of Statistics and Operations Research at the University of Valencia, Spain. He received a doctoral degree in Mathematics from the University of Valencia in 1994. He has done extensive research in metaheuristics for hard optimization problems. Dr. Martí has over 200 papers, half of them in indexed journals (JCR), including EJOR, INFORMS JoC, *IIE Transactions*, JOGO, C&OR, and *Discrete Applied Mathematics*. He is the co-author of several monographic books: *Scatter Search* (Kluwer 2003), *The Linear Ordering Problem* (Springer 2011), and *Metaheuristics for Business Analytics* (Springer 2018), and has secured a US patent.

Prof. Martí is currently Area Editor in the *Journal of Heuristics* and Associate Editor in the *Math. Prog. Computation*, and the *Int. Journal of Metaheuristics*. He is Senior Research Associate of OptTek Systems (USA) and has given about 50 invited and plenary talks. Dr. Martí has been invited Professor at the University of Colorado (USA), University of Molde (Norway), University of Graz (Austria), and University of Bretagne-Sud (France). He coordinates the Spanish Network on Metaheuristics, currently funded as a SEIO working group.

Panos M. Pardalos
Department of Industrial
and Systems Engineering
University of Florida
Gainesville, USA

Panos M. Pardalos is a Distinguished Professor and the Paul and Heidi Brown Preeminent Professor in the Departments of Industrial and Systems Engineering at the University of Florida, and a world renowned leader in Global Optimization, Mathematical Modeling, and Data Sciences. He is a Fellow of AAAS, AIMBE, and INFORMS and was awarded the **2013 Constantin Caratheodory** Prize of the International Society of Global Optimization. In addition, Dr. Pardalos has been awarded the **2013 EURO Gold Medal** prize bestowed by the Association for European Operational Research Societies. This medal is the preeminent European award given to Operations Research (OR) professionals for "scientific contributions that stand the test of time." Dr. Pardalos is also a Foreign Member of the Lithuanian Academy of Sciences, the Royal Academy of Spain, and the National Academy of Sciences of Ukraine. He is the Founding Editor of *Optimization Letters*, *Energy Systems*, and Co-Founder of the *International Journal of Global Optimization*. He has published over 500 papers, edited/authored over 200 books, and organized over 80 conferences.

Mauricio G. C. Resende
Amazon.com, Inc. and University of Washington
Seattle, WA, USA

Mauricio G. C. Resende grew up in Rio de Janeiro (BR), West Lafayette (IN-US), and Amherst (MA-US). He did his undergraduate training in electrical engineering (systems engineering concentration) at the Pontifical Catholic University of Rio de Janeiro. He obtained an M.S. in Operations Research from Georgia Tech and a PhD in Operations Research from the University of California, Berkeley. He is most known for his work with metaheuristics, in particular, GRASP and biased random-key genetic algorithms, as well as for his work with interior point methods for linear programming and network flows. Dr. Resende has published over 200 papers on optimization and holds 15 US patents. He has edited four handbooks, including the *Handbook of Heuristics*, the *Handbook of Applied Optimization*, and the *Handbook of Optimization in Telecommunications*, and is coauthor of the book *Optimization by GRASP*. He sits on the editorial boards of several optimization journals, including *Networks*, *J. of Global Optimization*, R.A.I.R.O., and *International Transactions in Operational Research*.

Prior to joining Amazon.com in 2014 as a Principal Research Scientist in the transportation area, Dr. Resende was a Lead Inventive Scientist at the Mathematical Foundations of Computing Department of AT&T Bell Labs and at the Algorithms and Optimization Research Department of AT&T Labs Research in New Jersey. Since 2016, Dr. Resende is also Affiliate Professor of Industrial and Systems Engineering at the University of Washington in Seattle.

Contributors

E. H. L. Aarts Tilburg University, Tilburg, The Netherlands

Ricardo Aceves Department of Systems, Universidad Nacional Autónoma de México, Mexico City, Mexico

Ahmad Alhindi Department of Computer Science, Umm Al-Qura University, Makkah, Saudi Arabia

Abdullah Alsheddy College of Computer and Information Sciences (CCIS), Al-Imam Muhammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia

Ramón Alvarez-Valdes Universitat de València, Burjassot, Spain

Rafael Andrade Departamento de Estatística e Matemática Aplicada, Centro de Ciências, Universidade Federal do Ceará, Fortaleza, Brazil

Jesús Arturo Orozco Department of Operations Management, IPADE Business School, Mexico City, Mexico

Jaume Barceló Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, Barcelona, Spain

Enrique Bermejo Department of Computer Science and Artificial Intelligence, Unviersity of Granada, Granada, Spain

Christian Blum Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), Bellaterra, Spain

Pascal Bouvry Computer Science and Communications (CSC) Research Unit, FSTC, University of Luxembourg, Luxembourg City, Luxembourg

Luciana S. Buriol Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Edmund K. Burke School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

Sergiy Butenko Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX, USA

Maria Antónia Carravilla INESC TEC and Faculty of Engineering, University of Porto, Porto, Portugal

Desiree M. Carvalho Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo – UNIFESP, São Paulo, Brazil

Carlos A. Coello Coello Departmento de Computación, CINVESTAV-IPN, Mexico City, México

Ángel Corberán Facultat de Ciències Matemàtiques, Departament d'Estadística i Investigació Operativa, Universitat de València, València, Spain

Oscar Cordón Department of Computer Science and Artificial Intelligence, Unviersity of Granada, Granada, Spain

David Corne Heriot-Watt University, Edinburgh, UK

Carlos Cotta Departamento Lenguajes y Ciencias de la Computación, Universidad de Málaga, Málaga, Spain

Andréa Cynthia Santos ICD-LOSI, UMR CNRS 6281, Université de Technologie de Troyes, Troyes Cedex, France

Sergio Damas Department of Software Engineering, University of Granada, Granada, Spain

Grégoire Danoy Computer Science and Communications (CSC) Research Unit, FSTC, University of Luxembourg, Luxembourg City, Luxembourg

Jesica de Armas Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Simone de Lima Martins Instituto de Ciência da Computação, Niterói, Rio de Janeiro, Brazil

Marco Dorigo IRIDIA, Université libre de Bruxelles(ULB), Brussels, Belgium

Abraham Duarte Department of Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain

Christophe Duhamel LIMOS-UBP, UMR CNRS 6158, Université Blaise Pascal, Aubière Cedex, Clermont-Ferrand, France

Michael Emmerich Leiden, Netherlands

Michael G. Epitropakis Data Science Institute, Department of Management Science, Lancaster University Management School, Lancaster University, Lancaster, UK

Christopher Expósito-Izquierdo Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

José Fernando Oliveira INESC TEC and Faculty of Engineering, University of Porto, Porto, Portugal

Paola Festa Department of Mathematics and Applications "Renato Caccioppoli", University of Napoli Federico II, Napoli, Italy

Martina Fischetti Technical University of Denmark, Kongens Lyngby, Copenhagen, Denmark

Vattenfall BA Wind, Kongens Lyngby, Copenhagen, Denmark

Matteo Fischetti DEI, University of Padova, Padova, Italy

Teodor Gabriel Crainic CIRRELT – Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, QC, Canada

School of Management, Université du Québec à Montréal, Montréal, QC, Canada

Carlos García-Martínez Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain

Fred Glover OptTek Systems, Inc, Boulder, CO, USA

Martín Gómez Ravetti Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

José Fernando Gonçalves INESC TEC, Porto, Portugal

Faculdade de Economia da, Universidade do Porto, Porto, Portugal

Hanna Grzybowska School of Civil and Environmental Engineering, Research Centre for Integrated Transport Innovation, University of New South Wales, Sydney, NSW, Australia

Pierre Hansen GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

Leticia Hernando Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Donostia, Spain

Mario Inostroza-Ponta Departamento de Ingeniería Informática, Universidad de Santiago, Santiago, Chile

Natalie Jane de Vries School of Electrical Engineering and Computing, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, Australia

Wiktor Jurkowski The Genome Analysis Centre (TGAC), Norwich Research Park, Norwich, UK

J. Korst Philips Research Laboratories, Eindhoven, The Netherlands

Roland Krause Luxembourg Centre for Systems Biomedicine (LCSB), University of Luxembourg, Luxembourg City, Luxembourg

Per Kristian Lehre School of Computer Science, University of Birmingham, Birmingham, UK

Manuel Laguna Leeds School of Business, University of Colorado Boulder, Boulder, CO, USA

Eduardo Lalla-Ruiz Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Michael A. Lones Heriot-Watt University, Edinburgh, UK

Manuel López-Ibáñez Alliance Manchester Business School, University of Manchester, Manchester, UK

Jose A. Lozano Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Donostia, Spain

Manuel Lozano Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain

Magdalene Marinaki School of Production Engineering and Management, Technical University of Crete, Chania, Greece

Yannis Marinakis School of Production Engineering and Management, Technical University of Crete, Chania, Greece

Rafael Martí Statistics and Operations Research Department, University of Valencia, Valencia, Spain

Anna Martínez-Gavara Departamento de Estadística e Investigación Operativa, Facultad de Matematicas, Universidad de Valencia, Valencia, Spain

Geraldo R. Mateus Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Luke Mathieson Centre for Bioinformatics, Biomarker Discovery and Information-Based Medicine, University of Newcastle, Callaghan, NSW, Australia

Belén Melián-Batista Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Alexander Mendiburu Intelligent Systems Group, Department of Computer Architecture and Technology, University of the Basque Country, Donostia, Spain

Laurent Michel University of Connecticut, Storrs, CT, USA

W. Michiels NXP, Eindhoven, The Netherlands

Eindhoven University of Technology, Eindhoven, The Netherlands

Athanasios Migdalas Industrial Logistics, Luleå Technical University, Luleå, Sweden

Department of Civil Engineering, Aristotle University of Thessalonike, Thessalonike, Greece

Nenad Mladenović GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

LAMIH, University of Valenciennes, Famars, France

LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia

Vinicius Morais Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

J. Marcos Moreno-Vega Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Pablo Moscato School of Electrical Engineering and Computing, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, Australia

Mariá C. V. Nascimento Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo – UNIFESP, São Paulo, Brazil

Sune S. Nielsen Computer Science and Communications (CSC) Research Unit, FSTC, University of Luxembourg, Luxembourg City, Luxembourg

Pietro S. Oliveto Department of Computer Science, University of Sheffield, Sheffield, UK

Eduardo G. Pardo Dept. of Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, Spain

Konstantinos E. Parsopoulos Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

Juanjo Peiró Facultat de Ciències Matemàtiques, Departament d'Estadística i Investigació Operativa, Universitat de València, València, Spain

Nelishia Pillay School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban, South Africa

Alexandre Plastino Instituto de Ciência da Computação, Niterói, Rio de Janeiro, Brazil

Oleg A. Prokopyev University of Pittsburgh, Pittsburgh, PA, USA

Helena Ramalhinho Lourenço Universitat Pompeu Fabra, Barcelona, Spain

Mauricio G. C. Resende Amazon.com, Inc. and University of Washington, Seattle, WA, USA

Roger Z. Ríos-Mercado Graduate Program in Systems Engineering, Universidad Autónoma de Nuevo León (UANL), San Nicolás de los Garza, Mexico

Francisco J. Rodriguez Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain

Isabel Rosseti Instituto de Ciência da Computação, Niterói, Rio de Janeiro, Brazil

Rubén Ruiz Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politécnica de València, València, Spain

Jesús Sánchez-Oro Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain

Fernando Sandoya Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador

Reinhard Schneider Luxembourg Centre for Systems Biomedicine (LCSB), University of Luxembourg, Luxembourg City, Luxembourg

Marc Sevaux Université de Bretagne-Sud, Lab-STICC, CNRS, Lorient, France

Ofer M. Shir Computer Science Department, Tel-Hai College, Upper Galilee, Kiryat Shmona, Israel

MIGAL Galilee Research Institute, Upper Galilee, Kiryat Shmona, Israel

Oleg V. Shylo University of Tennessee, Knoxville, TN, USA

Kenneth Sörensen University of Antwerp, Antwerp, Belgium

Fernanda S. H. Souza Departamento de Ciência da Computação, Universidade Federal de São João del-Rei, São João del-Reil, Brazil

Thomas Stützle IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium

Éric D. Taillard Embedded Information Systems, HEIG-VD, University of Applied Sciences and Arts of Western Switzerland, Yverdon-les-Bains, Delémont, Switzerland

El-Ghazali Talbi Université des sciences et technologies de Lille, INRIA Lille Nord Europe, Villeneuve d'Ascq, France

Raca Todosijević LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia

Edward P. K. Tsang Department of Computer Science, University of Essex, Colchester, UK

Andrea Valsecchi Department of Computer Science and Artificial Intelligence, Unviersity of Granada, Granada, Spain

Pascal Van Hentenryck University of Michigan, Ann Arbor, MI, USA

Stefan Voß Faculty of Business Administration, Institute of Information Systems, University of Hamburg, Hamburg, Germany

Christos Voudouris Department of Computer Science, University of Essex, Colchester, UK

Hao Wang Leiden Institute of Advanced Computer Science, Leiden University, Leiden, Netherlands

Horacio H. Yanasse Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo – UNIFESP, São Paulo, Brazil

Oleksandra Yezerska Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX, USA

Part I
Search Strategies

Adaptive and Multilevel Metaheuristics

1

Marc Sevaux, Kenneth Sörensen, and Nelishia Pillay

Contents

Introduction 4
Definitions 5
Configuring a Metaheuristic 7
Adaptive Metaheuristics 10
  Simple Adaptive Mechanisms 10
  Reactive Search 11
  Greedy Randomized Adaptive Search Procedure 11
  Adaptive Large Neighborhood Search 12
Multilevel Metaheuristics and Hyper-heuristics 13
  Hyper-heuristics 14
  Hyper-heuristics for Metaheuristic Configuration 15
  Discussion 17
Conclusion 19
Cross-References 19
References 19

M. Sevaux (✉)
Université de Bretagne-Sud, Lab-STICC, CNRS, Lorient, France
e-mail: marc.sevaux@univ-ubs.fr

K. Sörensen
University of Antwerp, Antwerp, Belgium
e-mail: kenneth.sorensen@uantwerpen.be

N. Pillay
School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban, South Africa
e-mail: pillayn32@ukzn.ac.za

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_16

Abstract

For the last decades, metaheuristics have become ever more popular as a tool to solve a large class of difficult optimization problems. However, determining the best configuration of a metaheuristic, which includes the program flow and the parameter settings, remains a difficult task. Adaptive metaheuristics (that change their configuration during the search) and multilevel metaheuristics (that change their configuration during the search by means of a metaheuristic) can be a solution for this. This chapter intends to make a quick review of the latest trends in adaptive metaheuristics and in multilevel metaheuristics.

Keywords

Metaheuristics · Multilevel · Adaptive · Configuration · Hyper-heuristics

Introduction

Metaheuristics are flexible frameworks that can be used to design heuristics for virtually any combinatorial optimization problem. This flexibility also comes at a cost: many researchers in metaheuristics spend a large amount of time to properly design and tune their algorithm in a trial-and-error fashion. As mentioned and observed in many published papers, designing an efficient metaheuristic is an art, requiring a lot of intuition on the part of the metaheuristic designer. There is no doubt, however, that the parameters and the structure of the metaheuristic may influence the performance of the solution approach and the quality of the results in the end.

In the design of a metaheuristic, a large fraction of the time is usually spent on determining the *control flow*, i.e., the order in which the different components of the metaheuristic are used and the optimal levels of the various parameters of the metaheuristic. A more structured approach than the one commonly used is wanted. The purpose of this chapter is to discuss one of the ways to alleviate this problem, through adaptive and multilevel metaheuristics.

Our overview is necessarily very short and cannot replace the many years of research, the large number of books and papers that have tried to clarify the topic. We encourage the reader to address the existing work that we point out and explore the references that we may have missed here. This chapter is not a complete review of all papers in the field of adaptive or multilevel metaheuristics but rather a set of good practices that can be done when designing metaheuristics.

In a first section, we will attempt to clearly define some concepts to come to a definition of the terms *adaptive* and *multilevel* (section "Definitions"). We will also discuss the impact of heuristic parameters on the behavior of a metaheuristic algorithm (section "Configuring a Metaheuristic") and how we can reduce the number of parameters or how we can automatically tune some parameters. Adaptive metaheuristics are discussed in section "Adaptive Metaheuristics", and multilevel metaheuristics and hyper-heuristics in section "Multilevel Metaheuristics and Hyper-heuristics". Section "Conclusion" will conclude the chapter.

Definitions

The terms *adaptive* and *multilevel* do not have a single, generally accepted definition within the metaheuristics community. The aim of this section is to develop clear definitions for both. Both *adaptive* and *multilevel* refer to the evolution of the *configuration* of a metaheuristic algorithm during the optimization process. We must therefore first define the term *configuration*. Our definitions will be intuitive rather than formal.

Each metaheuristic algorithm consists of several *components*, i.e., parts that form a more or less logical and atomic unit. Examples are a local search operator in a variable search algorithm, a tabu list in a tabu search algorithm, a crossover operator or a selection operator in an evolutionary algorithm. Each of these components can exist more or less independently of the rest of the metaheuristic algorithm, which includes their use in a different metaheuristic algorithm for the same problem. Within the same metaheuristics, components can often be rearranged in the overall structure of the algorithm. For example, local search operators in a variable neighborhood search algorithm can be executed in a specific order; an evolutionary algorithm may use its selection operator before or after the crossover operator (or both), etc. In programming, this is called the *control flow*, a term we will adopt here. Clearly, not every control flow makes sense, but generally speaking, a sizeable number of possibilities exists. Determining the control flow of the algorithm, i.e., the order in which the components are executed is a task for the algorithm designer.

Each component may have one or more parameters that determine its functioning. Such parameters may be numerical, such as the tabu tenure of a tabu list, the number of iterations without improvement before a perturbation move is used, etc. Other parameters may be nonnumerical, like the choice of a discrete set of move strategies to use in a local search operator (steepest descent, mildest descent, random improving, ...). The algorithm designer usually defines a finite set of potential (or sensible) values, e.g., the restricted candidate list of a GRASP algorithm is defined to be an integer number between 5 and 20. In some situations, a parameter might also be a real number.

Using these concepts, we may now define the term *configuration*.

Definition 1. Given the set of components of a metaheuristic algorithm together with the set of all potential control flow alternatives, as well as its parameters and their potential values, the **configuration** of a metaheuristic algorithm defines the specific control flow and the specific set of parameter values it uses.

For example, a local search-based metaheuristic like the most basic tabu search method depicted in Algorithm 1 has one parameter, the tabu tenure (the length of the tabu list), and needs one type of neighborhood. It also needs several functions such as *initialize* or *update memory*. One specific tabu tenure, one type of neighborhood and the set of necessary functions will be the configuration of the current search method.

Algorithm 1: Basic tabu search

initialise: find an initial solution x
repeat
  neighbourhood search: find a solution $x' \in \mathcal{N}^*(x)$
  update memory: tabu list, frequency-based memory, aspiration level, ...
  move $x \leftarrow x'$
until *stopping criterion satisfied*

For a simple variable neighborhood search heuristic presented in Algorithm 2, the order in which neighborhoods are explored (or whether they are explored at all), the way in which the starting solution in the current neighborhood is generated and the local search method that is applied to improve the solution with the current neighborhood, as well as the total number of iterations k_{max}, define the configuration of the VNS.

Algorithm 2: Basic variable neighbourhood search

initialise: find an initial solution x, k $\leftarrow 1$
repeat
  shake: generate a point x' at random from the neighbourhood $\mathcal{N}_k(x)$
  local search: apply a local search procedure starting from the solution x' to find a solution x''
  if x'' *is better than* x **then**
    $x \leftarrow x''$ and $k \leftarrow 1$ (centre the search around x'' and search again with neighbourhood 1)
  else
    $k \leftarrow k + 1$ (enlarge the neighbourhood)
until $k = k_{max}$

We can now define the terms *adaptive* and *multilevel* in the context of metaheuristic algorithms.

Definition 2. A metaheuristic is **adaptive** when it includes a mechanism to modify its configuration *during* its execution.

In other words, an adaptive heuristic includes a mechanism to modify either the control flow or the parameter values (or both) of a heuristic, and, by doing so modify the behavior of the metaheuristic. As an example, consider the basic tabu search shown in Algorithm 1. A common adaptive mechanism might make some changes to the *update memory*: the length of the tabu list, the aspiration level, etc.

Even though the definition has been conceived to be as watertight as possible, there is always room for interpretation. For example, considering the VNS heuristic in Algorithm 2, the basic design of the heuristic is such that the order in which the neighborhoods are searched depends on the instance being solved. Since the heuristic will move to the next neighborhood once a local optimum has been reached (and local optima are different for every instance), the control flow of the algorithm will be different for every instance. Yet, most researchers would not call a simple VNS heuristic adaptive.

The mechanism that does the actual adaptation can range from very simple to complex. Essentially, the aim of the mechanism is to search for the best configuration of the metaheuristic algorithm. This search can itself be seen as a combinatorial optimization problem, and an optimization algorithm may be used to solve it. When the adaptation mechanism itself is a metaheuristic algorithm, we call the overall result a *multilevel metaheuristic*.

Definition 3. A **multilevel** metaheuristic algorithm is a metaheuristic algorithm for which the configuration is altered by another metaheuristic algorithm.

Note that, since the algorithm doing the adapting is itself a heuristic, with its own components and parameters, the road is paved for a recursive structure in which the configuration of the lowest-level heuristic is adapted by a higher-level metaheuristic; the configuration of which is adapted by an even higher-level heuristic, the configuration of which is adapted by ..., ad infinitum. However, the complexity added by implementing a higher-level metaheuristic algorithm to adapt the configuration of a lower-level metaheuristic is usually considerable, which precludes the design of a multilevel heuristic having many levels.

For reasons of clarity, a hybrid metaheuristic, i.e., a metaheuristic algorithm that combines ideas from several metaheuristic frameworks, for which the configuration remains unchanged throughout the search, is not considered a multilevel metaheuristic in this chapter. Also, the term multilevel is often used to denote optimization problems that can decompose into several (simpler) problems (e.g., a location–routing problem can often be decomposed into a location problem and a routing problem). Each of the problems may be separately solved by a metaheuristic algorithm. In this chapter, we do not use the term multilevel metaheuristics to describe such approaches.

Configuring a Metaheuristic

An optimization method and especially a metaheuristic has several (potentially hundreds) possible configurations. Among all of them, only a few will allow the search to reach the optimal solution or the best possible solution, but not for all instances and not at all time. The configuration may have a great influence on the quality of the final solution or the effectiveness of the search method. This is the reason why properly tuning the parameters or choosing the right configuration is

a very important task. Because of the "no free lunch theorem" [35], we know that there is no optimal configuration of a metaheuristic that will outperform all others on all problems and all instances.

Despite this fact, it is necessary to find an initial configuration suitable to solve the problem at hand. This initial part will be described in this section. If the configuration is not satisfying, instead of starting again the resolution with a new configuration, one can change the configuration during the search (see section "Adaptive Metaheuristics").

As noted by several researchers (e.g., [14]), the first step of setting up a metaheuristic has to go through the configuration phase during which the control flow is established and the parameters are tuned. This phase is sometimes called "offline parameter initialization." This is long and fastidious and usually done with trial-and-error methods. Moreover, even when this step is completed, its efficiency is often effective on a subset of instances (usually close to the instances on which the parameters have been calibrated). In addition, as already mentioned for metaheuristics, the parameters are not only numerical values but can be search components, updating function, etc. [32].

As a general observation, tuning these parameters is often so difficult that the designers change them one by one until they get the right configuration. And the value of these parameters is obtained empirically. Hence, the final combination of the parameters deduced from a sequential empirical adaptation of the parameters cannot guarantee that the final configuration is optimal. Furthermore, by changing the parameters one by one, it is impossible (or too difficult) to detect the possible interactions between these parameters. Moreover, the parameters and the control flow of a heuristic generally heavily influence each other, which makes the process of determining both even more difficult.

Eiben et al. [17] clearly define the parameter tuning for evolutionary algorithms: *By parameter tuning we mean the commonly practiced approach that amounts to finding good values for the parameters before the run of the algorithm and then running the algorithm using these values, which remain fixed during the run.*

Once this initial tuning phase is completed, most metaheuristic designers keep the configuration as it is to run their solution approach on the set of instances studied. This configuration remains the same (as cited above) until the designer believes that it is not adapted anymore and should be reconfigured with the same process.

Among the potential methods for tuning the parameters before solving, one can list:

- Manual tuning (usually from the experience of a metaheuristic designer),
- Parameter tuning on a subset of representative instances,
- Automatic parameter tuning by the use of an external method.

An experienced metaheuristic designer is often able to decide the value of a large number of parameters beforehand. The rules of thumb prevail on every other considerations. The reason for this is that after several years of practicing, one can

know that some parameters need to have certain values, and the values that are close to it will not make a big difference on the final effectiveness of the results.

The population of a genetic algorithm is a rather good example. A small population will have a premature convergence because too many individuals will be the same (clones), and to avoid this without a specific mechanism, it is important to have a population of a large size. Many papers on genetic algorithms have a population of 100 individuals, but none mentioned how they have obtained this value, or the motivations to set it to this value. No analysis is done to see if 95 or 105 will give better results.

Only a few researchers report the difficulty of finding the right parameter settings and the limitations of this kind of approach [34]. Moreover, the manual tuning, without post-analysis experiments, has the drawback of not being applicable to different instances than the one presented in the paper. This is even more the case for transferring the method to any similar industrial application.

Whatever the technique used for tuning these parameters or configurations beforehand, it is important to keep in mind that every metaheuristic should be well balanced between *intensification* and *diversification* [31]. Hence, the tuning of the parameters should take this into account for ensuring that the metaheuristic is not converging too fast (too much intensification or exploitation) or is wandering in the search space without converging (too much diversification or exploration). Of course, for being able to detect this, one has to set up some indicators showing the speed of convergence, the evolution of the solution quality, the evolution of the solutions themselves, etc.

The best practice is to report the results as Prins [26] has done in his paper on the vehicle routing problem. In that paper, the tables present at the same time the results obtained on a set of instances with some "standard parameter settings" and the results obtained with the "best parameter settings." This is a fair comparison to existing work. The only drawback is the missing information on the time needed to set up the standard parameter settings as well as the best parameter settings. This can be a long and fastidious task and the total computational time can be high.

Setting up the parameters of a metaheuristic based on preliminary experiments is probably the most common technique used in designing metaheuristics. A subset of representative instances is selected, and the parameters are tested on these instances until they converge to stable results. Hence, they are applied to the whole set of instances, and the results are reported. The subset of instances should be carefully selected to be representative of the future experiments.

Usually, the designer selects one parameter at a time, adjusts its value to the best one, and reiterates with the next parameter. Only a few reports that they practice a full factorial design as stated by Hooker [19]. With such a design, authors may try to understand the relative contribution of each parameter to the global effectiveness of the metaheuristic and the possible influence of the parameters between them. One of the best examples of the application of this type of parameter design is presented by Xu and Kelly [38] on a tabu search algorithm. In their paper, they have selected a small subset of seven instances to tune five components of the tabu search. A more

general approach is presented by Xu et al. [39] but still based on the same factorial analysis.

Another technique to find the right parameter settings is to let an external method to tune the parameters for the metaheuristic designers. Very few studies exist on that issue if they are not used intrinsically during the metaheuristic search. Dean and Voss [10] in their book present a technique known as the *response surface methodology* in statistics. This method has been effectively used in [1]. This technique consists in running a local search method in the space of the parameters. A specific metric measuring the distance between each pair of parameters is calculated by running the metaheuristic. For a fixed setting of parameters (or a point in the search space of the parameters), the neighbors are also explored. If no better neighbor can be found, the value for the parameters is fixed, and the search stops; otherwise, the search continues with the best neighbor and the new parameter settings.

Of course, one cannot guarantee the optimality for all the parameters at once and even at the end of the search. But usually, this technique is able to discover good parameter settings. One important drawback is the definition of the metric that is very sensitive, especially if an order cannot be defined on the variables.

For a more elaborate discussion on this topic, we refer the reader to [3] and to section "Hyper-heuristics for Metaheuristic Configuration" in this chapter.

Adaptive Metaheuristics

As stated in the previous section, once a designer has the best parameter settings for its metaheuristics, he is able to run it confidently on the set of targeted instances and produce results. The question is: "Can it go further?" And the answer is yes. Yes, there is always some space for improvement. A parameter setting that works very well on one instance might work poorly on another one.

To overcome this difficulty, it is always possible to analyze the behavior of the metaheuristic during the search and adapt it to obtain better results. This phase can be named "online parameter tuning." Based on indicators (e.g., convergence, solution quality, similarity of explored solutions), the configuration of the metaheuristic is changed. This technique is particularly appealing when one has to solve only one large instance, and the tuning of parameters cannot be done beforehand.

Simple Adaptive Mechanisms

Detecting why a metaheuristic is not giving satisfying results is not an easy task. It largely depends on the type of metaheuristic itself. For example, identical individuals (clones) in a genetic or memetic algorithm are one of the known consequences of premature convergence. In a local search method, cycling in the objective space or in the solution space is also a situation that needs to be avoided.

The online parameter tuning can be simple as in [30] where a tabu search procedure is having a *cycle detection mechanism* and increases the tabu tenure along the search when cycles are detected. To really improve the behavior of a metaheuristic algorithm, however, more elaborate methods can be used during the search and exploited for a better efficiency.

Such a mechanism includes that of Boutillon et al. [5] that can be activated during the search like *retroactive loops* where a simulated annealing temperature parameter is controlled during the search to follow a predefined probability acceptance decreasing scheme.

Reactive Search

Among all existing methods, the work of Battiti [2] had traced the path a long time ago. In the most simple reactive search, the past history of the search is intensely used for *feedback-based parameter tuning* and for *automated balance of diversification and intensification*. In the former, the tuning of parameters is automated, and decisions on the new values of parameters are made based on the past events of the search. In the latter, the concept of balancing exploration *vs.* exploitation is used to guide the search.

One of the simplest forms of reactive search is reactive tabu search. The main idea is to change the length of the tabu list (i.e., the tabu tenure) based on the search trajectory. Essentially, the tabu list is made longer if the search is not finding better solutions, and shorter if it does.

Greedy Randomized Adaptive Search Procedure

GRASP (greedy randomized adaptive search procedure) is a constructive metaheuristic, the main idea of which is to balance greediness and randomness. Many constructive heuristics are greedy, which means that, at every iteration, they pick the best element from the set of potential solution elements. An example is the nearest-neighbor heuristic for the TSP which starts from a given city and moves to the closest unvisited city at every iteration. The drawback of a fully greedy heuristic is that it only generates a single solution, which is most likely suboptimal. A wrong decision early on in the constructive procedure may lead to bad solutions in the end.

GRASP attempts to overcome this drawback by introducing randomness into the solution construction process. Instead of picking the best element at each iteration, GRASP creates a *restricted candidate list*, i.e., a list of the α best elements and picks one element from this list *at random* (α represents a number of elements). By doing this, GRASP generates a different solution at each iteration. After several iterations, some solutions will likely have been found that are better than the one found by a purely greedy heuristic.

In *reactive* GRASP, introduced by Prais and Ribeiro [25], the parameter α is randomly chosen from a set of discrete value. Initially, each possible α_i has

the same probability of being chosen. The search remembers the quality of the solutions found for each possible value of α_i. After some iterations, the probabilities of selecting α_i are updated to reflect the quality of the solutions it produced. The probabilities corresponding to α_i's that have resulted in good solutions are increased; the others are decreased.

The reactive GRASP by Delorme et al. [11], e.g., works as follows. A *value* λ_i is defined for each α_i. The probability of selecting α_i, p_i is calculated as follows:

$$p_i = \frac{\lambda_i}{\sum_{k=1}^{n} \lambda_k},$$

supposing that n different α_i's have been defined.

Whenever a good solution is found using a certain α_i, this solution is added to the *pool* P_i for this α_i. Periodically, the values of λ_i (and hence p_i) are updated according to the following formula:

$$\lambda_i = \left(\frac{\text{mean}_x \in P_i [f(x) - f(\underline{x})]}{f(\overline{x}) - f(\underline{x})} \right)^{\delta},$$

where $\underline{x}$ and $\overline{x}$ are the worst and best solutions found so far and δ is a parameter introduced to attenuate the update of the probabilities p_i.

Adaptive Large Neighborhood Search

Large neighborhood search (LNS) is a constructive metaheuristic that works by building solutions from their constituting elements. For this purpose, it relies on a set of simple constructive procedures to build solutions and on a set of destructive procedures to partially destroy these solutions so they can be rebuilt. For this reason, LNS has also been called ruin-and-recreate. At each iteration, LNS selects a pair consisting of one destructive heuristic and one constructive heuristic. Using this pair, a new solution is obtained. Most LNS implementations use a probabilistic mechanism to select both the destructive and the constructive heuristic at each iteration. A (nonadaptive) LNS algorithm could, e.g., assign equal probabilities to each constructive heuristic and to each destructive heuristic.

Adaptive large neighborhood search goes a step further by selecting the heuristic pair with a probability determined by the previous performance of both the constructive and the destructive heuristics. As a result, constructive and destructive heuristics that perform well will have a higher probability of being selected, whereas those that will not have a lower probability. Usually, however, the probabilities are bounded by some values that ensure all heuristics have at least a tiny chance of being selected.

An example of an ALNS adaptive constructive/destructive heuristic selection mechanism is the following. Suppose an ALNS heuristic has n constructive and m destructive heuristics. Initially, each constructive heuristic i (and each destructive

heuristic j) is assigned a *value* $\lambda_i^c = 1$ ($\lambda_j^d = 1$). Then, a constructive heuristic is selected from the set of constructive heuristics with a probability p proportional to its value:

$$p_i = \frac{\lambda_i}{\sum_{k=1}^{n} \lambda_k}$$

Similarly, a destructive heuristic is selected according to an equivalent rule.

Using the pair of destructive and constructive heuristic, a new solution is generated. The quality of the new solution is evaluated, and the values of the selected constructive and destructive heuristic are updated.

$$\alpha = \begin{cases} 0.5 & \text{if the new solution is worse than the current solution} \\ 1.5 & \text{if the new solution is better than the current solution} \\ 2 & \text{if the new solution is better than the global best solution} \end{cases}$$

$$\lambda_{i,\text{new}}^c = \gamma \lambda_i^c + (1 - \gamma)\alpha$$

$$\lambda_{j,\text{new}}^d = \gamma \lambda_j^d + (1 - \gamma)\alpha$$

where γ is a parameter between 0 and 1.

Using the formulas above, the probabilities of selection for each constructive and destructive heuristic will adapt to the problem at hand. Moreover, using the formulas above, the values can never increase above 2 and never drop below 0.5. In other words, all constructive and destructive heuristics will keep having a positive probability of being selected, even if they consistently fail to find improving solutions.

Multilevel Metaheuristics and Hyper-heuristics

In the previous section, we have examined different techniques for adapting the configuration of metaheuristics. In this section, we look at multilevel metaheuristics, i.e., metaheuristic algorithms for evolving the configuration of a metaheuristic. Much of the research in this area has used evolutionary algorithms to configure metaheuristics with one of the earliest studies being that conducted by Bölte and Thonemann [4] which uses genetic programming for generating annealing schedules, which were previously manually created in a simulated annealing algorithm to solve the quadratic assignment problem. Various evolutionary algorithms, namely, genetic algorithms, evolutionary strategies, and estimation of distribution algorithms, have been used for parameter tuning of evolutionary algorithms [16]. These are referred to as meta-EAs and operate on the design level, while the EA solving the problem at hand is considered to form the algorithm layer. Hyper-heuristics are proving to be effective for the automatic configuration of metaheuristics, and we provide an overview of the use of hyper-heuristics for this purpose.

Hyper-heuristics

Hyper-heuristics were initially introduced as "heuristics to choose heuristics" [6,28]. Hyper-heuristics aim at providing a more generalized solution for a problem domain rather than producing the best solution for certain problem instances. This is achieved by exploring a space of low-level heuristics rather than a solution space directly. The low-level heuristics can be constructive or perturbative. Constructive low-level heuristics are used to create an initial solution, while perturbative heuristics are used to improve an existing candidate solution. The first generation of hyper-heuristics was essentially selection hyper-heuristics which chose which constructive or perturbative heuristics to use at a particular point in constructing or improving a candidate solution, respectively.

Selection constructive hyper-heuristics are used to select the low-level construction heuristic at each stage in constructing a solution. Similarly, in the case of selection perturbative hyper-heuristics, the hyper-heuristic chooses a perturbative low-level heuristic at each stage of the improvement process. We use an application of hyper-heuristics to the domain of examination timetabling to illustrate these concepts. Low-level construction heuristics generally used to solve examination timetabling problems are the graph coloring heuristics, namely, largest degree, largest weighted degree, largest color degree, largest enrollment, and saturation degree [27]. Each of these heuristics assesses the difficulty associated with allocating an examination to the timetable. For example, the saturation degree heuristic is the number of timetable slots, given the current state of the timetable at the particular point in construction, an examination can be allocated to without causing hard constraint violations such as a student being scheduled to write more than one examination at the same time. A selection hyper-heuristic chooses which of the low-level heuristics to use to schedule each of the examinations. This has proven to be effective as different low-level heuristics work well for different problem instances, and more importantly, different low-level heuristics are more effective at different points of solution construction. Metaheuristics such as simulated annealing, tabu search, variable neighborhood search, and genetic algorithms have generally been used to search the heuristic space [6,9].

Examples of low-level perturbative heuristics for the examination timetabling domain include swapping examinations, swapping rows of the timetable, de-allocating examinations, and allocating examinations. Selection perturbative hyper-heuristics can perform a single point search or a multipoint search. In the case of the former, the hyper-heuristic is comprised of a heuristic selection and move acceptance component [9]. Different techniques are employed to for heuristic selection and move acceptance. These techniques can be as simple as randomly selecting a low-level heuristic and accepting on moves that results in improvement. Metaheuristics can also be employed for heuristic selection and move acceptance, e.g., simulated annealing and tabu search have previously been employed for this purpose. Multipoint search selection perturbation hyper-heuristics employ population-based methods such as evolutionary algorithms and particle swarm optimization to explore

the heuristic space, with the population-based approach, by its nature, performing both the heuristic selection and move acceptance.

As the field developed, the idea of creating new low-level heuristics emerged resulting in a second category of hyper-heuristics, namely, generation hyper-heuristics. Generation hyper-heuristics generate low-level constructive or perturbative heuristics. Genetic programming has primarily been employed by generation hyper-heuristics to create low-level heuristics [8, 9]. An example of a generation hyper-heuristic is that implemented by Burke et al. [7] for the one-dimensional bin-packing problem. Construction low-level heuristics, e.g., first-fit, best-first, and next-fit, are used to choose which bin to allocate an item to. In this study, genetic programming is used to evolve low-level heuristics to decide which bin to allocate an item to. In the study conducted by Sabar et al. [29] grammatical evolution, a variation of genetic programming is used to evolve new low-level perturbative heuristics by combining mechanisms for heuristic selection and move acceptance. Generated low-level heuristics can be reusable or disposable. In the case of reusable heuristics, the generated low-level heuristic created to solve the problem for one instance can be used to solve other problem instances without any regeneration. Disposable low-level heuristics are generated for the specific problem instance and cannot be reused.

Selection and generation hyper-heuristics have generally been used for the automatic configuration of metaheuristic algorithms. In this case, the low-level heuristics represent parameters or operators of the metaheuristic and are essentially perturbative. The hyper-heuristic employs a metaheuristic to explore the space of low-level heuristics. Selection hyper-heuristics are used to determine control flow and for parameter tuning. In this case, the hyper-heuristic selects a component at each point in the application of the algorithm to solve the problem. Furthermore, generation hyper-heuristics are used to create new low-level heuristics; in this context, these represent the components of the metaheuristic.

Hyper-heuristics for Metaheuristic Configuration

Control flow is achieved by producing a combination of low-level heuristics, each is a component of the metaheuristic algorithms, i.e., the hyper-heuristic selects which low-level heuristic to apply at each point in a metaheuristic algorithm. The low-level heuristics are components of the metaheuristic algorithm. Lourenço et al. [21] use grammatical evolution to evolve evolutionary algorithms for the royal road functions. The aim is for the evolutionary algorithm to adapt itself during the evolutionary process. Evaluating the evolutionary algorithm proved to be a computationally intensive task, and hence, a limited number of runs were performed. Grammatical evolution combines the different evolutionary algorithm components, namely, mutation, crossover, and selection components and parameter values for these components. The evolved evolutionary algorithms are applied to a seeded initial population. One instance was used for training, and the evolved

evolutionary algorithms were tested on the remaining four instances. Performance was found to be similar to that of the standard evolutionary algorithm. The evolved EA that performed better than the standard EA did not follow the standard structure and contained two types of crossover. There are a number of early initiatives that can be categorized as hyper-heuristics for inducing evolutionary algorithms, although this is not explicitly stated in the papers. Oltean and Groşan [24] use multigene expression programming to evolve an evolutionary algorithm to induce Griewank's function. Algorithms comprised of initialization, mutation, and crossover components are evolved. The number of crossover, initialization, and mutation components in the best individual was found to increase as the evolution progressed. Linear genetic programming has also been used for purposes of evolutionary algorithm induction [23]. Each algorithm evolved is a generational evolutionary algorithm composed of selection, crossover, and mutation components and is applied to an initial population of randomly generated elements. Evolutionary algorithms are evolved for function optimization, the traveling salesman problem, and quadratic assignment problem. For all three problems, the evolved algorithms are trained on a problem instance and are able to generalize and solve other problem instances. In later work, Dioşan and Oltean [13] use genetic algorithms to evolve evolutionary algorithms for function optimization. Each chromosome is comprised of a combination of selection, mutation, and crossover operations as well as population altering strategies to place the newly created offspring into the population. A different evolutionary algorithm was evolved to induce each of the ten functions. As in previous studies, the crossover operator was the most prevalent in evolved evolutionary algorithms producing the best results.

In some studies, the hyper-heuristic achieves both control flow and parameter tuning. In this case, the hyper-heuristic selects the component of the metaheuristic and the parameter value. Tavares and Pereira [33] employ grammatical evolution to automatically configure ant colonization algorithms for solving the traveling salesman problem. The architecture of the ant colonization algorithm including the components of the algorithm, e.g., method for evaporation, and parameter values are evolved. The evolved architecture was found to be effective when applied to problem instances different from those used for training during evolution. The evolved architectures producing the best results were found to be different from those of the standard ant colonization algorithms. Lourenço et al. [22] use grammatical evolution to design evolutionary algorithms to solve the knapsack problem. The evolved evolutionary algorithms are applied to unseen instances. Each evolved evolutionary algorithm specifies the type of selection, type of crossover, type of mutation, and parameter values. Evolved evolutionary algorithms using binary swap mutation and uniform crossover performed the best.

Generation hyper-heuristics go a step further, and instead of choosing a component of a metaheuristic to decide the control flow, these hyper-heuristics create a new component. In the study conducted by Hong et al. [18] a generative hyper-heuristic, employing genetic programming is used to evolve mutation operators

for evolutionary programming. The terminal set is comprised of random numbers produced by a Gaussian random number generator, and the function set is comprised of arithmetic operators. The approach was used to evolve ten function classes, seven unimodal and three multimodal, and was found to require less processor time than the man hours needed to design new mutation operators. A different mutation operator was evolved for each function class using one instance of the class. Woodward and Swan [37] use local search, namely, hill-climbing, to evolve mutation operators for genetic algorithms. These mutation operators were found to outperform human-created mutation operators. Register machines are used to simulate the behavior of mutation operators. Seven function classes were used to test the effectiveness of the evolved mutation operators. Different mutation operators were created for each function class. Woodward and Swan [36] evaluate a similar approach which uses random search to generate search heuristics for a genetic algorithm. As in the previous study, register machines are used to emulate the selection process. Selection is based on the fitness or rank of a bit string. The evolved selection heuristics were tested for the mimicry problem set and were found to perform better than the human-designed selection heuristics. In the study conducted by Dioşan and Oltean [12], genetic programming induces crossover operators for a genetic algorithm for function optimization. A different crossover operator is evolved for each of the 11 function classes. The performance of the evolved operators was found to be comparative to human-designed crossover operators. Drake et al. [15] also employ grammatical evolution to design the construction heuristic and move operators used by variable neighborhood search to solve the vehicle routing problem. The move operators generated are ruined, and insertion heuristics which are used to perform the shaking process in the variable neighborhood searched. The variable neighborhood search produced results close to the global optimum for all problem instances. Løkketangen and Olsson [20] use the ADATE system to generate the move selection, tabu tenure, and the aspiration criteria in a tabu search for solving the Boolean optimization problem (BOOP). The authors describe ADATE as a generation hyper-heuristic that performs offline learning. ADATE is an automatic programming system that produces functional programs in metalanguage ML. The generated components were found to perform better than manually designed components. One of the generated components producing good results was found to give good moves a longer tabu tenure which is not typical of human-designed tabu searches. This again emphasizes the advantage of automatic generation of metaheuristic components.

Discussion

Multilevel metaheuristics can be categorized as selection or generation perturbative hyper-heuristics. As previously outlined, designing a metaheuristic involves making decisions regarding what parameter values to use, what operators to use, and the control flow of the overall algorithm. From the survey presented above, it is evident

that hyper-heuristics have been fairly effective in making these design decisions. Selection perturbative hyper-heuristics are used to select numerical and discrete parameter values. Hence, the hyper-heuristic explores the space of parameter values. Evolutionary algorithms have chiefly been used for this purpose. The parameter values can either be coevolved while solving the problem at hand or optimized separately. In the latter case, the parameters can be determined offline during the training phase. Selection perturbative hyper-heuristics can also be used to make control flow decisions. This essentially involves selecting different operators at different stages of solving the problem. Hence, the decision of which operator to use can be made as part of the control flow decision. Evolutionary algorithms, including linear genetic programming, genetic algorithms, multigene expression programming, and grammatical evolution have been used for control flow design. The decision regarding which operator to use may not be a matter of selecting an existing operator but creating a new operator. Generation hyper-heuristics can be used to generate new operators. This research has focused to a large extent on selection, mutation, and crossover operators in evolutionary algorithms. Genetic programming has primarily been used to generate new operators. Grammatical evolution, local search, and random search have also been used for this purpose. A hyper-heuristic can be used to make all three design decisions simultaneously. This can be seen in the studies conducted by Lourenço et al. [21, 22] and Tavares and Pereira [33] where grammatical evolution is used to make this decision for the induction of an evolutionary and an ant colonization algorithm simultaneously.

In the case of all three design decisions, the parameter values, operators, and algorithms induced by the hyper-heuristic can be reusable or disposable. In the case of reusability, two phases are performed, a training phase and a testing phase. During the training phase, one or a subset of problem instances are used. The induced parameter values, operators, and algorithms are then used to solve unseen instances. Disposable parameter values, operators, and algorithms are induced for the particular problem instance, and hence, a training phase is not required. Reusability has the advantage of the time required for design being reduced as redesign is not needed for every new instance; however, there may be a limited number of instances for which the generated design is applicable.

One of the challenges associated with using hyper-heuristics for the design of metaheuristics is the processing time needed. Given the advances made in multicore architectures and the availability of multicore architectures on a standard desktop machine, distributed architectures can be designed for the implementation of these hyper-heuristics. Most of the research conducted this far has focused on the configuration of evolutionary algorithms. These ideas can be transferred to the design of other metaheuristics. Evolutionary algorithms have primarily been used for design purposes. These have ranged from genetic algorithms through to grammatical evolution. A comparative study into the performance of the different types of evolutionary algorithms and their contribution to the design process would be interesting.

Conclusion

A metaheuristic algorithm's configuration is the combination of its control flow and its parameter settings. Determining the best possible configuration for a metaheuristic is a difficult task that is commonly done by trial and error and based on the experience of the algorithm designer. For this reason, metaheuristics have been developed that are able to adapt their configuration during the search (adaptive metaheuristics), potentially using a higher-level metaheuristic (multilevel metaheuristics). In this chapter, we have surveyed the literature on this topic. The chapter has also highlighted the effectiveness of (see ▸ Chap. 17, "Hyper-heuristics") as multilevel metaheuristics. This serves as a starting point for researchers wanting to use hyper-heuristics for the automated design of metaheuristics. Hyper-heuristics have been fairly effective for the purpose of design, and in most cases, the generated designs have produced better results than the manually designed metaheuristic, in some experiments producing designs that have not previously been thought of. This overview has also brought to light certain research questions and hence areas for future research. Research thus far has highlighted the potential of hyper-heuristics in the automated configuration of metaheuristics. This has now set the foundation for wider application, including more complex problems.

Cross-References

▸ Hyper-heuristics

References

1. Adenso-Díaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Oper Res 54(1):99–114. https://doi.org/10.1287/opre.1050.0243
2. Battiti R (1996) Reactive search: toward self-tuning heuristics. In: Modern heuristic search methods. Wiley, Chichester, pp 61–83
3. Birattari M (2009) Tuning metaheuristics. Springer, Berlin/Heidelberg. https://doi.org/10.1007/978-3-642-00483-4
4. Bölte A, Thonemann UW (1996) Optimizing simulated annealing schedules with genetic programming. Eur J Oper Res 92(2):402–416. https://doi.org/10.1016/0377-2217(94)00350-5
5. Boutillon E, Roland C, Sevaux M (2008) Probability-driven simulated annealing for optimizing digital FIR filters. In: Studies in computational intelligence. Springer Science & Business Media, pp 77–93. https://doi.org/10.1007/978-3-540-79438-7_4
6. Burke EK, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. International series in operations research & management science, vol 57. Springer, pp 457–474. https://doi.org/10.1007/0-306-48056-5_16
7. Burke EK, Hyde MR, Kendall G, Woodward J (2007) Automatic heuristic generation with genetic programming. In: Proceedings of the 9th annual conference on Genetic and evolutionary

computation – GECCO'07. Association for Computing Machinery (ACM). https://doi.org/10.1145/1276958.1277273
8. Burke EK, Hyde MR, Kendall G, Ochoa G, Ozcan E, Woodward JR (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Intelligent systems reference library. Springer Science & Business Media, pp 177–201. https://doi.org/10.1007/978-3-642-01799-5_6.
9. Burke EK, Gendreau M, Hyde MR, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724. https://doi.org/10.1057/jors.2013.71
10. Dean A, Voss D (eds) (1999) Design and analysis of experiments. Springer, Berlin. https://doi.org/10.1007/b97673
11. Delorme X, Gandibleux X, Rodriguez J (2004) GRASP for set packing problems. Eur J Oper Res 153(3):564–580. https://doi.org/10.1016/s0377-2217(03)00263-7
12. Dioşan L, Oltean M (2006) Evolving crossover operators for function optimization. In: Genetic programming. Springer Science & Business Media, pp 97–108. https://doi.org/10.1007/11729976_9
13. Dioşan L, Oltean M (2009) Evolutionary design of evolutionary algorithms. Genet Program Evolvable Mach 10(3):263–306. https://doi.org/10.1007/s10710-009-9081-6
14. Dobslaw F (2010) A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks. In: Proceedings of the international conference on computer mathematics and natural computing 2010. WASET
15. Drake JH, Kililis N, Ozcan E (2013) Generation of VNS components with grammatical evolution for vehicle routing. In: Genetic programming. Springer Science & Business Media, pp 25–36. https://doi.org/10.1007/978-3-642-37207-0_3
16. Eiben AE, Smit SK (2011) Evolutionary algorithm parameters and methods to tune them. In: Autonomous search. Springer, Berlin/Heidelberg, pp 15–36. https://doi.org/10.1007/978-3-642-21434-9_2
17. Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124–141. https://doi.org/10.1109/4235.771166
18. Hong L, Woodward J, Li J, Ozcan E (2013) Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In: Proceedings of the 16th European conference on genetic programming – EuroGP 2013, vol 7831, pp 85–96
19. Hooker JN (1995) Testing heuristics: we have it all wrong. J Heuristics 1(1):33–42. https://doi.org/10.1007/bf02430364
20. Løkketangen A, Olsson R (2009) Generating meta-heuristic optimization code using ADATE. J Heuristics 16(6):911–930. https://doi.org/10.1007/s10732-009-9119-1
21. Lourenço N, Pereira FB, Costa E (2012) Evolving evolutionary algorithms. In: Proceedings of the fourteenth international conference on genetic and evolutionary computation conference companion – GECCO 2012. ACM Press. https://doi.org/10.1145/2330784.2330794
22. Lourenço N, Pereira FB, Costa E (2013) The importance of the learning conditions in hyper-heuristics. In: Proceedings of the fifteenth annual conference on genetic and evolutionary computation conference – GECCO 2013. ACM Press. https://doi.org/10.1145/2463372.2463558
23. Oltean M (2005) Evolving evolutionary algorithms using linear genetic programming. Evol Comput 13(3):387–410. https://doi.org/10.1162/1063656054794815
24. Oltean M, Groşan C (2003) Evolving evolutionary algorithms using multi expression programming. In: Advances in artificial life. Springer Science & Business Media, pp 651–658. https://doi.org/10.1007/978-3-540-39432-7_70
25. Prais M, Ribeiro CC (2000) Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J Comput 12(3):164–176. https://doi.org/10.1287/ijoc.12.3.164.12639
26. Prins C (2004) A simple and effective evolutionary algorithm for the vehicle routing problem. Comput Oper Res 31(12):1985–2002. https://doi.org/10.1016/s0305-0548(03)00158-8

27. Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2008) A survey of search methodologies and automated system development for examination timetabling. J Sched 12(1):55–89. https://doi.org/10.1007/s10951-008-0077-5
28. Ross P (2005) Hyper-heuristics. In: Search methodologies. Springer Science & Business Media, pp 529–556. https://doi.org/10.1007/0-387-28356-0_17
29. Sabar NR, Ayob M, Kendall G, Qu R (2013) Grammatical evolution hyper-heuristic for combinatorial optimization problems. IEEE Trans Evol Comput 17(6):840–861. https://doi.org/10.1109/tevc.2013.2281527
30. Sevaux M, Thomin P (2001) Heuristics and metaheuristics for parallel machine scheduling: a computational evaluation. In: Proceedings of 4th metaheuristics international conference, MIC 2001, Porto, pp 411–415
31. Sörensen K, Sevaux M (2006) MA|PM: memetic algorithms with population management. Comput Oper Res 33(5):1214–1225. https://doi.org/10.1016/j.cor.2004.09.011
32. Talbi E-G (2009) Metaheuristics: from design to implementation. Wiley & Sons, Hoboken. ISBN:978-0-470-27858-1
33. Tavares J, Pereira FB (2012) Automatic design of ant algorithms with grammatical evolution. In: Genetic programming. Springer Science & Business Media, pp 206–217. https://doi.org/10.1007/978-3-642-29139-5_18
34. Van Breedam A (1995) Improvement heuristics for the vehicle routing problem based on simulated annealing. Eur J Oper Res 86(3):480–490. https://doi.org/10.1016/0377-2217(94)00064-J
35. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893
36. Woodward JR, Swan J (2011) Automatically designing selection heuristics. In: Proceedings of the 13th annual conference companion on genetic and evolutionary computation – GECCO 2011. ACM Press. https://doi.org/10.1145/2001858.2002052
37. Woodward JR, Swan J (2012) The automatic generation of mutation operators for genetic algorithms. In: Proceedings of the fourteenth international conference on genetic and evolutionary computation conference companion – GECCO 2012. ACM Press. https://doi.org/10.1145/2330784.2330796
38. Xu J, Kelly JP (1996) A network flow-based tabu search heuristic for the vehicle routing problem. Transp Sci 30(4):379–393. https://doi.org/10.1287/trsc.30.4.379
39. Xu J, Chiu SY, Glover F (1998) Fine-tuning a tabu search algorithm with statistical tests. Int Trans Oper Res 5(3):233–244. https://doi.org/10.1111/j.1475-3995.1998.tb00117.x

Biased Random-Key Genetic Progamming

2

José Fernando Gonçalves and Mauricio G. C. Resende

Contents

Introduction . . . 24
Program Representation . . . 24
Biased Random-Key Genetic Programming . . . 26
Overview . . . 26
Biased Random-Key Genetic Algorithms . . . 27
Encoding and Chromosome Structure . . . 29
Decoding a Chromosome into a *PE* . . . 30
Fitness Function . . . 33
Examples . . . 34
SR-Example 1 . . . 34
SR-Example 2 . . . 35
Conclusions . . . 36
Cross-References . . . 36
References . . . 36

Abstract

This chapter introduces biased random-key genetic programming, a new metaheuristic for evolving programs. Each solution program is encoded as a vector of random keys, where a random key is a real number randomly generated in the continuous interval [0, 1]. A decoder maps each vector of random keys to a solution program and assigns it a measure of quality. A Program-Expression is encoded in the chromosome using a head-tail representation which is later

J. F. Gonçalves (✉)
INESC TEC, Porto, Portugal

Faculdade de Economia da, Universidade do Porto, Porto, Portugal
e-mail: jfgoncal@fep.up.pt

M. G. C. Resende
Amazon.com, Inc. and University of Washington, Seattle, WA, USA
e-mail: resendem@amazon.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_25

transformed into a syntax tree using a prefix notation rule. The artificial simulated evolution of the programs is accomplished with a biased random-key genetic algorithm. Examples of the application of this approach to symbolic regression are presented.

Keywords
Genetic programming · Biased random-key genetic algorithms · head-tail representation · prefix notation

Introduction

Genetic programming (GP) is an evolutionary metaheuristic inspired by biological evolution to find computer programs that perform a predefined user computational task.

GP has its origins around 1954 with the evolutionary algorithms applied by Nils Aall Barricelli to evolutionary simulations [2]. One of the earliest practitioners of the GP methodology was Lawrence J. Fogel who, in 1964, applied evolutionary algorithms to the problem of discovering finite-state automata [9]. The first paper on tree-based genetic programming was presented in [5] and was later expanded by John R. Koza, who pioneered the application of genetic programming in various complex optimization and search problems [20–23].

Traditionally, GP has favored the use of programming languages that naturally embody tree structures [1], but several new non-tree representations were suggested and successfully implemented, such as linear genetic programming (LGP) [4], gene expression programming (GEP) [6], and Parallel Distributed Graphical Programming (PDGP) [24].

Genetic programming is still a young field of research. It attracts a growing research community, and there are many avenues of research yet to be explored. In this chapter, we introduce biased random-key genetic programming (*BRKGP*), a novel metaheuristic for computer program evolution.

The remainder of the chapter is organized as follows. In section "Program Representation" we describe the program representation. In section "Biased Random-Key Genetic Programming" we introduce the new approach, describing in detail the BRKGA, the chromosome structure and the decoding procedure, and the fitness function. Finally, in section "Examples" we illustrate the application of the approach to two cases in the area of symbolic regression, and in section "Conclusions" we make concluding remarks.

Program Representation

In genetic programming programs are usually expressed by syntax trees (ST). Figure 1 shows the tree representation of the program:

$$\max\left(2.2 - \frac{X}{11}\,,\ 7 \times \cos\left(Y\right)\right).$$

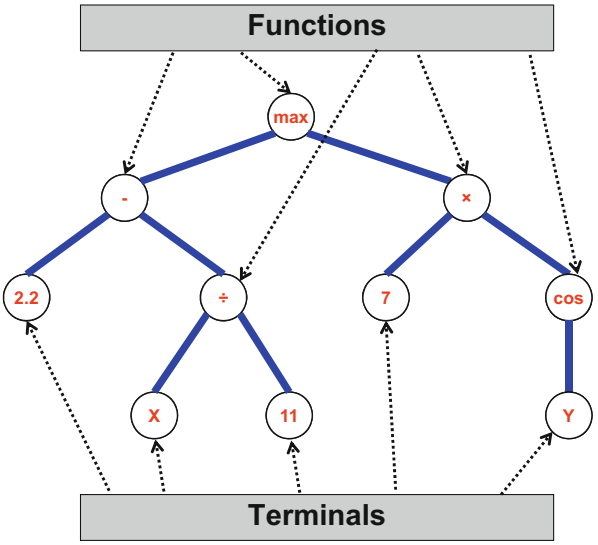


Fig. 1 Syntax tree of program $\max\left(2.2 - \frac{X}{11}\,,\ 7 \times \cos(Y)\right)$

The variables and constants in the program (X, Y, *2.2, 11*, and *7*) are leaves of the tree. In GP the nodes that correspond to the leaves are called *terminals*, and the internal nodes represent the operations ($-$, $\div$, $\times$, cos, and max) and are called *functions*. The sets of allowed functions ($\mathcal{F}$) and terminals ($\mathcal{T}$) together form what is called the *primitive set* ($\mathcal{P}$) of a GP system.

In this chapter a linear representation of syntax trees is used. This indirect representation, called Program-Expression (*PE*), encodes a syntax tree as a sequence of elements of the primitive set which will be later translated, according to some predefined rules, into a syntax tree.

The set of rules used to translate a *PE* into a *ST* follows a prefix notation convention (*P-rule*). The *P-rule* translates a *PE* into a *ST* by reading sequentially, from left to right, each element in the *PE* and placing it in the bottom-most (first-criteria) and left-most (second-criteria) available node in the partial *ST* being constructed. Note that if there are no nodes available, in the partially built syntax tree, in which to place an element in the *PE,* then the process stops and the remaining primitive elements in the *PE* are discarded and are called noncoding elements. Figure 2 demonstrates how the *PE*

$$\wedge + 1 \div r \; n \; n \cos Y \; 17.5$$

can be translated into a *ST* following the *P-rule*. The first primitive element in the *PE* is |∧| and is placed in node 1 (the root of the tree). The second element in the *PE* is |+| and is placed in node 2. The third primitive element in the *PE* is |1| and is placed in node 4. The fourth primitive element in the *PE* is |÷| and is placed in

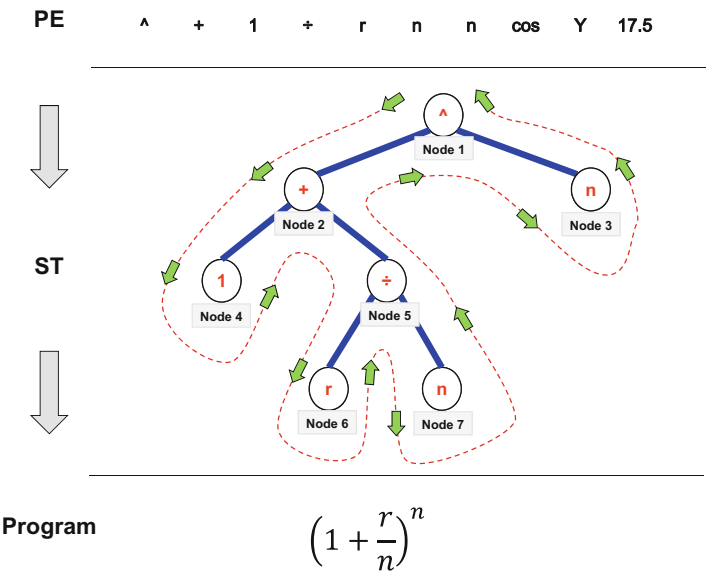


Fig. 2 Translation of a *PE* into a program using the *P-rule*

node 5. This process will be repeated until all elements in the *PE* are placed or there are no nodes available in the tree in which to place an element.

Note that the last three elements of the *PE,* |cos|, |*Y*|, and |*17.5*|, are not included in the *ST*. The program that corresponds to the final *ST* is

$$\left(1+\frac{r}{n}\right)^n.$$

Biased Random-Key Genetic Programming

This section begins with an overview of the proposed biased random-key genetic programming *(BRKGP)* methodology. This is followed by a discussion of the biased random-key genetic programming algorithm, including detailed descriptions of the program encoding and decoding and the fitness measure.

Overview

The *BRKGP* metaheuristic presented in this chapter is based on four main components: a biased random-key genetic algorithm (*BRKGA*) [13], chromosomes, a decoding procedure to translate a chromosome into a syntax tree, and a fitness measure to assess the quality of the resulting programs.

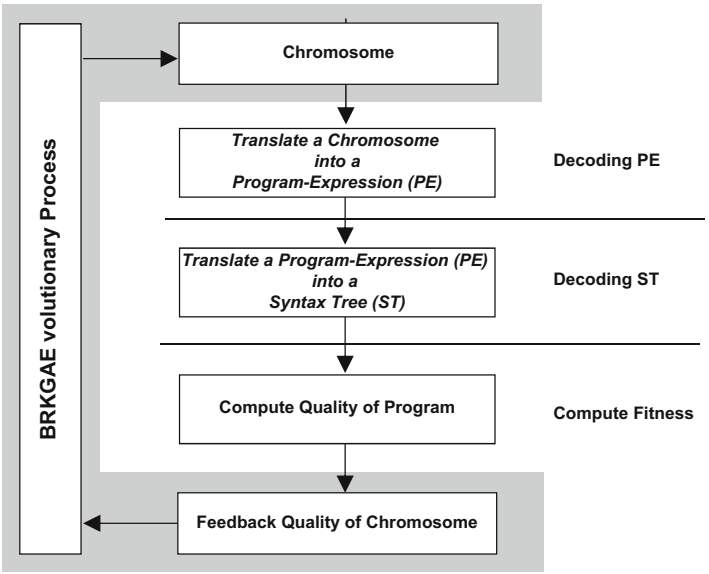


Fig. 3 Architecture of the *BRKGP*

The role of the *BRKGA* is to supply and evolve the chromosomes. A chromosome (genotype) indirectly represents a syntax tree (phenotype) that corresponds to a program. The decoding procedure receives as input a chromosome and in a first phase translates it into a Program-Expression (*PE*). Then, in a second phase the *PE*, obtained in the first phase, is translated into a syntax tree using the *P-rule* described in section "Program Representation."

Figure 3 illustrates the sequence of steps applied by the *BRKGP* to each chromosome.

The remainder of this section describes in detail the biased random-key genetic algorithm, the chromosome structure, the decoding procedure that maps a chromosome into a syntax tree, and the fitness measures.

Biased Random-Key Genetic Algorithms

Genetic algorithms with random keys, or *random-key genetic algorithms* (*RKGA*), were introduced by [3] for solving sequencing problems. In an *RKGA*, chromosomes are represented as vectors of randomly generated real numbers in the interval [0, 1]. A *decoder* is a deterministic algorithm that takes as input a chromosome and associates it with a solution of the combinatorial optimization problem for which an objective value or *fitness* can be computed.

An *RKGA* evolves a *population* of random-key vectors over a number of *generations* (iterations). The initial population is made up of p vectors each with r random keys. Each component of the solution vector, or random key, is randomly generated, independently, in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation g, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e., those with the best fitness values, and the remaining set of $p - p_e$ *nonelite* individuals. To evolve the population of generation g, a new generation $(g + 1)$ of individuals is produced. All elite individuals of the population of generation g are copied without modification to the population of generation $g + 1$. *RKGAs* implement mutation by introducing *mutants* into the population. A mutant is a vector of random keys generated in the same way that an element of the initial population is generated. Its role is similar to that of mutation in other genetic algorithms [11], i.e., to introduce noise into the population and avoid convergence of the entire population to a local optimum. At each generation, a small number p_m of mutants is introduced into the population. With $p_e + p_m$ individuals accounted for in population $g+1$, $p-p_e-p_m$ additional individuals need to be generated to complete the p individuals that make up population $g + 1$. This is done by producing $p - p_e - p_m$ offspring solutions through the process of *mating* or *crossover*.

A *biased random-key genetic algorithm* [13], *or simply BRKGA*, differs from an *RKGA* in the way parents are selected for mating. While in the *RKGA* of [3] both parents are selected at random from the entire current population, in a *BRKGA* each element is generated combining a parent selected at random from the elite partition in the current population and one from the rest of the population. Repetition in the selection of a mate is allowed, and therefore an individual can produce more than one offspring in the same generation. As in a *RKGA*, *parametrized uniform crossover* [25] is used to implement mating in a *BRKGA*. Let ρ_e be the probability that an offspring inherits the vector component of its elite parent. Recall that r denotes the number of components in the solution vector of an individual. For $i = 1, \ldots, r$, the i-th component $c\,[i]$ of the offspring vector c takes on the value of the i-th component $e\,[i]$ of the elite parent e with probability ρ_e and the value of the i-th component $\bar{e}\,[i]$ of the nonelite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, i.e., when it has p individuals, fitness values are computed for all of the newly created random-key vectors, and the population is partitioned into elite and nonelite individuals to start a new generation. Figure 4 depicts the transitional process between two consecutive generations.

A *BRKGA* searches the solution space of the combinatorial optimization problem indirectly by searching the continuous r-dimensional unit hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

*BRKGA*s have been applied with success to solve many types of combinatorial problems (see [10, 12, 14–19]).

A biased random-key genetic algorithm is specified by the parameters, the encoding and decoding of the solutions, and by the fitness measure. In the next

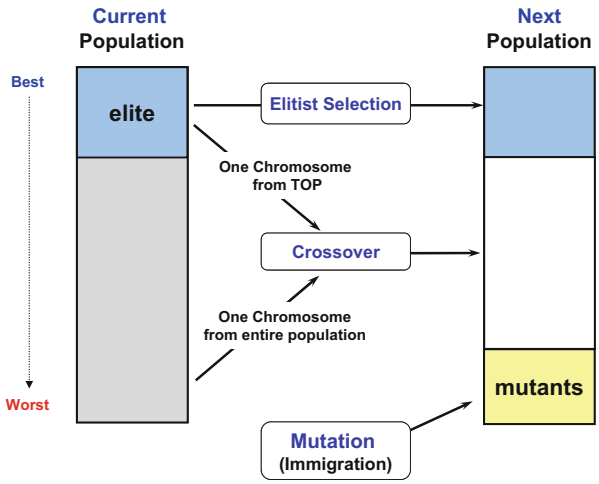


Fig. 4 Transitional process between consecutive generations

section the algorithm is specified by first showing how programs are encoded and then decoded and how their fitness evaluation is computed.

Encoding and Chromosome Structure

A chromosome represents a program and is made of two parts. The first part encodes a *PE* and the second part encodes the values of the constants that can be used in the *PE*. To encode a *PE* a head-tail representation is proposed as in [6–8]. The head, h, represents the maximum number of internal nodes that can be in the syntax tree (*ST*) and can contain both functions (elements from set $\mathcal{F}$) and terminals (elements from the set $\mathcal{T}$), whereas the tail, t, represents the number of leaves of the *ST* and can only contain terminals (elements from the set $\mathcal{T}$). For each problem, the length of the head, h, is chosen, whereas the length of the tail, t, depends on h and on the arity, a, of the function with the most arguments and is determined by the expression

$$t = h\,(a - 1) + 1. \tag{1}$$

This way of determining the length of the tail guarantees that there will always be enough terminals to create syntactically valid *PE*s (assuming closure among the set of primitives).

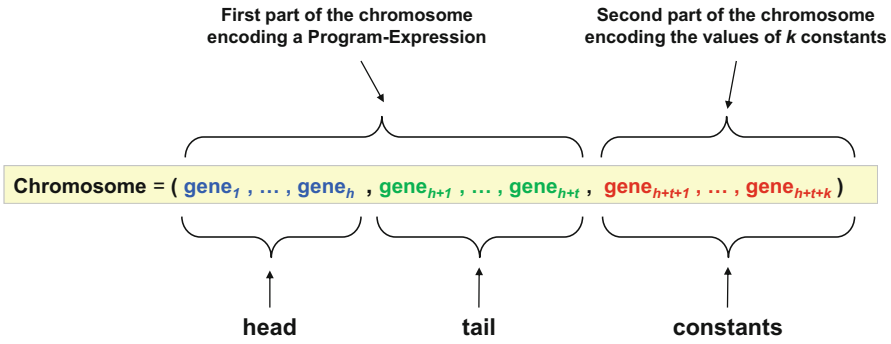


Fig. 5 Chromosome structure used to encode a *PE* with a vector of $h + t + k$ random keys

Table 1 Information relative to the example

$\mathcal{F} = \{+, -, \times, \div, \sqrt{}\}$	$\Longrightarrow$	$vF = (+, -, \times, \div, \sqrt{})\ nF = 5$
$\mathcal{T} = \{X, Y, Z, K_1, K_2\}$	$\Longrightarrow$	$vV = \{X, Y, Z\},\ nV = 3\ vK = \{K_1; K_2\},\ nK = 2$
$h = 5,\ a = 2$	$\Longrightarrow$	$t = 5 \times (2 - 1) + 1 = 6$

Assuming that the terminal set includes k constants, then every chromosome supplied by the *BRKGA* encodes a Program-Expression (*PE*) as a vector of $(h + t + k)$ random keys, and Fig. 5 depicts the corresponding chromosome structure.

Decoding a Chromosome into a *PE*

The decoding of a chromosome into a Program-Expression (*PE*) has the following three steps:

- Decoding the head of the *PE*;
- Decoding the tail of the *PE*;
- Decoding the values of the constants.

Let the terminal set $\mathcal{T}$ be divided into the two mutually exclusive sets $\mathcal{V}$ and $\mathcal{K}$ which represent the set of variables and the set of constants, respectively, and let vF, vV, and vK represent vectors containing all the elements in the sets $\mathcal{F}$, $\mathcal{V}$, and $\mathcal{K}$, respectively. Additionally, let nF, nV, and nK represent the number of elements in the vectors vF, vV, and vK, respectively.

To better illustrate the various steps in the decoding process, an example based on the information presented in Table 1 is used.

According to Table 1 and the chromosome structure defined in the previous section, the chromosome will have 13 $(5 + 6 + 2)$ random keys. Furthermore, assume that the chromosome being decoded is

```
procedure DecodeHead (vF, vV, vK, c, PE)
1   for r = 1, ..., h do
2       idx = ⌈c[r] × (nF + nV + nK)⌉
        // where ⌈x⌉ is the smallest integer greater or equal to x.
        // ** Decoding c[r] → PE[r] **
3       if (idx ≤ nF) then
            // it going to be a function
4           PE[r] = vF[idx];
5       else if (nF < idx ≤ nF + nV) then
            // it going to be a variable
6           PE[r] = vV[idx − nF];
7       else if (nF + nV < idx ≤ nF + nV + nK) ≤then
            // it going to be a constant
8           PE[r] = vK[idx − nF − nV];
9       endif
10  end for
end DecodeHead;
```

Fig. 6 Pseudo-code for the `DecodeHead` procedure

$$c = (0.25,\ 0.35,\ 0.93,\ 0.75,\ 0.05,\ 0.32,\ 0.67,\ 0.58,\ 0.15,\ 0.26,\ 0.86,\ 0.64,\ 0.43)\,. \tag{2}$$

The decoding of the head of the PE is based on the fact that the head can be made of elements in any of the sets $\mathcal{F}$, $\mathcal{V}$, and $\mathcal{K}$. The mapping of the random key in position $r = 1, \ldots, h$ of the chromosome c, $c\,[r]$, into the primitive element in position r of the PE, $PE\,[r]$, is accomplished by the procedure DecodeHead whose pseudo-code is shown in Fig. 6.

Table 2 presents the results of the decoding of the head of the PE corresponding to the first five components of the chromosome given in expression (2).

At this point the first five components of the PE will be

$$\times \div\ K_2\ Z\ +\,.$$

The decoding of the tail of the PE is based on the fact that the tail can only consist of elements in sets $\mathcal{V}$ and $\mathcal{K}$. The mapping of the random key in position

Table 2 Decoding the Head of the *PE*

r	$c[r]$	idx	Head of the *PE*
1	0.25	$\lceil 0.25 \times (5+3+2) \rceil = 3$	$vF\,[3] = \times$
2	0.35	$\lceil 0.35 \times (5+3+2) \rceil = 4$	$vF\,[5] = \div$
3	0.93	$\lceil 0.93 \times (5+3+2) \rceil = 10$	$vK\,[10-5-3] = K_2$
4	0.75	$\lceil 0.75 \times (5+3+2) \rceil = 8$	$vV\,[8-5] = Z$
5	0.05	$\lceil 0.05 \times (5+3+2) \rceil = 1$	$vF\,[1] = +$

```
procedure DecodeTail (vV, vK, c, PE)
1   for r = h + 1, ..., h + t do
2       idx = ⌈c[r] × (nV + nK)⌉
        // where ⌈x⌉ is the smallest integer greater or equal to x.
        // ** Decoding c[r] → PE[r] **
3       if (idx ≤ nV) then
            // it going to be a variable
4           PE[r] = vV[idx];
5       else if (nV < idx ≤ nV + nK) then
            // it going to be a constant
6           PE[r] = vK[idx − nV];
7       endif
8   end for
end DecodeTail;
```

Fig. 7 Pseudo-code for the `DecodeTail` procedure

$r = h+1, \ldots, h+t$ of the chromosome c, $c\,[r]$, into the primitive element in position r of the PE, $PE\,[r]$, is accomplished by the procedure DecodeTail which has the pseudo-code presented in Fig. 7.

Table 3 presents the results of the decoding of the head of the *PE* corresponding to the chromosome given in expression (2).

At this point the components of the *PE* will be

$$\times \;\div\; K_2\; Z\; +\; Y\; K_1\; Z\; X\; Y\; K_2.$$

This *PE* is translated with the *P-Rule* into

Table 3 Decoding the tail of the *PE*

r	$c[r]$	idx	Tail of the *PE*
6	0.32	$\lceil 0.32 \times (3+2) \rceil = 2$	$vV\,[2] = Y$
7	0.67	$\lceil 0.67 \times (3+2) \rceil = 4$	$vK\,[4-3] = K_1$
8	0.58	$\lceil 0.58 \times (3+2) \rceil = 3$	$vV\,[3] = Z$
9	0.15	$\lceil 0.15 \times (3+2) \rceil = 1$	$vV\,[2] = X$
10	0.26	$\lceil 0.26 \times (3+2) \rceil = 2$	$vV\,[4-3] = Y$
11	0.86	$\lceil 0.86 \times (3+2) \rceil = 5$	$vK\,[5-3] = K_2$

$$\frac{K_2}{Z} \times (Y + K_1)\,. \tag{3}$$

Note that the last four components in the *PE* are noncoding elements.

The final step in the decoding process consists in decoding the values of the constants in the terminal set. That is accomplished by using the decoding expression

$$K_i = fk(c\,[h+t+i]\,,\ p_1, p_2, \ldots, p_p) \quad i = 1, \ldots, nK,$$

where $fk()$ is a function that accepts as input a random key, $c\,[r]$, and a set of parameters $p_1, p_2, \ldots, p_p$ and outputs a real or integer value. The simplest case can be obtained when $fk(c\,[r]) = c\,[r]$, i.e., the value of the constant is equal to the value of the random-key input. Many functions can be used for $fk()$. However, the following functions were used:

1. **randReal**$(c\,[r]\,,\ l,\ u) = l + c\,[r] \times (u - l\,)$ – which generates a real value between l and u;
2. **randInt**$(c\,[r]\,,\ l,\ u) = \lfloor\, l + c\,[r] \times (u - l\,)\,\rfloor$ – which generates an integer value between the integers l and u.

For the example, assuming $fk(c\,[r]) = c\,[r]$, the values of the constants K_1 and K_2 are, respectively, $c[5+6+1] = 0.64$ and $c[5+6+2] = 0.43$. The final program can be obtained by replacing the constants K_1 and K_2 in expression (3) by 0.64 and 0.43, respectively, i.e.:

$$\frac{0.43}{Z} \times (Y + 0.64)\,. \tag{4}$$

Fitness Function

The objective of the fitness measure is to assign a quality value to each chromosome so that the evolutionary process differentiates the different chromosomes and directs the search to better solutions (chromosomes). Depending on the problem being solved, different fitness measures can be used.

Suppose that a symbolic regression problem is to be solved where the best fitting function to a set of points is to be found. In this case the usual measures used in the linear or nonlinear regression can be used, least squares, mean squared error, etc. However, if, for example, a rule to decide when to buy or sell a certain stock in the NYSE is to be discovered, then the quality of rule could be evaluated by using historical data, and the profit obtained with the rule, over a certain number of historical days, could be considered the measure of fitness.

Examples

This section presents two examples in the area of symbolic regression (*SR*) to illustrate the application of *BRKGP*. The mean absolute error (MAE)

$$\text{MAE} = \frac{1}{ND} \sum_{d=1}^{d=ND} |f_d - y_d|$$

is used as fitness measure, where ND is the number of data points, y_d is the value of data point d, and f_d is the value obtained by the expression generated by the *BRKGP* for data point d.

SR-Example 1

Table 4 presents 30 data points that are sampled from the function $y = x + 2x^2 + 3x^3 + 4x^4$. *BRKGP* is used to try to discover the polynomial that best fits the data.

The *BRKGP* configuration used for this example is given in Table 5.

Note that, since the best value of h to use is not known, three possibilities for h ($h = 5$, $h = 10$, $h = 15$) are tried. For $h = 5$ and $h = 10$ the *BRKGP* was not able to find the correct polynomial (i.e., MAE > 0). However, with $h = 15$ it was

Table 4 Dataset for *SR*-example 1

x	y	x	y	x	y
1	10	11	62,810	21	806,610
2	98	12	88,428	22	969,958
3	426	13	121,186	23	1,156,946
4	1252	14	162,302	24	1,369,752
5	2930	15	213,090	25	1,610,650
6	5910	16	274,960	26	1,882,010
7	10,738	17	349,418	27	2,186,298
8	18,056	18	438,066	28	2,526,076
9	28,602	19	542,602	29	2,904,002
10	43,210	20	664,820	30	3,322,830

able to find the correct polynomial (i.e., MAE = 0) after 12 generations. The final expression was

$$(((((X + X) \times (X \times X)) + X) \times ((X + X) + (X \div X))) + ((X \times X) \times X))$$

which after being simplified gives the correct polynomial.

SR-Example 2

In this example *BRKGP* is used to try to discover the relation between the $Area$ of the circle and its radius R that best fits the 20 data points presented in Table 6. The 20 data points were sampled from the function $Area = \pi R^2$.

The *BRKGP* configuration used for this example is given in Table 7. Note that, in this case, three constants K_1, K_2, K_3 are included that will be decoded using $fk() = \textbf{randInt}(1, 10000)$.

For $h = 10$ *BRKGP* found, after 67 generations, the expression

$$((R + ((R \div 689) \div 689)) \times (((9151 \div 4273) \times R) + R))$$

which after being simplified is equivalent to $Area = 3.1415933\,R^2$ and has MAE = 0.00027997.

Table 5 *BRKGP* configuration for the *SR*-example 1

$\mathcal{F} = \{+, -, \times, \div\}$	$p = 500$
$\mathcal{T} = \{x\}$	$p_e = 100$
$h = 5, 10, 15$	$p_m = 100$
$a = 2$	$\rho_e = 0.85$
$t = h \times (a - 1) + 1$	Fitness = mean absolute error (MAE)
	Stopping criterion = 100 generations

Table 6 Dataset for *SR*-example 2

R	$Area$	R	$Area$
10	314.1592654	21	1385.44236
11	380.1327111	22	1520.530844
12	452.3893421	23	1661.902514
13	530.9291585	24	1809.557368
14	615.7521601	25	1963.495408
15	706.8583471	26	2123.716634
16	804.2477193	27	2290.221044
17	907.9202769	28	2463.00864
18	1017.87602	29	2642.079422
19	20,1256.637061	30	2827.433388

Table 7 *BRKGP* configuration for the *SR*-example 2

$\mathcal{F} = \{+, -, \times, \div\}$	$p = 500$
$\mathcal{T} = \{R, K_1, K_2, K_3\}$	$p_e = 100$
$fk() = \textbf{randInt}(1, 10000)$	$p_m = 100$
$h = 10, 15$	$\rho_e = 0.85$
$a = 2$	Fitness = mean absolute error (MAE)
$t = h \times (a - 1) + 1$	Stopping criterion = 100 generations

For $h = 15$ *BRKGP* found, after 82 generations, the expression

$$((R \times (((R + R) + R) \times ((476 + 9787) - (6699 \div 476)))) \div 9787)$$

which after being simplified is equivalent to $Area = 3.1415939\,R^2$ and has MAE $= 0.00051594$.

Note that, in both cases, the value of π was approximated to five decimals.

Conclusions

This chapter introduced biased random-key genetic programming, a novel metaheuristic for genetic programming. After introducing how programs are represented using a linear Program-Expression representation, the chapter presents the head-tail encoding and explains how the decoding of the chromosome can be accomplished. The chapter concludes by illustrating how *BRKGP* can be applied to solve problems using two symbolic regression examples.

Cross-References

- Evolutionary Algorithms
- Genetic Algorithms
- Random-Key Genetic Algorithms

Acknowledgments The first author was supported by project PTDC/EGE-GES/117692/2010 funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT – Foundation for Science and Technology.

References

1. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming: an introduction. Morgan Kaufmann, San Francisco
2. Barricelli NA et al (1954) Esempi numerici di processi di evoluzione. Methodos 6(21–22): 45–68

3. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6:154–160
4. Brameier MF, Banzhaf W (2007) Linear genetic programming. Springer, New York
5. Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Proceedings of the first international conference on genetic algorithms, Pittsburg, pp 183–187
6. Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13:87–129
7. Ferreira C (2006) Designing neural networks using gene expression programming. In: Abraham A (ed) Applied soft computing technologies: the challenge of complexity. Springer, Berlin, pp 517–535
8. Ferreira C (2006) Gene expression programming: mathematical modeling by an artificial intelligence. Studies in computational intelligence. Springer, New York
9. Fogel LJ (1964) On the organization of intellect. PhD thesis, UCLA
10. Fontes DBMM, Gonçalves JF (2013) A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. Optimization Letters, 7(6):1303–1324
11. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
12. Gonçalves JF, Almeida J (2002) A hybrid genetic algorithm for assembly line balancing. J Heuristics 8:629–642
13. Gonçalves JF, Resende MGC (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17:487–525
14. Gonçalves JF, Resende MGC (2013) A biased random-key genetic algorithm for a 2D and 3D bin packing problem. Int J Prod Econ 145:500–510
15. Gonçalves JF, Resende MG (2014) An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. Int Trans Oper Res 21(2):215–246
16. Gonçalves JF, Resende MGC (2015) A biased random-key genetic algorithm for the unequal area facility layout problem. Eur J Oper Res 246(1):86–107
17. Gonçalves J, Resende M, Toso R (2014) An experimental comparison of biased and unbiased random-key genetic algorithms. Pesquisa Operacional 34:143–164
18. Gonçalves JF, Resende MGC, Costa MD (2014) A biased random-key genetic algorithm for the minimization of open stacks problem. Int Trans Oper Res. Published online 2 July 2014
19. Gonçalves JF, de Magalhães Mendes JJ, Resende MG (2015) The basic multi-project scheduling problem. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 2. Springer, Berlin, pp 667–683
20. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT, Cambridge
21. Koza JR (1994) Genetic programming II: automatic discovery of reusable subprograms. MIT, Cambridge
22. Koza JR, Bennett FH III, Andre D, Keane MA (1999) Genetic Programming III: Darwinian invention and problem solving. Morgan Kaufmann, San Francisco
23. Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2003) Genetic programming IV: routine human-competitive machine intelligence. Kluwer Academic, Norwell/Dordrecht
24. Poli R (1997) Evolution of graph-like programs with parallel distributed genetic programming. In: Proceedings of the 7th international conference on genetic algorithms (ICGA), East Lansing, pp 346–353
25. Spears WM, DeJong KA (1991) On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms, San Diego, pp 230–236

Data Mining in Stochastic Local Search

3

Simone de Lima Martins, Isabel Rosseti, and Alexandre Plastino

Contents

Introduction . . . 40
Data Mining . . . 42
Strategies to Use Data Mining in Heuristics . . . 44
Memoryless Heuristics . . . 48
  Hybrid DM-GRASP . . . 49
  Hybrid DM-GRASP+VND . . . 58
  Hybrid DM-ILS . . . 63
Memory-Based Heuristics . . . 70
  Hybrid DM-GRASP+PR . . . 71
  Hybrid DM-HMS . . . 79
Conclusion . . . 84
Cross-References . . . 85
References . . . 85

Abstract

This chapter explores some stochastic local search heuristics that incorporate a data mining procedure. The basic idea of using data mining inside a heuristic is to obtain knowledge from previous iterations performed by a heuristic to guide the search in next iterations. Patterns extracted from good quality solutions can be used to guide the search, leading to a more effective exploration of the solution space. This survey shows that memoryless heuristics may benefit from the use of data mining by obtaining better solutions in smaller computational times. Also, some results are revisited to demonstrate that even memory-based heuristics can benefit from using data mining by reducing the computational time to achieve good quality solutions.

S. de Lima Martins (✉) · I. Rosseti · A. Plastino
Instituto de Ciência da Computação, Niterói, Rio de Janeiro, Brazil
e-mail: simone@ic.uff.br; rosseti@ic.uff.br; plastino@ic.uff.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_11

Keywords
Data mining · GRASP · Heuristics · ILS · Path-relinking · VND

Introduction

Combinatorial optimization problems are defined based on an objective function and on logical restrictions such that candidate solutions satisfying the restrictions are called feasible solutions. The optimal solutions, based on maximizing or minimizing the objective function, can be found among the feasible solutions. Many practically relevant combinatorial optimization problems are NP-complete or NP-hard and therefore, in general, are not solved efficiently.

So, one adopted option is to accept suboptimal solutions instead of trying to find optimal solutions. Several computational approaches for getting suboptimal or optimal solutions for hard combinatorial problems can be characterized as search algorithms. The search algorithms iteratively generate and evaluate candidate solutions. Despite that the complexity of time to find optimal solutions can grow exponentially with the size of the instances for NP-hard problems, the evaluation of candidate solutions can often be performed much more efficiently, i.e., in polynomial time [29].

Construction heuristics can be formulated as search procedures, where partial candidate solutions are iteratively extended to obtain complete good candidate solutions aiming to optimize the objective function.

Moreover, local search algorithms begin with a complete candidate solution and subsequently move from the current solution to a neighboring solution.

Many widely known and high-performance local search algorithms make use of randomized choices in generating or selecting candidate solutions for a given combinatorial problem instance. These algorithms are called stochastic local search (or SLS) algorithms [29].

Metaheuristics are general purpose high-level procedures that guide basic heuristics in order to explore the solution space more efficiently. Very well-known metaheuristics such as simulated annealing, tabu search, GRASP, and ILS consist of SLS methods and have been successfully used for many years in the context of combinatorial optimization problems.

These metaheuristics are iterative processes, where a feasible candidate solution is obtained in each iteration. Some of these metaheuristics, like GRASP and ILS, are memoryless, i.e., to obtain a solution in one iteration, no information about the candidate solutions obtained in previous iterations is used.

Otherwise, tabu search, scatter search, path relinking, and vocabulary building are memory-based heuristics that incorporate special forms of memory by generating new solutions through the combination of high-quality solutions previously found and stored in an elite set. These strategies were used to solve successfully several combinatorial optimization problems [4, 15, 16].

Data mining refers to the automatic extraction of knowledge from data [21, 52, 54, 56]. The extracted knowledge, expressed in terms of patterns, represents important features of the data at hand. Hence, data mining provides a means to better understand concepts implicit in raw data, which is fundamental in a decision-making process.

There are some works in the literature that incorporates data mining process in metaheuristics [31, 48]. The contribution of this chapter is to survey a set of hybrid metaheuristics that we have previously developed, which incorporate a data mining procedure to extract patterns from elite set solutions. The basic idea of using data mining is to obtain knowledge from the previous iterations to guide the search in next iterations. Patterns extracted from good quality solutions could be used to guide the search, leading to a more effective exploration of the solution space.

Ideas to extract good features from previously found solutions to guide future search of the solution space have been successfully exploited by stochastic local search heuristics. Lin and Kernighan [35] developed a multistart heuristic for the traveling salesman problem, where they fix some links that occur in good quality routes previously found by the algorithm. Fleurent and Glover [13] describe multistart strategies to the problem of quadratic assignment, wherein, during the construction process, the elements to be inserted into a solution are selected from an elite set that has the best solutions generated so far. Heuristics based on evolutionary algorithms [11] also try to maintain good characteristics of solutions found earlier by selecting some of the parents with good fitness function. Lodi et al. [36] developed an evolutionary heuristic for quadratic programming 0–1, where they present an intensification strategy, used in a genetic algorithm, to set variables, that can have their values fixed during all stages of the algorithm or only for a certain number of steps.

The aim of the hybrid data mining proposal is to use more elaborated techniques found in the data mining research area to search for good patterns extracted from a set of high-quality solutions.

This survey shows that memoryless heuristics may benefit from the use of data mining by obtaining better solutions in smaller computational times. Indeed, when a data mining process is used with heuristics without memory, these heuristics incorporate memory and better results are expected. Also, some revisited results show that even heuristics using memory mechanisms can benefit from data mining procedures by reducing the computational time to achieve good quality solutions.

The goal of this chapter is to review some applications of data mining within both memoryless and memory-based metaheuristics and survey the results obtained by us until now. In section “Data Mining”, an overview of basic data mining theory is outlined, and in section. “Strategies to Use Data Mining in Heuristics”, the strategy adopted to combine metaheuristics with a data mining process is described. In sections. “Memoryless Heuristics” and “Memory-Based Heuristics”, several implementations for different applications that use memoryless heuristics and that

use memory-based heuristics, respectively, along with corresponding computation results, are reviewed and analyzed. Some concluding remarks and a discussion about future work are presented in section "Conclusion".

Data Mining

Data mining processes aim at discovering novel, interesting, and potentially useful patterns and models, also referred as knowledge, from data. This knowledge discovery has been applied in a large variety of fields, such as marketing, finance, healthcare, education, and security. Independent of the application domain, data mining techniques can be employed in two distinct scenarios: predictive and descriptive. Predictive models are built in order to allow inferences and predictions on the data domain. Descriptive patterns are mined to enable a deeper understanding of the data at hand. Classification and regression models are examples of predictive knowledge. Frequent itemsets, association rules, sequential patterns, and data clusters are kinds of descriptive patterns.

This chapter describes and explores previously proposed hybrid metaheuristics that incorporate a data mining process. The basic idea of this hybridization is that patterns mined from a set of good quality solutions of a combinatorial problem represent characteristics of these elite solutions. Then these extracted patterns can be used to guide the search for better solutions, leading to a more effective exploration of the search space. This is a kind of descriptive mining application, and, in all hybrid metaheuristics explored here, the type of pattern used to characterize the good features of the elite solutions was frequent itemset, which is defined as follows.

All data mining concepts used and reviewed in this section can be found in any fundamental book of this field [21, 52, 54, 56]. Let $\mathscr{I} = \{i_1, i_2, \ldots, i_n\}$ be a set of items from an application domain. Let $\mathscr{D}$ be a set of transactions (a transactional database) defined over $\mathscr{I}$, where each transaction t is a subset of $\mathscr{I}$ ($t \subseteq \mathscr{I}$). Then, a nonempty subset X of $\mathscr{I}$ ($X \subseteq \mathscr{I}$), called itemset, holds in database $\mathscr{D}$ with support s % if s % of the transactions in $\mathscr{D}$ contain X. The support of X is also referred as $sup(X)$. Given a user-specified minimum support $minsup$, X is called a frequent itemset if $sup(X) \geq minsup$.

Market basket analysis is a typical application of this kind of pattern: a frequent itemset represents a set of products that appears together in a significant number of customers' purchases. Frequent itemsets can reveal interesting correlations among items within data.

As a general illustration, consider the following example. Let $\mathscr{I} = \{A, B, C, D, E, F, G, H\}$ be a set of products. Let $\mathscr{D}$ be the database composed of the following five purchase transactions: $\{A, C, D, E\}$, $\{A, C, E, F\}$, $\{B, E, F, G, H\}$, $\{A, B, C, E, G, H\}$, $\{A, E, H\}$. Suppose a minimum support $minsup$ of 0.6, i.e., the user wants to identify every set of products that occur together in at least 60% of the database transactions. Considering these settings, the frequent itemsets are $\{A\}$, $\{C\}$, $\{E\}$, $\{H\}$, $\{A, C\}$, $\{A, E\}$, $\{C, E\}$, $\{E, H\}$, and $\{A, C, E\}$ since all of them have support greater or equal to 60%. For example,

$sup(\{E\}) = 100\%$, $sup(\{A, E\}) = 80\%$, and $sup(\{A, C, E\}) = 60\,\%$. The itemset $\{B, E, G\}$ is not frequent since it appears in only 40% of the transactions, which is less than the user minimum threshold. If $minsup$ were 80%, there would be only three frequent itemsets: $\{A\}, \{E\}$, and $\{A, E\}$.

Since every subset of a frequent itemset is also frequent, in order to avoid presenting the frequent itemsets and their frequent subsets, one could prefer to mine only the maximal frequent itemsets. A frequent itemset X is maximal if none of its supersets is frequent. In the example, considering $minsup = 60\%$, there are two maximal frequent itemsets: $\{E, H\}, \{A, C, E\}$. With $minsup = 80\%$, there is only one: $\{A, E\}$.

In order to illustrate how these concepts could be employed on combinatorial optimization data, suppose that each transaction in the database $\mathscr{D}$ represents a good solution of an optimization problem. For example, consider a packing problem, where one has to pack a maximum number of elements (or a maximum cost associated to the elements), respecting the limit of the pack and restrictions regarding elements that cannot appear together in the pack. The first transaction indicates that packing the elements A, C, D, and E leads to a good solution.

In this scenario, a frequent itemset mined from an elite set of solutions represents characteristics of these suboptimal solutions. Considering $minsup = 60\%$, the frequent itemset $\{A, E\}$ indicates that the elements A and E are included in most elite solutions (80%) and could be good candidates to be inserted when constructing new solutions. The frequent itemset $\{A, C, E\}$ is not so strong, since its support is 60%; however, it has more elements and could lead to a faster construction of new good solutions.

When incorporating this kind of data mining task into a metaheuristic, it is important to guarantee the efficient extraction of the frequent itemsets – patterns – in order to avoid overloading and making the metaheuristic slower. Next, the employed data mining task will be defined, and the main proposed algorithms to solve it will be indicated.

Given a transactional database $\mathscr{D}$ defined over $\mathscr{I} = \{i_1, i_2, \ldots, i_n\}$ and a minimum support s, the frequent itemset mining (FIM) problem consists in identifying every frequent itemset that holds in $\mathscr{D}$, i.e., each set of items that occur together in at least s % of the database transactions. In 1993, the FIM problem was first introduced in [1], where the AIS algorithm was proposed to find all frequent itemsets in the search space of 2^n subsets of $\mathscr{I}$.

In 1994, the well-known iterative Apriori algorithm was proposed in [2], achieving significant improvements on the earlier strategy. Over the following ten years, many Apriori-like algorithms had been proposed to efficiently mine frequent itemsets [17]. Another competitive and important FIM strategy, which does not follow the Apriori approach, is the FP-growth algorithm [22]. The FPmax* strategy [19] implements the main ideas of the FP-growth to mine maximal frequent itemsets.

For most combinatorial problems explored in this chapter, the solution can be represented as a set of elements. Therefore, frequent itemsets, as well as maximal frequent itemsets, were suitably adopted to represent patterns hidden in the elite

set of solutions. The data mining modules inserted into the hybrid metaheuristics that will be shown in the next sections were built based on the FPmax* algorithm (available at http://fimi.cs.helsinki.fi).

Strategies to Use Data Mining in Heuristics

This section describes strategies used to incorporate a data mining process in stochastic local search metaheuristics.

As described in [28, 29], given a (combinatorial) problem Π, a stochastic local search algorithm for solving an arbitrary problem instance $\pi \in \Pi$ is defined by the following components:

- The search space $S(\pi)$ of instance π, which is a set of candidate solutions $s \in S$;
- A set of (feasible) solutions $S'(\pi) \subseteq S(\pi)$;
- A neighborhood relation on $S(\pi)$, $N(\pi) \subseteq S(\pi) \times S(\pi)$;
- An initialization procedure $init_proc(\pi)$ that draws an element from a probabilistic function $init(\pi)$, which specifies a probability distribution over initial search positions;
- A step procedure $step_proc(\pi, s)$ that draws an element from a probabilistic function $step(\pi)(s)$, which maps each given position s onto a probability distribution over its neighboring positions, for specifying the local search steps;
- A termination procedure $term_proc(\pi, s)$ which uses a terminate predicate T which indicates the probability with which the search is to be terminated upon reaching a specific point in the search space.

In Fig. 1, a pseudo-code for a stochastic local search for a minimization problem is shown. In line 1, an initial solution is obtained using the probabilistic $init$ function, and from lines 3 to 10, an iterative process is performed that moves the incumbent solution s through the search space trying to improve the objective function $f()$ until a termination predicate is achieved.

Fig. 1 Pseudo-code of the SLS for a minimization problem

```
procedure SLS_Minimization(π' ∈ Π', f(π'))
 1.   s ← init_proc(π');
 2.   best_sol ← s;
 3.   while not term_proc(π', s) do
 4.     s ← step_proc(π', s);
 7.     if f(π', s) < f(π', best_sol)
 8.       best_sol ← s;
 9.     end if
10.   end while
11.   if best_sol ∈ S'(π')
12.     return best_sol;
13.   else
14.     return ∅;
15.   end if
```

```
procedure MS_Heur_Minimization(n_iter,π' ∈ Π', f(π'))
 1.   f(π', best_sol) ← ∞;
 2.   for it ← 1 to n_iter do
 3.     s ← ∅
 4.     s ← construct_sol(s, π', f(π'));
 5.     s ← local_search(s, π', f(π'));
 6.     if f(π', s) < f(π', best_sol)
 7.       best_sol ← s;
 8.     end if
 9.   end for
10.   if best_sol ∈ S'(π')
11.     return best_sol;
12.   else
13.       return ∅;
14.   end if
```

Fig. 2 Pseudo-code of the multistart SLS heuristic for a minimization problem

Usually in constructive heuristics, the initial solution s obtained in line 1 is an empty solution, and the $step_proc$ procedure consists of using a function $step$ that generates neighbors by extending the initial solution and evaluating these neighbors so that one of them is chosen to be part of the solution. The $term_proc$ procedure returns $TRUE$ when a feasible solution is obtained.

In local search heuristics, a complete initial solution s is obtained in line 1, and the $step_proc$ procedure consists of using a function $step$ that generates neighbors by changing some attributes of the initial solution and evaluating these neighbors, so that one of them is chosen to be the next incumbent solution. The $term_proc$ procedure may return $TRUE$ when no better solution than the incumbent solution is found.

Many times, stochastic local search metaheuristics are multistart methods where several iterations are performed, and, in each iteration, a solution is built using a constructive heuristic, and then a local search is executed until a termination criterion is achieved. Figure 2 shows a pseudo-code for this kind of multistart heuristics. In line 4, a solution is built, and, in line 5, a better solution is sought in the neighborhood. The best solution is updated in line 7.

When using stochastic local search, the solutions obtained may be different in each run of the algorithm, so multistart SLS metaheuristics are able to find different solutions in each iteration.

So the basic concept of incorporating a data mining process in multistart SLS metaheuristics is that patterns found in high-quality solutions obtained in earlier iterations can be used to conduct and improve the search process.

Basically two strategies developed by us to use data mining are presented. In the first one, called DM-MSH, the multistart heuristic (MSH) is divided in two phases. The first one is called *elite set generation phase*. It consists in executing some gen_elite_iter MSH iterations to obtain a set $\mathscr{S}$ of different solutions. The d best solutions from $\mathscr{S}$ compose an elite set $\mathscr{D}$.

Between the two phases, the data mining procedure is applied. It is responsible for extracting a set of patterns $\mathscr{P}$ from the elite set $\mathscr{D}$. The patterns to be mined

are sets of elements that frequently appear in solutions in $\mathscr{D}$, which characterizes a frequent itemset mining application. A frequent itemset mined from $\mathscr{D}$ with support s % represents a set of elements that occur in s % of the elite solutions.

Next, the second phase, called *hybrid phase*, is performed. In this phase, a number of slightly different MSH iterations are executed. In these iterations, the construction and/or local search heuristics may use a mined pattern $p \in \mathscr{P}$. For example, in constructive heuristics, all elements of a pattern p may be inserted into the partial solution, from which a complete solution will be built executing the standard construction procedure. This way, all constructed solutions will contain the elements of a pattern $p \in \mathscr{P}$. Likewise, local search procedures may use the elements of pattern p by requiring that only neighbors that contain elements from pattern p should be evaluated.

The pseudo-code of DM-MSH is illustrated in Fig. 3. In line 1, the elite set $\mathscr{D}$ is initialized with an empty set. The loop from line 3 to line 11 corresponds to the *elite set generation phase*, in which gen_elite_iter iterations of the multistart heuristic are performed. The original construction method is executed in line 5, having as input an empty solution that will be constructed. Then a local search method is performed in line 6. The elite set $\mathscr{D}$ is updated in line 7. In line 8, it is checked whether the best solution should be updated, which is done in line 9. The data mining procedure extracts the set of patterns $\mathscr{P}$ from $\mathscr{D}$ in line 12. The loop

Fig. 3 Pseudo-code of DM-MSH

```
procedure DM_MSH(n_iter, π' ∈ Π', f(π'))
 1.  D ← ∅;
 2.  f(π', best_sol) ← ∞;
 3.  for it ← 1 to gen_elite_iter do
 4.    s ← ∅
 5.    s ← construct_sol(s, π', f(π'));
 6.    s ← local_search(s, π', f(π'));
 7.    UpdateElite(D, s);
 8.    if f(π', s) < f(π', best_sol)
 9.      best_sol ← s;
10.    end if
11.  end for
12.  P ← Mine(D);
13.  p ← SelectPattern(P);
14.  for it ← gen_elite_iter to n_iter do
15.    s ← ∅;
16.    s ← dm_construct_sol(p, s, π', f(π'));
17.    s ← dm_local_search(p, s, π', f(π'));
18.    if f(π', s) < f(π', best_sol)
19.      best_sol ← s;
20.    end if
21.    p ← SelectPattern(P);
22.  end for;
23.  if best_sol ∈ S'(π')
24.    return best_sol;
25.  else
26.    return ∅;
27.  end if
```

from line 14 to line 22 corresponds to the *hybrid phase*. Each iteration is based on a pattern $p \in \mathscr{P}$. This pattern p is initialized in line 13. The construction procedure that may use data mining is performed in line 16. In line 17, the local search that may use data mining is executed. From lines 18 through 20 the best solution is updated. In line 21, another pattern is chosen to be used in the next iteration. The best solution is returned in line 24.

In DM-MSH, the data mining procedure is executed just once. Although satisfactory results were obtained with this strategy, a second strategy was developed called MDM-MSH to attempt to further improve the results. The main ideas of this strategy are to mine more than once and as soon as the elite set is stable and good enough to be mined. The mining process will be executed (a) as soon as the elite set becomes stable which means that no change in the elite set occurs throughout a given number of iterations and (b) whenever the elite set has been changed and again has become stable.

Figure 4 shows the pseudo-code of MDM-MSH. The loop from line 4 to 13 corresponds to the first elite set generation phase, in which MSH iterations are

Fig. 4 Pseudo-code of MDM-MSH

```
procedure MDM_MSH(n_iter, π' ∈ Π', f(π'))
 1.   𝒟 ← ∅;
 2.   f(π', best_sol) ← ∞;
 3.   it ← 0;
 4.   repeat
 5.     s ← ∅
 6.     s ← construct_sol(s, π', f(π'));
 7.     s ← local_search(s, π', f(π'));
 8.     UpdateElite(𝒟, s);
 9.     if f(π', s) < f(π', best_sol)
10.       best_sol ← s;
11.     end if
12.     it ← it + 1;
13.   until elite_set_is_ready or it = n_iter
14.   while it < n_iter do
15.     if elite_set_is_ready
16.       𝒫 ← Mine(𝒟);
17.     end if
18.     p ← SelectPattern(𝒫);
19.     s ← ∅;
20.     s ← dm_construct_sol(p, s, π', f(π'));
21.     s ← dm_local_search(p, s, π', f(π'));
22.     if f(π', s) < f(π', best_sol)
23.       best_sol ← s;
24.     end if
25.     UpdateElite(𝒟, s);
26.     it ← it + 1;
27.   end while;
28.   if best_sol ∈ S'(π')
29.     return best_sol;
30.   else
31.     return ∅;
32.   end if
```

performed until the elite set becomes ready to be mined or the total number of iterations is executed. Next, in the loop from line 14 to 27, whenever the elite set is ready, the data mining procedure is executed in line 16. In line 18, a pattern is selected. Then the adapted construction is performed in line 20, and, in line 21, the adapted local search is executed. If a better solution is found, the best solution is updated in line 23. After the execution of all iterations, the best solution is returned in line 29.

One important characteristic of SLS metaheuristics is the level of intelligence used to exploit memory during its execution. Some metaheuristics use explicit designs for recording past elements and use this information in a strategic way. Based on the explicit use of memory, metaheuristics can be classified as memoryless or memory-based methods. Metaheuristics such as greedy randomized adaptive search procedure (GRASP) and iterated local search (ILS) do not incorporate the explicit use of memory structures and are classified as memoryless heuristic. Tabu search (TS) and scatter search (SS) are memory-oriented methods in which past choices are used and are classified as memory-based heuristics [8].

The next two sections show that inserting data mining to memoryless methods was able to improve the quality and computational time of the original heuristics and that inserting data mining into memory-based heuristics improved computational time of the original heuristic preserving its quality.

Memoryless Heuristics

This section presents how data mining was used with some memoryless heuristics. First, the integration of data mining with a greedy randomized adaptive search procedure (GRASP) developed to solve the server replication for reliable multicast problem is described. The solutions of this problem are unordered set of elements.

Then another GRASP created to solve the traveling salesman problem with pickup and delivery is presented. This problem has solutions that are ordered set of elements, which required an approach that was different from the one employed in the first problem. In both problems, patterns were used in the construction phase.

Initially, some iterations of the original GRASP are performed, and an elite set with some good quality solutions is obtained. Then a data mining procedure extracts some patterns from the elite set, and some elements from these patterns are fixed as initial elements of a solution in the construction phase. The computational results showed that using data mining improved the results related to both the quality and computational time.

Finally, the results obtained by using data mining with an iterated local search (ILS) heuristic for the set covering with pairs problem are showed. In this problem, also some original ILS iterations were performed, and an elite set with some good solutions was obtained. The data mining process extracted some patterns from the elite set whose elements were used in the next ILS iterations within just the local search procedure. The results showed that in this case the quality of the solution remained the same, but the processing time was reduced.

Hybrid DM-GRASP

GRASP [12] was the first SLS metaheuristic to be hybridized with data mining. It is a multistart heuristic which performs several iterations and each iteration consists of two phases: construction and local search. A feasible solution is obtained in the construction phase, which starts from an empty solution and builds a solution by performing iterations to insert new elements into the solution. In each iteration, an element is randomly selected from a restricted candidate list (RCL) which is obtained from a candidate list (CL). This CL is a list with all candidates ordered by a greedy evaluation function, and the RCL is obtained by picking the top elements from CL. A real value parameter $\alpha \in [0, 1]$ determines the size of RCL: if $\alpha = 0$ the RCL contains just the first element from CL and if $\alpha = 1$ the RCL contains all elements from CL.

After the construction phase, the neighborhood of the constructed solution is explored by a local search in order to find a better solution. The result is the best solution found over all iterations.

The hybridization of GRASP with a data mining process was first introduced and applied to the set packing problem [47] and to the maximum diversity problem [49], and the results were equally successful.

In both applications, the data mining technique was applied in the construction phase. The implementations used for these problems were divided in two parts. In the first one, a number of GRASP iterations were executed, and the best solutions were stored in an elite set. Then, a data mining algorithm was used to extract patterns from this set of suboptimal solutions. In the second part, the GRASP iterations used the mined patterns just to construct new solutions. In this strategy, the data mining process was performed only once, after exactly half of the GRASP iterations.

New versions of the DM-GRASP metaheuristic were explored to experiment a single activation in different points of the GRASP execution and also multiple and adaptive executions of the data mining process during the GRASP execution.

These proposals were evaluated using the server replication for reliable multicast problem, which consists in selecting, from a set of nodes in a computer network, those which should act as replicated servers to provide a reliable and efficient multicast service, which minimizes the transmission cost.

The pseudo-code of the DM-GRASP is illustrated in Fig. 5. In lines 1 and 2, the cost of the best solution and the elite set are initialized. The loop from line 3 to line 10 corresponds to the elite set generation phase, in which original GRASP is performed for n iterations. The original construction method is executed in line 4, followed by the local search method in line 5. The elite set, composed of d solutions, is updated in line 6. In line 7, it is checked whether the best solution should be updated, which is done in line 8. In line 11, the data mining procedure extracts p patterns from the elite set, which are inserted in decreasing order of pattern size in the set of patterns $patterns_set$. The loop from line 12 to line 19 corresponds to the hybrid phase. Each iteration is based on a pattern selected in line 13 from the set of patterns in round-robin way. The adapted construction procedure is performed in line 14, using the selected pattern as a starting point. In line 15, the same local search

```
procedure DM_GRASP()
1.  Cost(best_sol) ← ∞;
2.  elite_set ← ∅;
3.  for it ← 1 to n do
4.     sol ← Construction();
5.     sol ← Local_Search(sol);
6.     UpdateElite(elite_set, sol, d);
7.     if Cost(sol) < Cost(best_sol)
8.        best_sol ← sol;
9.     end if
10. end for
11. patterns_set ← Mine(elite_set);
12. for it ← 1 to n do
13.    pattern ← SelectNextLargestPattern(patterns_set);
14.    sol ← Adapted_Construction(pattern);
15.    sol ← Local_Search(sol);
16.    if Cost(sol) < Cost(best_sol)
17.       best_sol ← sol;
18.    end if
19. end for;
20. return best_sol;
```

Fig. 5 Pseudo-code of the DM-GRASP

performed in line 5 is executed. From line 16 to 18 the best solution is updated. The best solution is returned in line 20.

Multicast communication is the delivery of information to many receivers simultaneously. The server replication approach was used for this problem, in which data are replicated at some of the multicast-capable relaying hosts and each of them is responsible by the retransmission of packets to receivers of its group.

The developed DM-GRASP was an extension of the GRASP implementation proposed in [34]. The solution of the problem is a set of M multicast-capable relaying hosts that will be the replicated servers. The optimal solution is the one that minimizes the cost of sending the packets in the network.

In the construction phase, relaying hosts are added to the solution one by one. In the local search phase, the neighborhood structure considered is based on the 1-exchange procedure. Every solution that can be obtained by replacing one element of a solution s is part of its neighborhood. The best solution of the neighborhood becomes the starting point of the next local search iteration.

In the context of the problem addressed, a pattern of suboptimal solutions is a set of relaying hosts that frequently appears in solutions of the elite set. The algorithm FPmax* [19] was used to extract the maximal patterns from the elite set.

Computational Results

In order to evaluate the proposed strategies, five simulated multicast scenarios were generated in [39]. Three of them represent virtual conferences and two represent broadcast transmissions.

Both GRASP and DM-GRASP were run ten times with a different random seed in each run. The GRASP parameter α was set to 0.7, as used in [34]. Each run consisted of 500 iterations. In the DM-GRASP strategy, the data mining process is performed only once, after exactly half the iterations. The elite set generation phase took 250 iterations and the hybrid phase took the rest. The size of the elite set, from which the patterns are mined, was set to 10. A set of nodes was considered a pattern if it was present in at least two of the elite solutions. The ten largest patterns found were inserted in the pattern set.

The results related to the quality of the obtained solutions were calculated based on the deviation value computed as follows:

$$dev = \frac{(HeuristicCost - BestKnownCost)}{BestKnownCost} \times 100, \tag{1}$$

where *HeuristicCost* is the (best or average) cost obtained by the heuristic technique and the *BestKnownCost* is the best known value for the working instance.

For the 20 instances, DM-GRASP found 12 better results for best deviation, and GRASP and DM-GRASP found 8 equal results. DM-GRASP found 12 better results for average deviation, GRASP found 3, and GRASP and DM-GRASP found the same result for 5 instances. The average value of the deviation value of the best cost obtained by GRASP was 0.31 and by DM-GRASP was 0.08. And the average value of the deviation value of the average cost obtained by GRASP was 0.42 and by DM-GRASP was 0.21.

Figures 6 and 7 present the behavior of the construction and local search phases, in terms of the cost values obtained, by GRASP and DM-GRASP throughout the execution of 500 iterations, for a specific instance ($\text{BROAD}_1, m = 75, seed = 10$).

Since the server replication is a minimization problem, the figures show that the local search always reduce the cost of the solution obtained by the construction phase. In Fig. 6, GRASP presents the same behavior for both construction and local search during all iterations. In Fig. 7, after iteration 250, when the data mining procedure is executed in the DM-GRASP strategy, there is an improvement in the quality of the solutions reached by the construction and local search phases.

Table 1 presents the percentage difference from the DM-GRASP average computational time to the GRASP average computational time for each group of instances. The execution times for DM-GRASP are considerably smaller than those for GRASP.

Figures 8 and 9 show the behavior of the construction and local search phases, in terms of the time used in each phase, by GRASP and DM-GRASP throughout the execution of 500 iterations, for a specific instance ($\text{BROAD}_1, m = 75, seed = 10$).

Figure 8 shows that when executing the GRASP heuristic, the behavior of both construction and local search is the same during all iterations. Figure 9 presents the behavior when using the DM-GRASP strategy. After iteration 250, the time spent in both construction and local search phases reduces considerably.

There are two main reasons for the faster behavior of DM-GRASP. First, the computational effort of the adapted construction phase is smaller than the traditional

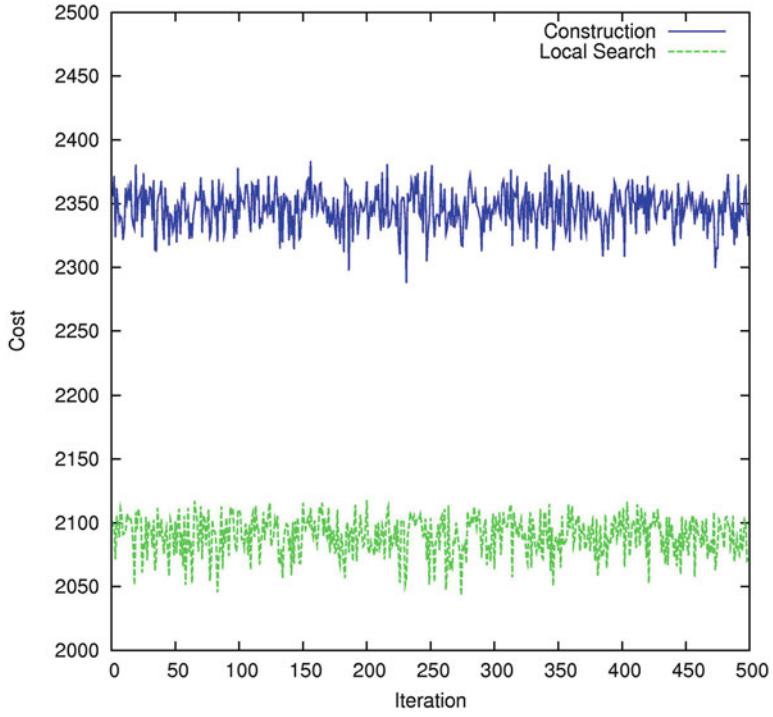


Fig. 6 One execution of GRASP

construction, since the elements from a pattern are initially fixed in the solution. Then a smaller number of elements must be processed and inserted into the constructed solution. Second, the use of patterns leads to the construction of better solutions which are the input for the local search. This incurs in less computational effort taken to converge to a local optimal solution.

Next, the DM-80 strategy was developed, which executed the data mining process closer to the last iterations, reserving 80% of the iterations to the elite set generation. The objective of changing the point where the data mining occurs is to evaluate if it is better to mine only near the end (DM-80), when the elite set elements would have probably achieved higher quality.

Then, the DM-3X strategy was developed to execute the data mining algorithm three times: after the execution of 20%, 50%, and 80% of the total number of iterations. The elite set generation phase is performed until 20% of the total number of iterations is executed. After that, the hybrid phase starts. However, in this strategy, the elite set continues to be updated, and the data mining process is performed two more times.

The results obtained for both versions (DM-80 and DM-3X) in [39] were not so different concerning quality solution, so it was not possible to conclude how to define the best fixed moment to activate the data mining process. So, an investigation

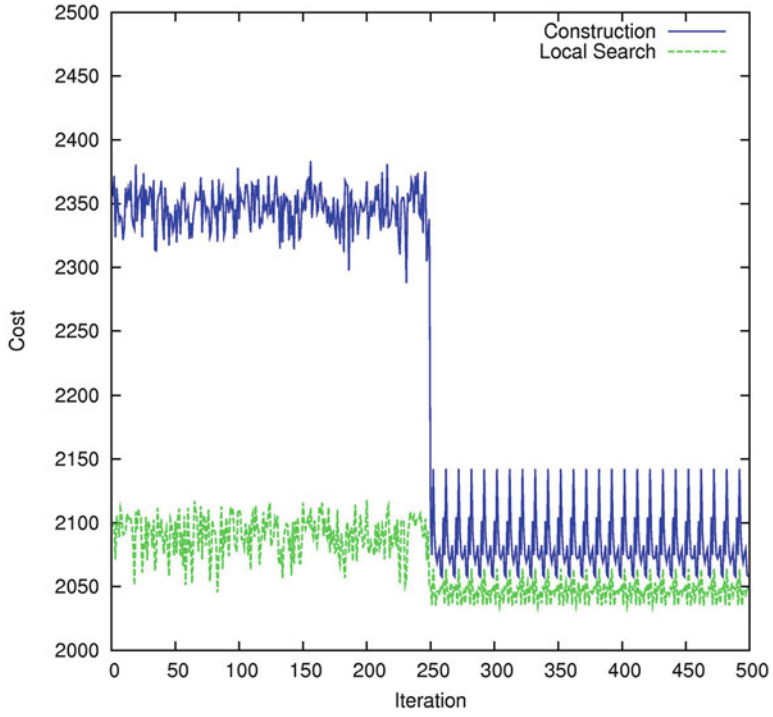


Fig. 7 One execution of DM-GRASP

was carried out about using a dynamic way to set the moment when the data mining process should be performed.

The main idea of this proposal, called Multi DM-GRASP (MDM-GRASP), was to execute the mining process: (a) as soon as the elite set becomes stable and (b) whenever the elite set has been changed and again has become stable. This idea originated the MDM-MSH which was already described in section "Strategies to Use Data Mining in Heuristics".

The pseudo-code of the Multi DM-GRASP is illustrated in Fig. 10. In lines 1 and 2, the best solution and the elite set are initialized. The loop from line 3 to 10 corresponds to the first elite set generation phase, in which original GRASP iterations are performed until the elite set becomes ready to be mined or the termination criterion – the total number of iterations – becomes true. To determine that the elite set is ready to be mined, the procedure checks if the set has not being changed for a specific percentage of the total number of iterations after its last update. Next, in the loop from line 11 to 22, whenever the elite set is ready, the data mining procedure is executed, and a new pattern set is created in line 13, and, from line 15 to 21, a hybrid iteration is executed. In line 15, a pattern is selected from the patterns set. Then the adapted construction is performed in line 16, using

Table 1 Comparing computational time of GRASP and DM-GRASP

Instance	% time DM-GRASP
$BROAD_1$	43.75
$BROAD_2$	46.75
$CONF_1$	29.00
$CONF_2$	30.50
$CONF_3$	37.75

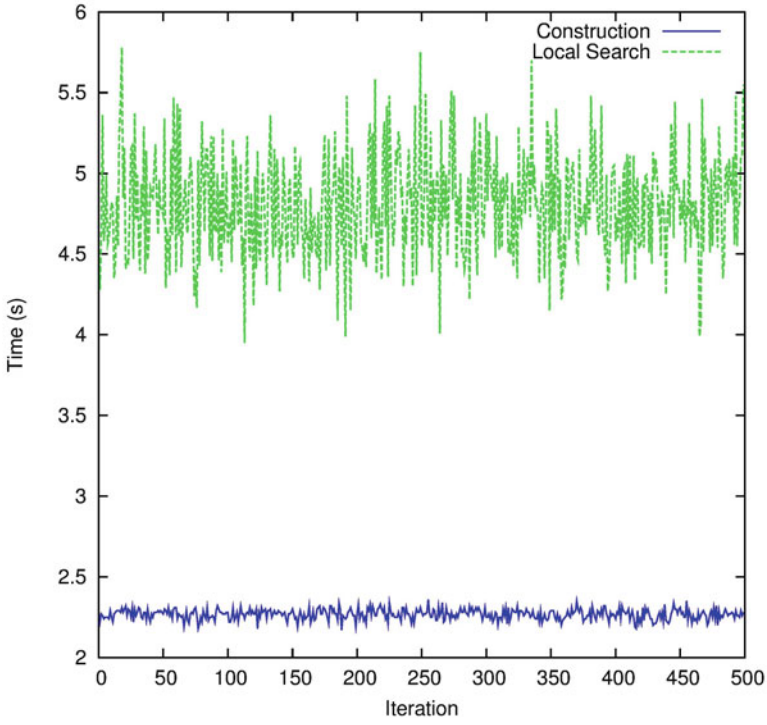


Fig. 8 Computational time spent in each phase of GRASP

the selected pattern as a starting point. In line 17, the local search is executed. From line 19 to 21, the best solution is updated. The best solution is returned in line 23.

The dynamic strategy awaits a given amount of iterations during which there are no changes in the elite set and only then performs the data mining process. Experiments were performed using three different values for the number of iterations to wait: 5%, 10%, and 20% of the total number of iterations, which are called MDM-D5, MDM-D10, and MDM-D20, respectively.

The results obtained in [39] showed that the MDM-D5 strategy presented the best behavior related to solutiont's quality and computational time.

Table 2 shows the percentage difference from DM-80 and MDM-D5 average computational times to GRASP average computational time, for each group of

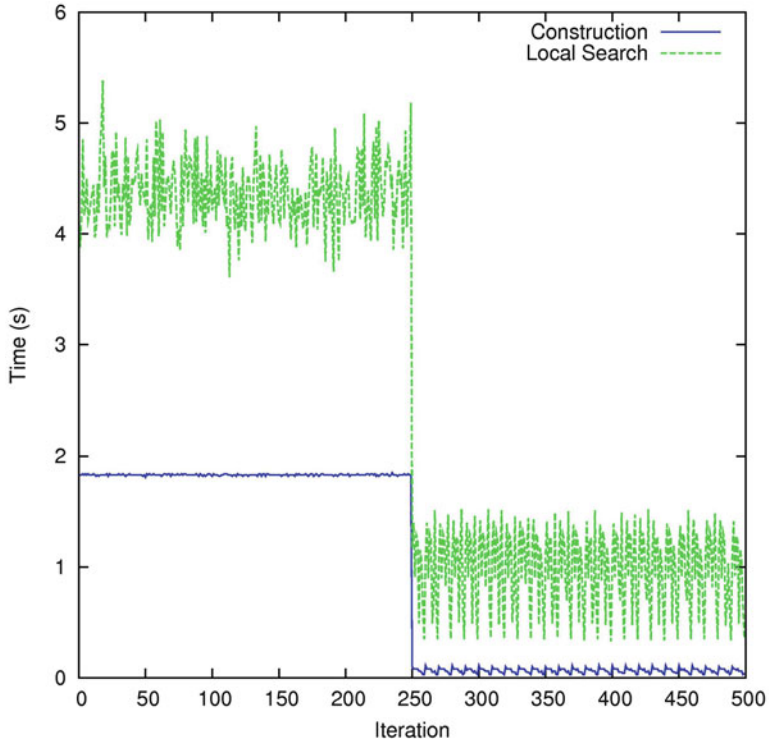


Fig. 9 Computational time spent in each phase of DM-GRASP

instances. It can be seen that the MDM-D5 was faster than GRASP and DM-80 for all instances.

Another experiment was performed to evaluate the time required for GRASP, DM-80, and MDM-D5 to achieve a target solution value.

Figures 11 and 12 show *time-to-target* plots (TTT) [3] used to analyze the behavior of randomized algorithms. A TTT plot is generated by executing an algorithm several times and measuring the time required to reach a solution at least as good as a target solution.

GRASP, DM-80, and MDM-D5 were executed 100 times each (with different random seeds), until a target solution was reached for a specific instance. The instance BROAD_1, $m = 25$, was used for test cases, and two targets were analyzed. An easy target (value 2820) was selected, which was a value achieved by the three strategies in the majority of the previous performed executions, and a more difficult one (value 2818.92) was used, which was the best value obtained by the GRASP for the instance.

The i-th sorted running time t_i is associated with a probability $p_i = (i - 1/2)/100$, and the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 100$ are plotted. Each plotted point will then indicate the probability (vertical axis) for the strategy to achieve the target solution in the indicated time (horizontal axis).

```
procedure MDM_GRASP()
1.  Cost(best_sol) ← ∞;
2.  elite_set ← ∅;
3.  repeat
4.     sol ← Construction();
5.     sol ← Local_Search(sol);
6.     UpdateElite(elite_set, sol, d);
7.     if Cost(sol) < Cost(best_sol)
8.        best_sol ← sol;
9.     end if
10. until elite_set_is_ready or end_criterion;
11. while not end_criterion;
12.    if elite_set_is_ready
13.       patterns_set ← Mine(elite_set);
14.    end if
15.    pattern ← SelectNextLargestPattern(patterns_set);
16.    sol ← Adapted_Construction(pattern);
17.    sol ← Local_Search(sol);
18.    UpdateElite(elite_set, sol, d);
19.    if Cost(sol) < Cost(best_sol)
20.       best_sol ← sol;
21.    end if
22. end while;
23. return best_sol;
```

Fig. 10 Pseudo-code of the Multi DM-GRASP

Table 2 Comparing DM-80 and MDM-D5 computational time related to GRASP computational time

Instance	% time DM-80	% time MDM-D5
$BROAD_1$	23.75	53.75
$BROAD_2$	23.25	55.75
$CONF_1$	12.25	48.25
$CONF_2$	10.25	49.25
$CONF_3$	17.75	55.25

For the easy target, the behaviors of the three strategies were similar. For difficult targets, the probability for MDM-D5 to reach the target in less time is always bigger than the probability for DM-80 and GRASP. The plots in Fig. 12 indicate that MDM-D5 is able to reach difficult solutions much faster than the other two strategies, demonstrating the robustness of this strategy.

In this evaluation, the superiority of the data mining strategies was confirmed, with a better behavior for the MDM-D5.

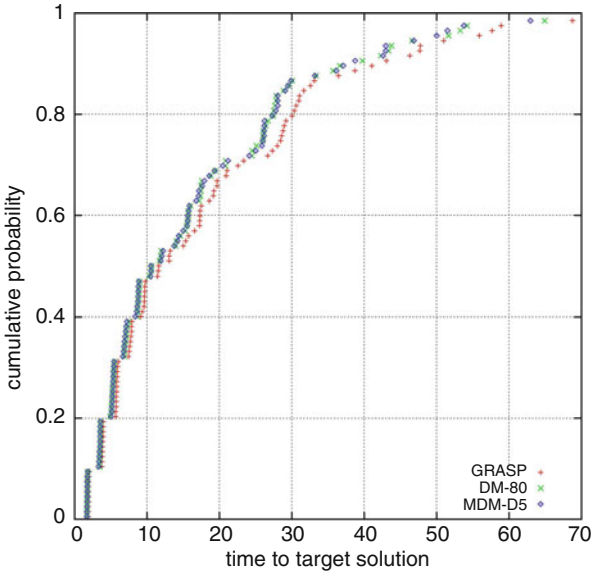


Fig. 11 Time-to-target plot for an easy target for instance BROAD_1, $m = 25$

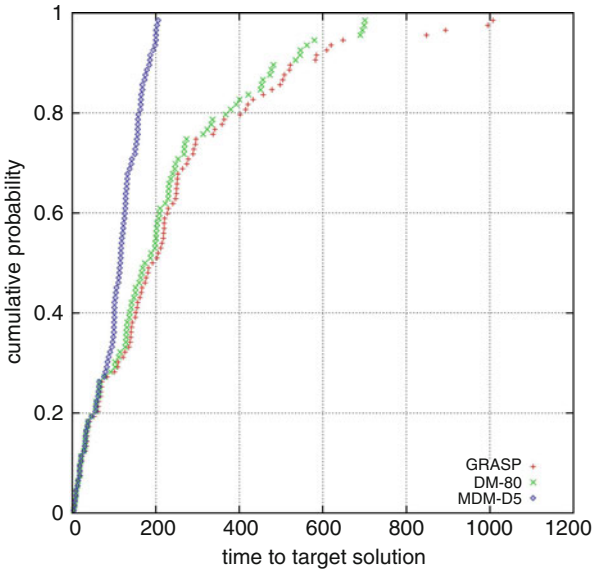


Fig. 12 Time-to-target plot for a difficult target for instance BROAD_1, $m = 25$

Hybrid DM-GRASP+VND

The hybrid DM-GRASP+VND heuristic, proposed in [20], combined a data mining technique with an existing heuristic for the one-commodity pickup and delivery traveling salesman problem (1-PDTSP). 1-PDTSP was first presented in [26] and is a generalization of the traveling salesman problem (TSP) because each customer has a certain demand by the distributed product. Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes representing customers and E is the set of edges and a vehicle with limited capacity Q. Given a nonnegative weight function $w : E \rightarrow \mathbb{R}_+$ associated with the edges, the 1-PDTSP consists in finding a path of shortest cost going through all customers, respecting the constraints of demand and of vehicle capacity. 1-PDTSP is NP-hard and has some practical applications in the repositioning scenario.

The hybrid GRASP+VND heuristic presented in [27] has the same structure of a classical GRASP metaheuristic. This strategy consists of a main loop where the termination criterion is the number of iterations. Each iteration has a construction phase and a local search procedure. After all iterations, a post-optimization phase is run, using another local search procedure, trying to improve the best overall solution found. Both local search approaches are based on the VND procedure.

In the construction phase, one customer is selected at random to be the depot. After that, customers are inserted iteratively at the end of the path as follows. In each iteration, customers that can be feasibly inserted into the solution under construction are sorted by their distance to the last customer in the solution, and only the first l elements will be part of the restricted candidate list (RCL). If there is no customer that can be feasibly added to the path, the RCL is built by the first l closest customers to the one at the end of the current solution. Finally, one customer of the RCL is chosen at random and inserted at the end of the path. The construction ends when all customers are in the solution.

The local search phase, named VND_1, is based on the variable neighborhood descent procedure [38], which consists in applying multiple neighborhood structures to a given solution in a predefined order, and whenever the current solution is improved, the procedure returns to the first neighborhood structure. The VND_1 is made by two classic moves, 2-opt and 3-opt, modified to accept infeasible solutions as a start point. These moves are applied in the following order: first, the 2-opt heuristic, which removes two nonadjacent edges and inserts them in another way to build a new route, and, next, the 3-opt, which is almost the same as the previous one, but handling three edges.

After the end of the main loop, the post-optimization phase is performed with another VND procedure, named VND_2, which is applied to the best solution found so far. The VND_2 consists of two other neighborhood structures based on the reinsertion move, also well-known for TSP. This move is divided into two smaller structures, applied in this order: first, the reinsertion forward, that removes a customer and reinserts it in a position after its original position, and secondly, reinsertion backward, similar to the first one but the removed customer is reinserted in a previous position.

When the solution of the problems is represented by a subset of elements, without setting any ordering, the application of the FIM technique is directly and successfully applied. However, in the case of the 1-PDTSP, the order of the elements is essential. A solution for the 1-PDTSP is defined by a sequence of visit to the elements, and, hence, an intermediate step must be performed in order to apply the FIM technique.

This transformation is explained as follows: for each pair of consecutive customers (i and j) from a given solution, an arc (i, j) is generated, mapping each solution to a set of arcs. For example, considering a solution $s = \{5, 2, 7, 4, 1, 6, 3, 5\}$ represented by this sequence of elements, s can be mapped as $s_{map} = \{a_{5\to2}, a_{2\to7}, a_{7\to4}, a_{4\to1}, a_{1\to6}, a_{6\to3}, a_{3\to5}\}$, where $a_{5\to2}$ is the arc from 5 to 2 and so on. Thereby, the sequence of s can be mapped to s_{map}, which is a set, without losing the order of elements. Once all the solutions were transformed into sets, it is possible to apply the FIM technique.

The algorithm in Fig. 13 illustrates the DM-GRASP+VND approach [20] for the 1-PDTSP. DM-GRASP+VND is divided into two main phases. The first one is called the elite set generation phase and consists of executing n original GRASP+VND iterations which generate a set of different solutions, storing the d best solutions in the elite set (lines 3–11). After this, the transformation step is called. Afterwards, the data mining technique is applied and the t largest patterns are chosen (line 12).

```
procedure DM_+GRASP+VND(n, d, t)
1.  f* ← ∞;
2.  M ← ∅;
3.  for k = 1, ..., n do
4.      x ← GreedyRandomizedConstruction1 - PDTSP;
5.      x ← VND1(s);
6.      UpdateElite(M, x, d);
7.      if f(x) < f* then
8.          x* ← x;
9.          f* ← f(x);
10.     end if;
11. end for;
12. patterns_set ← Mine(M, t);
13. for k = 1, ..., n do
14.     pattern ← SelectNextLargestPattern(patterns_set);
15.     x ← AdaptedGreedyRandomizedConstruction1-PDTSP(pattern);
16.     x ← VND1(x);
17.     if f(x) < f* then
18.         x* ← x;
19.         f* ← f(x);
20.     end if
21. end for;
22. x* ← VND2(x*);
23. return x*;
```

Fig. 13 Pseudo-code of the hybrid DM-GRASP+VND for the 1-PDTSP

A mined pattern is made of a set of arcs that frequently appeared in the elite set, i.e., with support greater or equal to the minimum support. Inside a pattern, an arc (i, j) has an origin costumer i and a destination costumer j. Moreover, one can find that, in the same pattern, two or more arcs may be consecutive and can be easily connected to set up a bigger route segment, named path segment (PS). This way, each pattern is made of one or more PS.

The second phase of the DM-GRASP+VND consists of executing other n iterations, replacing the original construction phase by an adapted construction which uses the patterns extracted in the first phase to build new solutions (lines 13–21). In line 22, the post-optimization is called. After the execution of all iterations, the best solution is returned in line 23.

Computational Results

The computational results obtained for GRASP+VND and DM-GRASP+VND strategies were compared in [20].

The set of instances used for the 1-PDTSP is provided by [27] which contains a few randomly generated instances from 100 to 500 customers, using a vehicle capacity of 10. These instances are the biggest in terms of number of customers and hardest in terms of vehicle capacity. The maximum number of iterations n, the elite set size d, the minimum support value, the number of patterns extracted t, and the number of reserved iterations to build the elite set are, respectively, 200, 10, 20%, 10, e 100. With the exception of the number of iterations, which was chosen according to the original parameter in [27], the others were defined based on the reports made in [40]. Each strategy was run ten times with a different random seed in each run.

In Table 3, the results for the set of difficult instances are shown. The first column shows the instance identifier. The second, third, and fourth columns have the best cost values, the average cost, and the average execution time (in seconds) obtained for GRASP+VND. The fifth, sixth, and seventh columns report the percentage difference (Diff %) of the hybrid DM-GRASP+VND over the GRASP+VND for each criteria. The percentage difference is defined as follows:

$$\text{Diff } \% = \frac{\text{Hybrid DM-Heuristic} - \text{GRASP+VND}}{\text{GRASP+VND}}$$

The intermediate rows show the partial averages of the percentage differences for each group of the same size instances, and the last row of the table presents the overall average. The smallest values considering the best solution, the average solution, and the average time, i.e., the best results among them, are boldfaced.

The results shown in Table 3 point out that the proposed DM-GRASP+VND method produced better solutions in less computational time for almost all instances. Only in 5 out of 50 instances, the DM-GRASP+VND did not outperform GRASP+VND in terms of best solution found, giving an overall percentage difference of 1.46% and being on average 30.63% faster than the original method. In

Table 3 Computational results for difficult instances

	GRASP+VND			DM-GRASP+VND		
Instances	Best solution	Average solution	Average time (s)	Diff % best	Diff % average	Diff % time
n100q10A	12369	12514.4	4.01	**−3.67**	**−1.11**	**−25.98**
n100q10B	13668	13885.7	3.86	**−0.53**	**−0.45**	**−28.07**
n100q10C	14619	14810.8	4.01	**−2.11**	**−1.40**	**−28.92**
n100q10D	14806	14993.4	4.15	**−0.95**	**−1.47**	**−24.76**
n100q10E	12594	12819.7	3.94	**−4.57**	**−1.81**	**−33.27**
n100q10F	12082	12297.2	3.57	**−1.58**	**−1.40**	**−25.24**
n100q10G	12344	12623.4	3.84	**−1.36**	**−1.12**	**−29.56**
n100q10H	13405	13590.7	3.72	**−0.32**	**−0.96**	**−27.93**
n100q10I	**14512**	14715.9	3.74	0.01	**−0.12**	**−30.58**
n100q10J	**13700**	13992.0	4.00	0.09	**−0.62**	**−25.28**
Group Average				**−1.50**	**−1.05**	**−27.96**
n200q10A	18707	19053.1	34.34	**−2.07**	**−1.72**	**−30.10**
n200q10B	19046	19406.7	33.27	**−1.87**	**−0.69**	**−34.18**
n200q10C	17445	17740.2	37.19	**−0.09**	**−0.62**	**−26.17**
n200q10D	22428	22772.4	33.65	**−1.70**	**−1.09**	**−32.58**
n200q10E	20409	20738.2	36.77	**−0.42**	**−0.47**	**−33.02**
n200q10F	22483	22709.4	37.10	**−0.84**	**−0.41**	**−26.63**
n200q10G	18585	18855.3	34.72	**−2.36**	**−0.64**	**−37.16**
n200q10H	22165	22588.2	39.85	**−1.16**	**−1.06**	**−33.12**
n200q10I	19533	19859.3	34.22	**−0.88**	**−1.79**	**−33.47**
n200q10J	20179	20471.6	32.80	**−0.83**	**−1.11**	**−29.42**
Group Average				**−1.22**	**−0.96**	**−31.59**
n300q10A	24942	25148.1	136.01	**−2.21**	**−1.63**	**−32.23**
n300q10B	24413	24802.3	133.15	**−0.27**	**−0.84**	**−32.68**
n300q10C	23212	23418.2	142.24	**−1.61**	**−1.06**	**−34.69**
n300q10D	27080	27614.3	147.46	**−2.79**	**−1.82**	**−32.74**
n300q10E	28643	28914.2	147.16	**−2.31**	**−1.69**	**−32.11**
n300q10F	25843	26213.9	143.07	**−0.97**	**−1.22**	**−24.17**
n300q10G	25631	25814.5	144.66	**−2.05**	**−1.55**	**−24.86**
n300q10H	23590	23795.3	138.41	**−1.89**	**−1.19**	**−32.79**
n300q10I	26018	26358.4	136.85	**−2.21**	**−1.49**	**−31.02**
n300q10J	24050	24466.0	140.90	**−1.01**	**−1.34**	**−29.84**
Group Average				**−1.73**	**−1.38**	**−30.71**
n400q10A	33087	33266.8	393.04	**−2.77**	**−1.94**	**−28.20**
n400q10B	26677	26797.2	347.47	**−2.14**	**−1.5**	**−29.01**
n400q10C	30394	30682.2	399.14	**−1.83**	**−1.46**	**−32.59**
n400q10D	25814	26267.5	400.79	**−2.03**	**−1.97**	**−33.98**
n400q10E	26795	27313.9	355.53	**−1.50**	**−1.79**	**−26.86**
n400q10F	**28107**	28910.0	361.85	0.29	**−1.28**	**−29.19**

(*continued*)

Table 3 (continued)

Instances	GRASP+VND			DM-GRASP+VND		
	Best solution	Average solution	Average time (s)	Diff % best	Diff % average	Diff % time
n400q10G	25697	26220.6	398.57	**−2.27**	**−2.78**	**−29.88**
n400q10H	27158	27773.1	393.40	**−1.27**	**−1.93**	**−29.20**
n400q10I	**30115**	30898.7	387.77	0.31	**−1.13**	**−31.95**
n400q10J	27655	28059.0	383.00	**−2.65**	**−1.86**	**−30.00**
Group Average				**−1.59**	**−1.76**	**−30.08**
n500q10A	29874	30661.4	825.94	**−1.06**	**−1.35**	**−29.82**
n500q10B	28559	29042.9	846.08	**−1.07**	**−1.58**	**−32.18**
n500q10C	32360	33162.5	867.24	**−0.91**	**−1.79**	**−33.36**
n500q10D	32750	33074.3	863.71	**−1.93**	**−1.78**	**−31.23**
n500q10E	32298	32667.1	881.04	**−1.84**	**−1.24**	**−32.09**
n500q10F	30856	31354.6	813.26	**−1.37**	**−1.16**	**−37.12**
n500q10G	28879	29123.4	885.22	**−1.81**	**−1.65**	**−32.48**
n500q10H	38579	39023.5	849.81	**−1.69**	**−1.72**	**−29.80**
n500q10I	32718	33217.7	858.72	**−1.19**	**−1.79**	**−36.28**
n500q10J	**32407**	33131.7	873.12	0.38	**−1.24**	**−33.94**
Group Average				**−1.25**	**−1.53**	**−32.83**
Overall Average				**−1.46**	**−1.34**	**−30.63**

terms of average quality of the solution, the average percentage difference between these heuristics was 1.34%.

The two main reasons for the faster behavior of the hybrid data mining approaches are the same of the previous application describe in section "Hybrid DM-GRASP": the adapted construction is faster than the original one, and the quality of the solutions constructed after the data mining process is better than that of the original construction causing less effort for local search.

Figures 14 and 15 illustrate the behavior of the construction and local search phases, for both GRASP+VND and DM-GRASP+VND, in terms of the solution cost values obtained, along the execution of 1000 iterations for the n500q10G instance with a specific random seed. The behavior is the same as the one presented in section "Hybrid DM-GRASP". In Fig. 14, the GRASP+VND heuristic behaves similarly throughout the iterations. Furthermore, both strategies (see Fig. 15) have exactly the same behavior until the 500th iteration, where the data mining procedure is done. From this point on, the quality of the solutions obtained by DM-GRASP+VND, both in construction and local search procedures, is improved.

Figure 16 presents a comparison between these approaches, based on the time-to-target plots (TTT-plots) [3], which were used to analyze the behavior of algorithms in section "Hybrid DM-GRASP". The instance n500q10G was used, with the target value equal to 29123, and each strategy was run 100 times (with different random seeds) until the target solution cost value was reached for a specific instance.

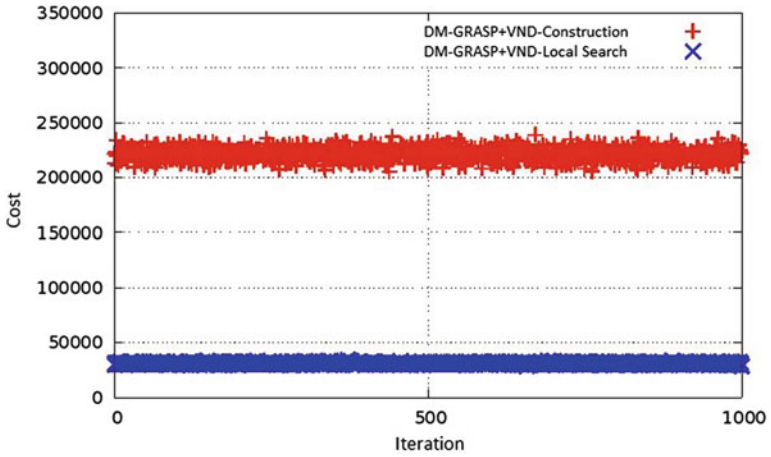


Fig. 14 One execution of GRASP+VND for instance n500q10G

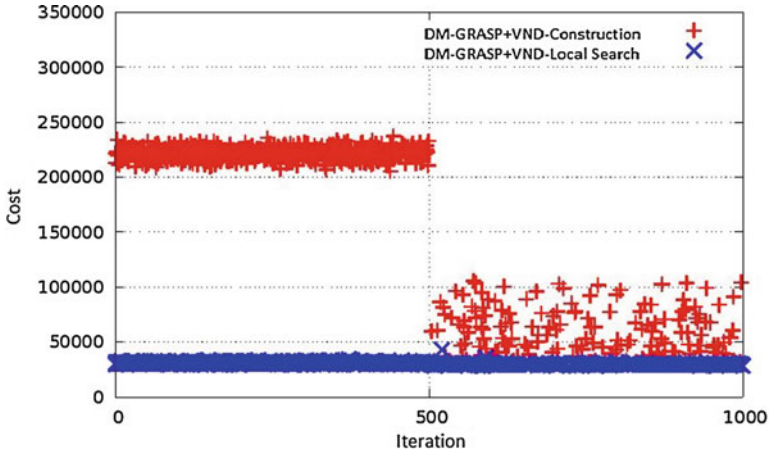


Fig. 15 One execution of DM-GRASP+VND for instance n500q10G

It is possible to observe that the data mining strategy outperformed the original GRASP+VND. For example, the cumulative probability for DM-GRASP+VND to find the prefixed target in 1000 s is almost 100%, while the same probability for the original GRASP+VND is of about 55%.

Hybrid DM-ILS

This section shows the results obtained when applying data mining to another SLS metaheuristic named ILS.

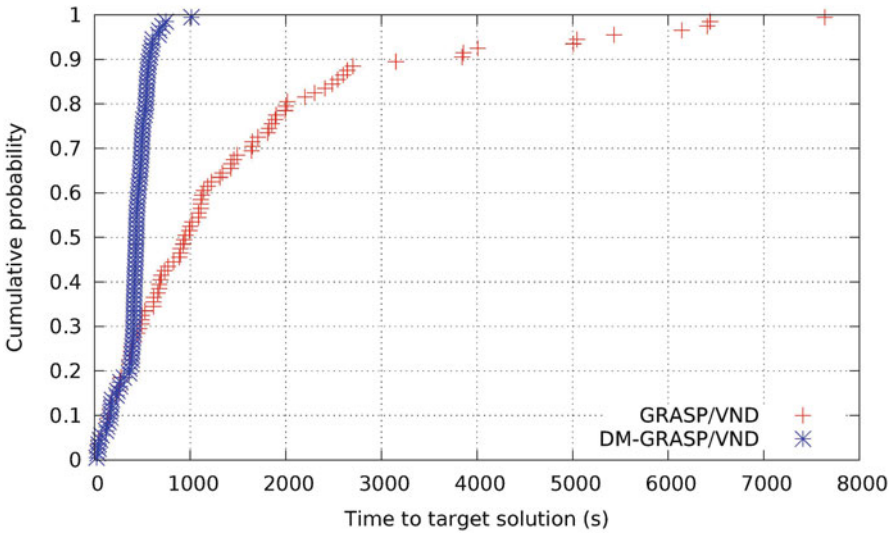


Fig. 16 Time-to-target plot for an intermediate target for instance n500q10G

The ILS is a metaheuristic method for obtaining solutions to combinatorial optimization problems that has been used with considerable success in many computationally intractable problem [37].

This metaheuristic constructs an initial solution and performs an iterative process where, in each iteration, a perturbation is applied to the incumbent solution and a local search is performed starting from the perturbed solution. An initial solution is obtained by a constructive algorithm, and a solution of better quality is searched in its neighborhood. Then, while a stopping criterion is not satisfied, at each iteration, a perturbation is applied in the current solution, and then its neighborhood is explored by a local search algorithm. If the acceptance criterion is satisfied, the solution obtained after the local search step becomes the current solution.

The ILS heuristic developed in [18] for the set covering with pairs problem (SCPP) achieved good performance in both quality of solution and computational time. So data mining was used with this heuristic to verify if the results obtained by an ILS heuristic with good performance could be improved.

The SCPP has been defined in [25] as a generalization of the set covering problem, known to be NP-hard. In the SCPP, the elements have to be covered by specific pairs of objects, instead of a single object.

The problem of finding a minimum sized set of monitoring vertices at which to place equipment to allow the estimation rates for all hop-on and hop-off paths was also modeled as a SCPP in [7].

The SCPP can be formally defined using a coverage function. Let $U = \{u_1, \cdots, u_n\}$ be a ground set of elements to be covered, and let $A = \{a_1, \cdots, a_m\}$ be a set of objects, where each object $a_i \in A$ has a nonnegative cost c_i. The cover

function $\mathscr{C} : A \times A \rightarrow 2^U$ specifies, for each pair of objects $\{a_i, a_j\} \subseteq A$, the subset of elements from U which is covered by this pair. The SCPP aims to find a collection of objects $S \subseteq A$ such that $\bigcup_{\{i,j\} \subseteq S} \mathscr{C}(i, j) = U$ and $\sum_{i \in S} c_i$ are minimized.

Following, the construction and local search procedures, the perturbation strategy, and the stopping and acceptance criteria developed for the ILS heuristic for SCPP [18] are presented.

The constructive add-drop heuristic (ADH) builds an initial solution. A feasible solution S is initially created containing all objects from A. The objects in A are sorted in non-decreasing order according to their associated cost and stored in A'. The first object of the ordered set A' is selected and removed from the set. Then, it is checked if the removal of the this object from solution S will cause some element of U to be uncovered. If the removal of this object does not turn the solution infeasible, this object is removed from S. The algorithm ends when there are no more candidates in the set A'.

Two local search heuristics were developed: drop-add local search (DALS) and change object local search (COLS), which are used one after the other.

The DALS algorithm receives as input parameters an S solution obtained by ADH constructive algorithm earlier described, a constant $\gamma \in [0, 1]$, and a positive integer k indicating the stopping criterion of the algorithm. The first step of the algorithm consists in removing $\gamma \times |S|$ objects of the current solution S which forms the set Q. These elements are removed from the current solution and are prevented from being removed in the next I iterations by inserting them into a tabu list lt. After this step, the elements of Q are removed from the solution set, making it infeasible because some of the elements from U will become uncovered. The reconstruction procedure recovers the feasibility of the solution by considering only the elements uncovered in U and generates the solution S' determining the set of objects that should be included in the solution to cover these uncovered elements at a lower cost.

If the solution S' has a lower cost that the best solution cost found until the current iteration, the best solution $S*$ is updated. This process is repeated while better solutions are found, being interrupted when the algorithm runs without improving the solution cost for k iterations.

The COLS algorithm determines the best solution in the neighborhood of $S*$, each time, by removing an object from $S*$, and the solution is reconstructed using a heuristic drawn between the two heuristics Best Pair Heuristic (BPH) and Modified Best Object Heuristic (MBOH) [18]. These heuristics consist in performing iterations until a feasible solution is obtained. In each iteration, element(s) is(are) selected for incorporation in the solution according to different strategies.

The perturbation strategy consists of removing a percentage of elements from the current solution and reconstructing the solution using the BPH or MBOH heuristics which are randomly chosen.

As a stopping criterion, the following criteria may be used: execution timeout, reaching a fixed number of iterations, and achieving a number of followed iterations without improvement.

In each iteration, the current solution is updated by the solution generated after the local search if its cost is better than the cost of the current solution.

The general idea of using data mining with ILS heuristic was to perform a number of iterations of the original ILS heuristic to form a repository of good quality solutions. Subsequently, patterns of this repository using a data mining technique are obtained. The FPmax* [19] algorithm was used for extracting maximal frequent itemsets. These patterns are used in subsequent iterations of the ILS in local search procedures.

A repository rep contains the M best solutions found by ILS in $n/2$ early iterations, and P contains the patterns mined from rep. Seven following strategies were developed.

1. ILS-DM-AUnion10: a super-pattern SP is generated composed by the intersection of all patterns from P, and the elements from P are not allowed to be removed during the local search COLS.
2. ILS-DM-A: the elements that belong to a pattern from P are not allowed to be removed from the solution during local search COLS. In each iteration, one pattern from P is selected in round-robin mode.
3. ILS-DM-B: the elements belonging to a pattern from P are not allowed to be removed from the solution during local search DALS and are inserted in a tabu list at the beginning of the local search (after a few iterations, the element belonging to the tabu list loses its validity and leave the tabu list and may be removed from solution). In each iteration, one pattern from P is selected in round-robin mode.
4. ILS-DM-B-PatternTT: defined as the previous strategy, however, in this approach, the elements belonging to the pattern remains in the tabu list until the end of the local search procedure.
5. ILS-DM-AB: the patterns are used in local search COLS and DALS. The ILS-MD-A is applied to COLS and the ILS-MD-B to DALS. In each iteration, one pattern from P is selected in round-robin way.
6. ILS-DM-AB-PatternTT: defined as the previous strategy, however, in this approach, the elements belonging to the pattern remain in the tabu list until the end of the local search procedure.

Computational Results

The instances scp_p were created in [18]. They have between 50 and 28,160 elements, and the number of objects ranges from 192 to 11,264. The value of the parameter p affects the number of pairs of each instance. Lower p values mean greater number of pairs in the instances. For example, for $p = 25\%$, the average

number of pairs is 238,810 and for $p = 75\%$ is 77,543. Each group has 24 scp_p instances.

In all strategies, the repository of solutions to be mined contains ten solutions, and the ten largest mined patterns were used. A minimum support equal to 90% was used for data mining.

The stopping criterion used for all strategies was running 100 iterations, and to evaluate each strategy, ten runs with different random seeds were executed.

In each table of results, the first three lines report the results obtained for each instance group, showing the percentage difference between the value obtained by running the hybrid heuristic and the value obtained using the heuristic ILS. Positive percentage differences denote that hybrid heuristics obtained better results than the heuristic ILS, and negative values denote worse values. In the following lines, the average percentage difference and the number of wins, losses, and draws of each hybrid heuristic are presented.

Tables 4 and 5 present the average values of solution and computational times obtained by the strategies that use patterns in local search.

The results obtained related to the solution's quality, when using the patterns in local search, were similar to those obtained by the ILS heuristic, but different results were obtained with respect to computational times. Heuristics ILS-MD-AUnion10, ILS-MD-A, ILS-MD-AB, and ILS-MD-AB-PatternTT had significantly lower computational times than the ILS heuristic, while the ILS-MD-B strategy had larger computational times.

The behaviors of the heuristics ILS-MD-A and ILD-MD-AB were studied in more detail because both significantly decreased the computational times. Heuristic ILS-MD presented computational time 25% lower than the ILS heuristic, and the percentage difference of the average quality of the solution was only 0.032% below the ILS heuristic solution. Heuristic ILS-MD-AB presented computational time

Table 4 The mean values of solutions using patterns in local search

Group	AUnion10	A	A-B	A-B-PatternTT	B	B-PatternTT
scp_25	−0.059	0.004	0.01	0.08	0.094	0.13
scp_50	−0.089	−0.073	−0.046	0.007	0.04	0.023
scp_75	−0.103	−0.028	0.077	0.063	0.031	0.047
Average	−0.084	−0.032	0.014	0.05	0.055	0.067
Wins/losses/draws	23/46/3	28/39/5	40/30/2	42/27/3	47/22/3	44/25/3

Table 5 The mean values of computational times using patterns in local search

Group	AUnion10	A	A-B	A-B-PatternTT	B	B-PatternTT
scp_25	29.74	26.84	18.09	18.05	−10.58	0.19
scp_50	26.86	23.78	15.72	15.16	−14.93	1.1
scp_75	30.1	26.74	15.4	15.76	−2.83	1.48
Mean	28.9	25.79	16.4	16.32	−9.45	0.93
Wins/losses/ties	72/0/0	72/0/0	71/1/0	72/0/0	18/54/0	49/23/0

16% lower than the ILS heuristic, and the percentage difference of the average quality of the solution was 0.014% above the ILS heuristic solution.

In the first experiment, each hybrid heuristic was executed during the same time that the ILS heuristic. Table 6 presents the solution values obtained for these heuristics. It can be seen that for both heuristics, solutions of better quality were obtained.

In another experiment, 100 iterations were performed for the instance scp41_25, and the computational times used by the perturbation phase and local searches COLS and DALS were obtained in each iteration. In Figs. 17, 18, and 19, the iteration number is shown on the x-axis, and the execution time of each phase is presented on the y-axis.

It can be observed that in the execution of the heuristic ILS-DM-A, which uses the patterns extracted on the local search COLS, there is a decrease in COLS's execution time in the fifty iteration when the patterns are in use. In the execution of heuristic ILS-MD-AB, where patterns are used both in COLS and DALS, one can observe a decrease in execution time in both local searches also in 50 iteration. These experiments indicate that using patterns decreased the computational time, and the solutions have presented similar quality to the ones obtained by the original heuristic ILS.

Table 6 Mean values of solution of hybrid heuristics using the same time of ILS

Group	A	A-B
scp_25	0.104	0.192
scp_50	0.055	0.123
scp_75	0.083	0.245
Mean	0.084	0.185
Wins/losses/draws	53/17/2	60/12/0

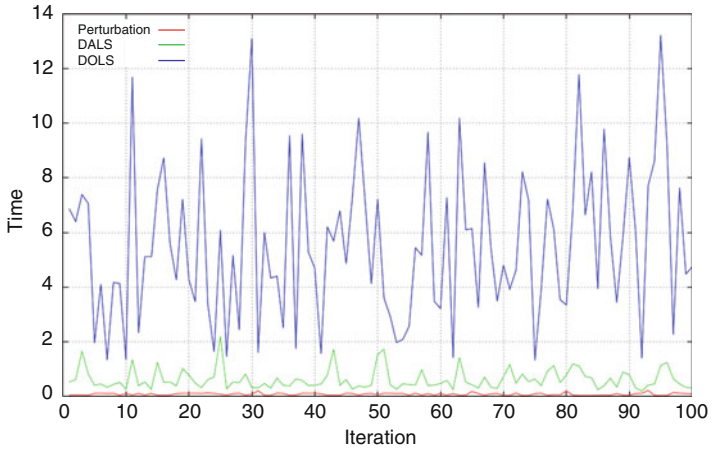


Fig. 17 Computational time for ILS

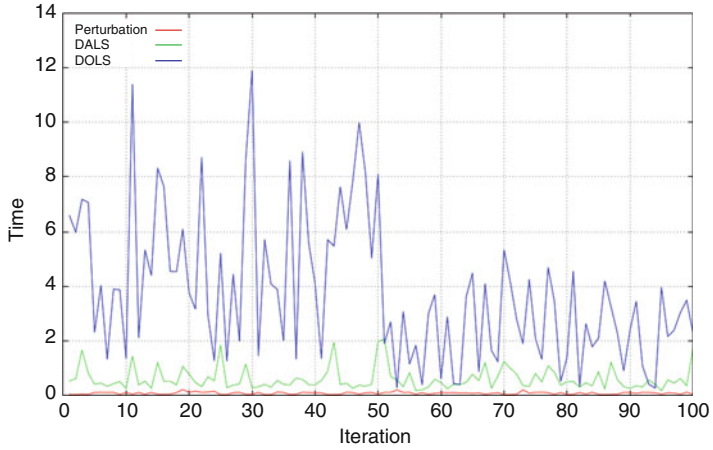


Fig. 18 Computational time for ILS DM-A

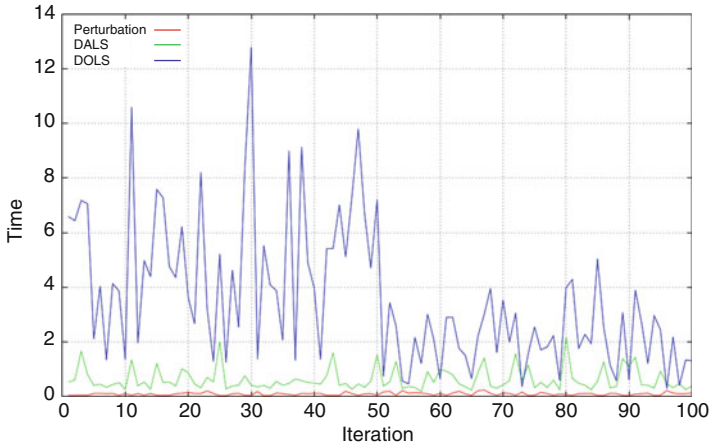


Fig. 19 Computational time for ILS DM-A-B

The graphics in Figs. 20 and 21 show a comparison between the heuristics ILS, ILS-DM-A, and ILS-DM-AB. Time-to-target (TTT) plots [3] were used to compare the behavior of these heuristics through their execution times. The graphics shown in these figures were generated by running each metaheuristic 100 times, using the instance scp41_25, with different random seeds, with two target values: (a) an easy target and (b) a difficult target. It is observed that both ILS-DM-A and ILS-DM-AB reach the targets faster than ILS, proving that these hybrid heuristics require less computational time to achieve similar quality solutions. The heuristic ILS-DM-AB hits the easy target faster than the heuristic ILS-DM-A, while both have similar behavior when using the more difficult target.

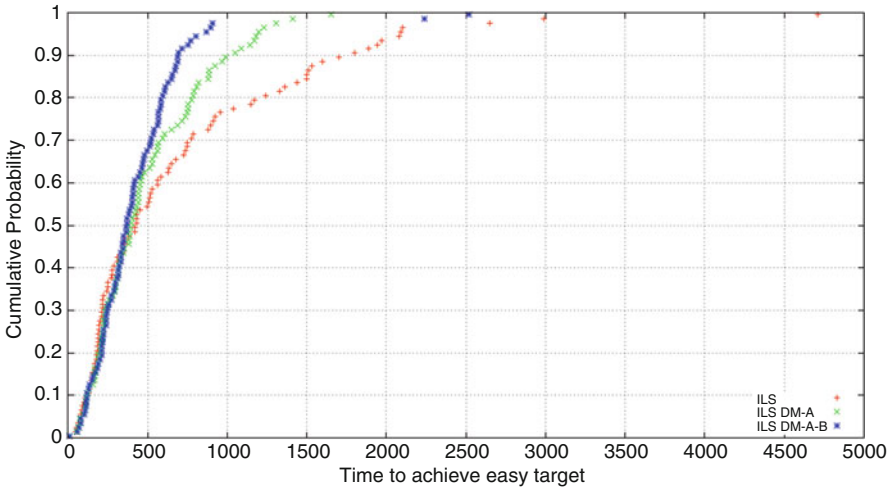


Fig. 20 TTT plot for an easy target

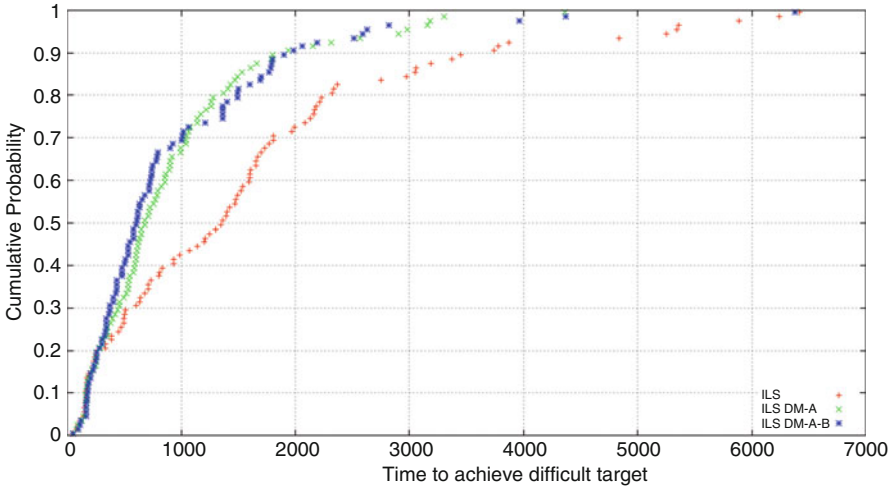


Fig. 21 TTT plot for a difficult target

Memory-Based Heuristics

This section presents ways to incorporate data mining with memory-based heuristics. Data mining was integrated with two GRASPs that were hybridized with a path-relinking strategy, which brings the explicit use of memory to the memoryless

GRASP heuristic. In both problems, data mining was used only in the construction phase as already explained in previous sections. The results obtained for the first problem showed that using data mining was able to improve the quality of the solutions and also the computational time. The introduction of data mining in the second problem made it possible to achieve results with similar quality but in less computational time.

Hybrid DM-GRASP+PR

As described in section "Memoryless Heuristics", GRASP is a memoryless metaheuristic, i.e., it does not learn from the solutions found in previous iterations.

Path relinking was originally proposed by Glover [15] as an intensification strategy exploring trajectories connecting solutions obtained by tabu search or scatter search approaches. Starting from one or more elite solutions, path relinking generates paths leading toward other elite solutions and explores them in the search for better solutions. To generate paths, moves are selected to introduce attributes in the current solution that are present in the elite guiding solution. Path relinking is a strategy that seeks to incorporate attributes of high-quality solutions, by favoring them in the selected moves. Laguna and Martí [33] were the first to use path relinking within a GRASP strategy. Several extensions, improvements, and successful applications of this technique can be found in the literature [9, 45].

Basically, path relinking is applied to a pair of solutions $\{x_i, x_g\}$ by starting from the initial solution x_i and gradually incorporating attributes from the guide solution x_g to x_i, until x_i becomes equal to x_g. The best solution found in this path is returned by this algorithm. To use path relinking within a GRASP procedure, an elite set L is maintained, in which good solutions found in previous GRASP iterations are stored.

Two basic strategies for introducing path relinking into GRASP may be used [42]: (a) performing path relinking after each GRASP iteration using a solution from the elite set and a local optimum obtained after the GRASP local search and (b) applying path relinking to all pairs of elite solutions, either periodically or after all GRASP iterations terminate.

The DM-GRASP+PR and MDM-GRASP+PR algorithms [5] are extensions of the GRASP+PR implementation, proposed in [46], to solve the 2-path network design problem (2PNDP) [10]. Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes and E is the set of edges. A k-path between nodes $s, t \in V$ is a sequence of at most k edges connecting them. Given a nonnegative weight function $w : E \to \mathbb{R}_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the 2PNDP consists in finding a minimum-weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair in D. Applications of the 2PNDP can be found in the design of communication networks, in which paths with few edges are sought to enforce high reliability and small delays. The decision version of the 2PNDP has been proved to be NP-complete by Dahl and Johannessen [10].

The GRASP+PR heuristic for the 2PNDP problem developed in [46] runs, in each iteration, three procedures: construction, local search, and path relinking. The greedy randomized construction algorithm computes one shortest 2-path at a time. Solution x is computed from scratch using edge weights w' that are initially equal to the original weights w. Each iteration starts by the random selection of a pair (a, b) still to be routed. The shortest path P from a to b using the modified weights w' is computed, and the weights of the edges in P are temporarily set to 0 to avoid that edge weights be considered more than once in the solution cost during the forthcoming iterations. Pair (a, b) is removed from the set of origin-destination pairs to be routed, and the edges in P are inserted into the solution under construction. This construction procedure stops when a 2-path has been computed for every origin-destination pair.

The local search phase attempts to improve the solutions built greedily during the construction phase. Each solution x may be viewed as a collection of 2-paths for each pair (a, b). Given any solution x, its neighbor solutions x' may be obtained by replacing any 2-path in x by another 2-path between the same origin-destination pair. This procedure stops when it is not possible to change any origin-destination pair without improvement.

The path-relinking phase is applied to the solution obtained by local search and to a randomly selected solution from the pool L twice (one using the latter as the starting solution and the other using the former). The local optimal solution obtained by local search and the best solutions found along each relinking trajectory are considered as candidates for insertion into the pool. A solution is inserted in the pool if it is different from all solutions of the pool and its cost is better than the cost of the worst solution of the pool.

The algorithm in Fig. 22 illustrates the DM-GRASP+PR procedure [5] for the 2PNDP. In line 3, the elite set used for data mining is initialized with the empty set. The loop from line 4 to line 21 corresponds to the elite set generation phase, in which GRASP with path relinking is performed for n iterations. The original construction method is executed in line 5, followed by the local search method in line 6 and the path-relinking procedure executed from line 8 to line 15. The elite set M, composed of d solutions, is updated in line 16. A solution is inserted in the elite set if it is not already in the set and its cost is better than the worst cost found in the set. In line 18, the best solution is updated, if the new generated solution presents a better cost than the best solution found in previous iterations. In line 22, the data mining procedure extracts t patterns from the elite set, which are inserted in decreasing order of pattern size in the set of patterns. The loop from line 23 to line 38 corresponds to the hybrid phase. In line 24, one pattern is picked from the set of patterns in a round-robin way. Then, the adapted construction procedure is performed in line 25, using the selected pattern. In line 26, the local search is executed. From line 28 to 33 the path-relinking procedure is executed. If a better solution is found, the best solution is updated in line 35. After the execution of all iterations, the best solution is returned in line 39.

```
procedure DM-GRASP+PR(n, d, t)
1.  L ← ∅;
2.  f* ← ∞;
3.  M ← ∅;
4.  for k = 1, ..., n do
5.      x ← GreedyRandomizedConstruction2Path;
6.      x ← LocalSearch2Path(x);
7.      Update the pool of elite solutions L with x;
8.      if |L| ≥ 2 then
9.          Select at random an elite solution y from the pool L;
10.         x1 ← PathRelinking(x,y);
11.         Update the pool of elite solutions L with x1;
12.         x2 ← PathRelinking(y,x);
13.         Update the pool of elite solutions L with x2;
14.         Set x ← argmin{f(x), f(x1), f(x2)};
15.     end if;
16.     UpdateElite(M, x, d);
17.     if f(x) < f* then
18.         x* ← x;
19.         f* ← f(x);
20.     end if;
21. end for;
22. patterns_set ← Mine(M, t);
23. for k = 1, ..., n do
24.     pattern ← SelectNextLargestPattern(patterns_set);
25.     x ← AdaptedGreedyRandomizedConstruction2Path(pattern);
26.     x ← LocalSearch2Path (x);
27.     Update the pool of elite solutions L with x;
28.     Select at random an elite solution y from the pool L;
29.     x1 ← PathRelinking(x,y);
30.     Update the pool of elite solutions L with x1;
31.     x2 ← PathRelinking(y,x);
32.     Update the pool of elite solutions L with x2;
33.     Set x ← argmin{f(x), f(x1), f(x2)};
34.     if f(x) < f* then
35.         x* ← x;
36.         f* ← f(x);
37.     end if
38. end for;
39. return x*;
```

Fig. 22 Pseudo-code of the hybrid DM-GRASP with path relinking for the 2PNDP

The DM-GRASP+PR and MDM-GRASP algorithms proposed in [5] are similar to DM-MSH and MDM-MSH already presented in section "Strategies to Use Data Mining in Heuristics".

Computational Results

The computational results obtained for GRASP+PR, DM-GRASP+PR, and MDM-GRASP+PR strategies were compared in [5]. The 25 instances used, similar to the instances generated in [46], are complete graphs with $|V| \in \{100, 200, 300, 400, 500\}$. The edge costs were randomly generated from the uniform distribution on the interval (0, 10], and $10 \times |V|$ origin-destination pairs were randomly chosen.

The three strategies were run ten times with a different random seed in each run. Each strategy executed 1000 iterations.

In the mining strategies, the size of the elite set (d), from which the patterns are mined, and the size of the set of patterns (t) were set to 10. A set of edges was considered a pattern if it was present in at least two of the elite solutions. In the MDM-GRASP+PR approach, the elite set was considered stable when 1% of the total number of iterations is executed without change in the elite set d.

In Table 7, the results obtained by GRASP+PR, DM-GRASP+PR, and MDM-GRASP+PR heuristics are presented. The first column shows the instance identifier. The second, third, and fourth columns have the best cost values, the best average cost, and the best average execution time (in seconds) obtained by, at least, one of the three approaches. The remaining columns report the percentage difference (Diff %) of the GRASP+PR, DM-GRASP+PR, and MDM-GRASP+PR over the best results found. The percentage difference is evaluated as follows:

$$Diff\ \% = \frac{(value_{heuristic} - best_value)}{best_value} * 100,$$

where

$$best_value = \min\{value_{GRASP+PR}, value_{DM-GRASP+PR}, value_{MDM-GRASP+PR}\}$$

The intermediate rows show the partial averages of the percentage differences for each group of the same size instances, and the last row of the table presents the overall average. The smallest values considering the best solution, the average solution, and the average time, i.e., the best results among them, are boldfaced.

The results presented in Table 7 showed that the mining strategies proposed were able to improve all results obtained by the original GRASP+PR, including solution costs and execution times. MDM-GRASP+PR found 18 better values and DM-GRASP+PR found four. Furthermore, MDM-GRASP+PR found 24 better results and DM-GRASP+PR reached just one. In relation to running times, DM-GRASP+PR was faster than MDM-GRASP+PR in 19 instances, while MDM-GRASP+PR was faster than DM-GRASP+PR in 6 instances. These results indicated that, in general, the MDM-GRASP+PR strategy produced better solutions than the other strategies. MDM-GRASP+PR was, on average, 1.3% slower than DM-GRASP+PR which is not very significant in terms of the heuristic performance.

Figures 23, 24, and 25 illustrate the behavior of the construction, local search, and path-relinking phases, in terms of the cost values obtained, by GRASP-PR, DM-

Table 7 GRASP+PR, DM-GRASP+PR, and MDM-GRASP+PR results

				GRASP+PR			DM-GRASP+PR			MDM-GRASP+PR		
Instances	Best sol.	Best avg.	Best avg. time	Diff % best	Diff % avg.	Diff % time	Diff % best	Diff % avg.	Diff % time	Diff % best	Diff % avg.	Diff % time
a100-1	674	681.9	37.4	0.74	0.82	18.18	0.30	0.01	**0.00**	**0.00**	**0.00**	2.94
a100-10	659	665.2	36.1	0.61	0.69	19.94	0.46	0.53	**0.00**	**0.00**	**0.00**	3.88
a100-100	666	670.0	38.9	0.60	0.69	20.05	**0.00**	0.04	**0.00**	0.15	**0.00**	3.86
a100-1000	640	646.7	36.1	0.63	0.49	19.11	0.16	0.05	**0.00**	**0.00**	**0.00**	3.88
a100-10000	658	665.4	36.9	0.91	0.57	18.16	0.46	0.17	**0.00**	**0.00**	**0.00**	4.07
Group average				0.70	0.65	19.09	0.27	0.16	**0.00**	**0.03**	**0.00**	3.72
a200-1	1379	1383.9	161.9	0.51	0.58	24.34	**0.00**	0.05	**0.00**	0.07	**0.00**	0.80
a200-10	1362	1372.5	166	0.88	0.98	24.28	**0.00**	0.26	**0.00**	**0.00**	**0.00**	0.66
a200-100	1352	1360.7	157.4	0.67	0.64	25.41	0.15	0.10	**0.00**	**0.00**	**0.00**	3.30
a200-1000	1356	1364.0	158.6	0.52	0.77	25.85	0.15	0.29	**0.00**	**0.00**	**0.00**	1.07
a200-10000	1363	1374.3	166.5	0.88	0.95	24.32	0.44	0.23	**0.00**	**0.00**	**0.00**	0.06
Group average				0.69	0.78	24.84	0.15	0.19	**0.00**	**0.01**	**0.00**	1.18
a300-1	2081	2099.3	401.9	1.20	0.84	28.54	**0.00**	0.15	**0.00**	0.05	**0.00**	1.87
a300-10	2122	2132.1	401.3	0.57	0.75	28.36	**0.00**	0.08	**0.00**	0.14	**0.00**	2.22
a300-100	2069	2076.3	404.2	0.92	0.96	28.10	0.14	0.29	2.00	**0.00**	**0.00**	**0.00**
a300-1000	2076	2090.3	395.9	1.16	0.74	30.36	0.19	0.20	0.78	**0.00**	**0.00**	**0.00**
a300-10000	2060	2075.1	399.9	0.83	0.85	28.91	0.34	0.15	**0.00**	**0.00**	**0.00**	1.03

(*continued*)

Table 7 (continued)

				GRASP+PR			DM-GRASP+PR			MDM-GRASP+PR		
Instances	Best sol.	Best avg.	Best avg. time	Diff % best	Diff % avg.	Diff % time	Diff % best	Diff % avg.	Diff % time	Diff % best	Diff % avg.	Diff % time
Group average				0.93	0.83	28.85	0.14	0.17	**0.56**	**0.04**	**0.00**	1.02
a400-1	2786	2791.4	749.8	0.75	0.89	33.48	0.07	0.22	2.65	**0.00**	**0.00**	**0.00**
a400-10	2819	2844.1	780.4	1.03	0.72	28.61	0.50	0.13	**0.00**	**0.00**	**0.00**	4.05
a400-100	2803	2808.9	799.7	0.54	0.90	28.32	**0.00**	0.36	6.92	**0.00**	**0.00**	**0.00**
a400-1000	2793	2810.9	797.9	1.04	0.80	28.21	0.25	0.20	3.40	**0.00**	**0.00**	**0.00**
a400-10000	2793	2810.9	797.9	2.26	2.27	28.96	1.83	1.65	1.37	**0.00**	**0.00**	**0.00**
Group average				1.12	1.12	29.52	0.53	0.51	2.87	**0.00**	**0.00**	**0.81**
a500-1	3567	3576.9	1330.4	0.87	0.83	29.84	0.11	0.08	**0.00**	**0.00**	**0.00**	1.43
a500-10	3566	3580.1	1302.5	0.81	0.77	31.49	0.20	0.02	**0.00**	**0.00**	**0.00**	3.40
a500-100	3572	3583.1	1396.3	0.73	0.82	25.12	0.11	0.04	**0.00**	**0.00**	**0.00**	1.24
a500-1000	3554	3564.2	1332.7	0.53	0.78	29.17	**0.00**	**0.00**	**0.00**	**0.00**	0.02	3.77
a500-10000	3573	3596.1	1337.3	0.90	0.80	31.64	0.20	0.05	**0.00**	**0.00**	**0.00**	6.18
Group average				0.77	0.80	29.45	0.12	0.04	**0.00**	**0.00**	**0.00**	3.21
Overall average				0.84	0.84	26.35	0.24	0.21	**0.68**	**0.02**	**0.00**	1.99

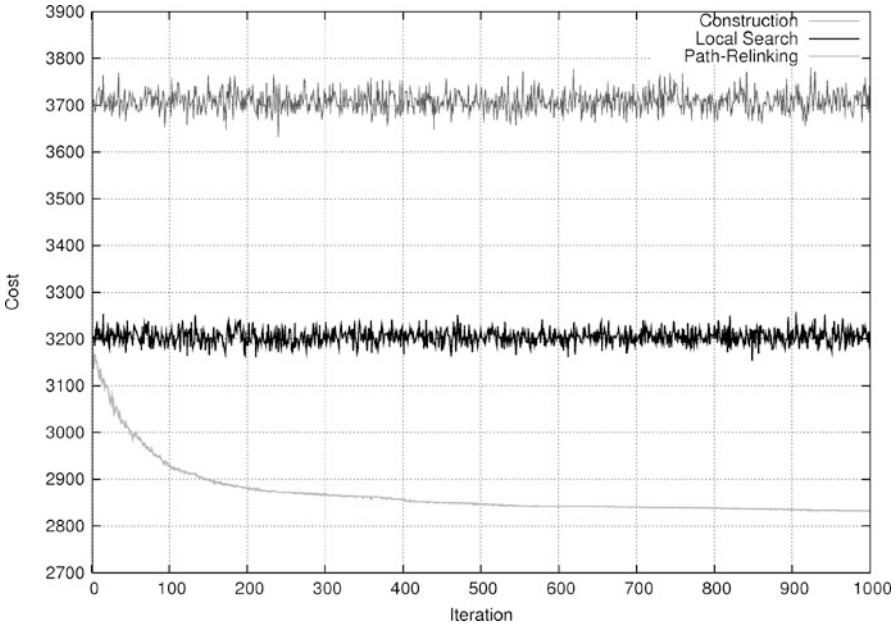


Fig. 23 One execution of GRASP-PR for instance a400-100

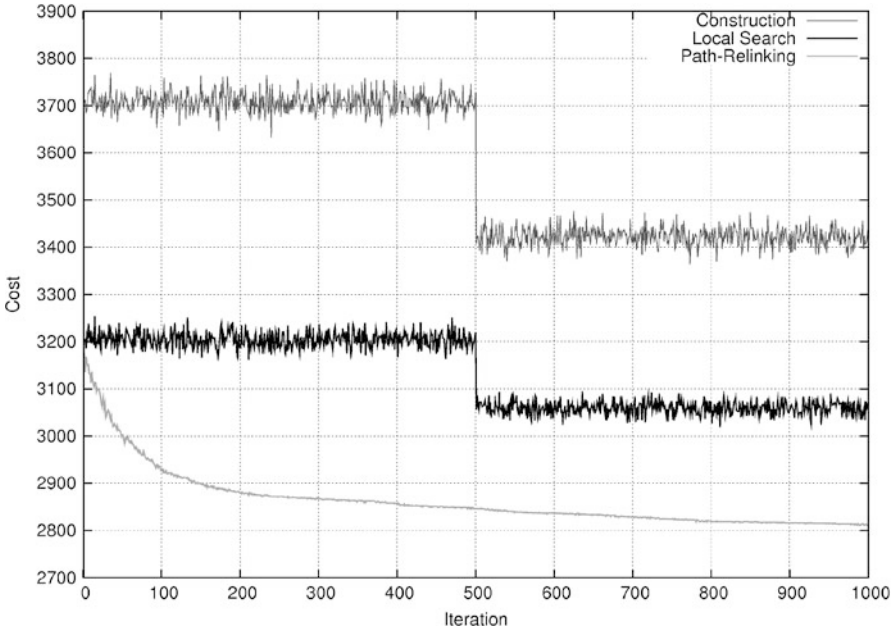


Fig. 24 One execution of DM-GRASP-PR for instance a400-100

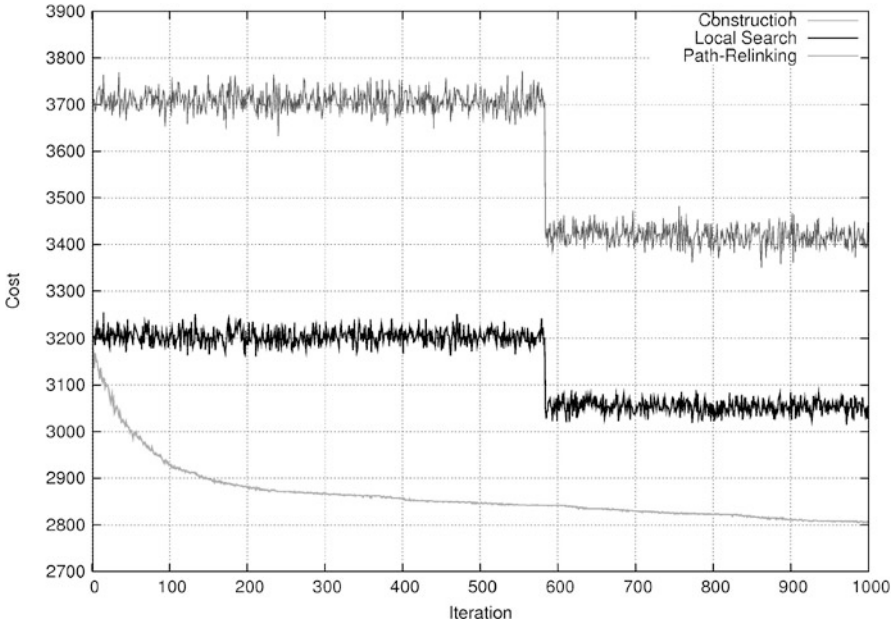


Fig. 25 One execution of MDM-GRASP-PR for instance a400-100

GRASP-PR, and MDM-GRASP-PR throughout the execution of 1000 iterations, for the a400-100 instance, with a specific random seed.

The 2PNDP is a minimization problem, and the figures show that the local search always reduces the cost of the solution obtained by the construction phase. In general, the path-relinking procedure also always reduces the cost obtained after the local search.

Figure 23 indicates that the construction and local search of GRASP-PR present similar behavior throughout the iterations. The path-relinking procedure becomes more effective in reducing the cost, after some iterations, when the pool contains more solutions of better quality. In the last iterations, the path relinking still improves the solution cost but with a smaller rate of improvement, because the pool contains less diverse solutions.

Figure 24 presents the behavior of DM-GRASP-PR strategy. The data mining procedure is executed immediately after iteration 500. After this iteration, the quality of the solutions obtained by the construction, local search, and path-relinking phases is improved.

The behavior of MDM-GRASP-PR is presented in Fig. 25. According to [5], the data mining procedure was activated four times, after the iterations 584, 603, 654, and 822. In this specific case, the improvement due to the activation of the data mining process started to happen immediately after the first mining execution, later than the mining execution by the DM-GRASP-PR. On the other hand, differently from the DM-GRASP-PR, the MDM-GRASP-PR continues to

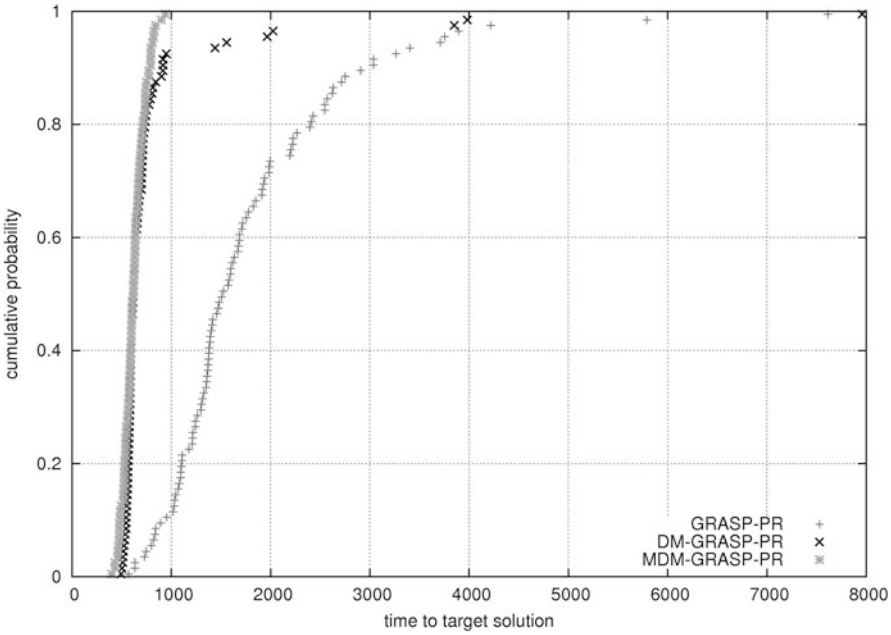


Fig. 26 Time-to-target plot to a difficult target for instance a400-100

slightly and gradually reduce the cost of the solutions obtained by the construction, local search, and path-relinking phases, since patterns are extracted more than once.

Another comparison between the three strategies, based on *time-to-target* (TTT) plots [3], are shown in Fig. 26. The plots presented in this figure were generated by the executions of GRASP-PR, DM-GRASP-PR, and MDM-GRASP-PR, for instance, a400-100, using the same difficult target solution (2820). Each strategy was executed a hundred times [5]. Figure 26 shows that MDM-GRASP-PR behaves better than DM-GRASP-PR, and both present a better behavior than GRASP-PR.

Hybrid DM-HMS

This section describes the incorporation of a data mining procedure into a state-of-the-art heuristic for a specific problem in order to give evidences that, when a heuristic is able to find an optimal solution, or a near-optimal solution with little chance of improvements, the mined patterns used to guide the search are able to help to reach solutions with the same quality in less computational time.

The state-of-the-art algorithm to be the base of this study was the multistart hybrid heuristic proposed in [44] for the classical p-median problem, which

combines elements of different traditional metaheuristics and uses path relinking as a memory-based intensification mechanism.

Given a set F of m potential facilities, a set U of n customers, a distance function $d : U \times F \rightarrow \mathbb{R}$, and a constant $p \leq m$, the p-median problem consists in determining which p facilities to open so as to minimize the sum of the distances from each costumer to its closest open facility. It is a well-known NP-hard problem [32], with numerous applications [53].

At each iteration of the original algorithm, a randomized construction of a solution is performed, which is then submitted to local search. After this, a solution is chosen from the pool of elite solutions, made with some of the best solutions found in previous iterations, and is combined with the solution obtained by the local search through a path-relinking process [15]. Furthermore, after all iterations are completed, this algorithm executes the second phase, called post-optimization, in which elite solutions are combined with each other using the path-relinking process, and the best solution found after the post-optimization phase execution is taken as result.

The hybrid heuristic (HH) was compared with VNS (variable neighborhood search) [23], VNDS (variable neighborhood decomposition search) [24], LOPT (local optimization) [51], DEC (decomposition procedure) [51], LSH (Lagrangian-surrogate heuristic) [50], and CGLS (column generation with Lagrangian surrogate relaxation) [50]. In all cases, the solutions obtained by hybrid heuristic were within 0.1% of the best known upper bounds.

The data mining hybrid heuristic DM-HH was developed using data mining in the construction phase as already showed in previous sections. The elite set is created selecting d best solutions obtained by executing n pure HH iterations. Then, a frequent itemset mining is applied to extract a set of patterns from the elite set.

Next, another n slightly different HH iterations are executed where an adapted construction phase starts building a solution guided by a pattern selected from the set of mined patterns. Initially, all elements of the selected pattern are inserted into the partial solution, from which a complete solution is built executing the standard construction procedure.

The pseudo-code of the DM-HH for the p-median problem is illustrated in Fig. 27. In lines 1 and 2, the elite set of the original heuristic and the elite set for mining are initialized with the empty set. The loop from line 3 to line 14 corresponds to the first phase of the strategy, in which pure HH is performed for $maxit/2$ iterations. The original construction method is executed in line 4, followed by the local search method in line 5. In line 6, a solution is chosen from the pool of elite solutions of the original approach to be combined, in line 8, using the path-relinking process with the solution obtained by the local search. In lines 9 to 13, the elite set of the original algorithm and the elite set for mining, composed of d solutions, are updated with the solution obtained by the path-relinking process and with the solution obtained by the local search. In line 15, the data mining procedure extracts t patterns from the elite set, which are inserted in decreasing order of pattern size in the set of patterns. The loop from line 16 to line 26 corresponds to the second

```
procedure DM_HH(maxit,elitesize)
 1.   elite_set ← ∅;
 2.   elite_set_DM ← ∅;
 3.   for it ← 1 to maxit/2 do
 4.     S ← Construction_p_Median();
 5.     S ← Local_Search_p_Median(S);
 6.     S' ← Select(elite_set);
 7.     if (S' ≠ ∅)
 8.       S' ← Path_Relinking(S,S');
 9.       Update_Elite(elite_set,S');
10.       Update_Elite(elite_set_DM,S');
11.     end if
12.     Update_Elite(elite_set,S);
13.     Update_Elite(elite_set_DM,S);
14.   end for;
15.   patterns_set ← Mine(elite_set_DM, t);
16.   for it ← 1 to maxit/2 do
17.     pattern ← Select_Next_Largest_Pattern(patterns_set);
18.     S ← Adapted_Construction_p_Median(pattern);
19.     S ← Local_Search_p_Median(S);
20.     S' ← Select(elite_set);
21.     if (S' ≠ ∅)
22.       S' ← Path_Relinking(S,S');
23.       Update_Elite(elite_set,S');
24.     end if
25.     Update_Elite(elite_set,S);
26.   end for;
27.   S ←Post_Optimization(elite_set);
28.   return S;
```

Fig. 27 Pseudo-code of the DM-HH

phase of the strategy. In line 17, one pattern is picked from the set of patterns in a round-robin way. Then the adapted construction procedure is performed in line 18, using the selected pattern as a starting point. In line 19, the original local search is executed. After this, a solution is chosen from the pool of elite solutions of the original approach to be combined using the path relinking with the solution obtained by the local search (lines 20 to 25). After all iterations are completed, this algorithm executes the post-optimization in line 27, and the best solution found after the post-optimization phase is taken as result.

The extraction of patterns from the elite set, which is activated in line 15 of the pseudo-code presented in Fig. 27, corresponds to the frequent itemset mining (FIM) task. To execute this task, the FPmax* algorithm [19] was adopted.

The adapted construction is quite similar to the original construction with the difference that, instead of beginning the solution with an empty set, it starts with all elements of the pattern supplied as a parameter. Then, the solution is completed using the original construction method.

Computational Results

The strategies were evaluated using three classes of instances. The first class, named ORLIB, consists of 40 instances and was taken from the OR-Library [6]. Each instance is a different graph with a corresponding value for p. The number of nodes (customers) varies from 100 to 900, and the value of p ranges from 5 to 200. The optimal cost is known for these 40 instances. In the OR-Library, these 40 instances are identified by *pmed01* to *pmed40*.

Instances of the second class, named TSP, are sets of points on the plane. Originally proposed for the traveling salesman problem, they are available at the TSPLIB [41]. Every point is considered both a potential facility and a customer, and the cost of assigning customer c to facility f is simply the Euclidean distance between the points representing c and f (the costs are real values). From the TSP class, the $FL1400$ instances were considered [23, 24], with 1400 nodes and with several different values for p.

The third class is named RW. Originally proposed in [43], it corresponds to completely random distance matrices. In every case, the number of potential facilities (m) is equal to the number of customers (n). The distance between each facility and each customer has an integer value taken uniformly at random from the interval $[1, n]$. Four different values of n were considered: 100, 250, 500, 1000, 1500, and 2000. In each case, several values of p were tested.

Both HH and DM-HH were performed nine times with a different random seed in each execution. Each strategy executed 500 iterations. The size of the elite set for mining (d) and the size of the set of patterns (t) were set to 10. And a set of facilities was considered a pattern if it was present in at least two of the elite solutions.

When executed for the 40 instances from the ORLIB class, both HH and DM-HH reached the optimal solution cost in all nine runs. DM-HH was always faster than HH for all instances, and the standard deviations are quite small, varying from 0.02 to 0.86. On average, DM-HH was 25.06% faster than the HH strategy for the ORLIB instances.

When executed for the 45 instances from the RW class, both HH and DM-HH reached the best known solutions related to quality solution in all nine runs for 23 instances. Out of the 22 instances for which HH and DM-HH presented different results, the DM-HH strategy found 11 better results for best values, and 4 were found by HH. Considering the average results in nine runs, DM-HH found 15 better values and HH found 7. These results show that the DM-HH strategy was able to improve slightly the results obtained by HH for the RW class.

In terms of computational time, again, the DM-HH strategy was faster than HH. On average, for the RW class, DM-HH was 28.26% faster, and the standard deviation for DH-HH computational time varied from 0.03 to 28.33.

The results obtained for the 18 instances from the FL1400 class showed that both strategies reached the best known solutions in all nine runs for only three instances. For the other 15 instances, they obtained slightly different solutions. The HH strategy found six better results for best values and just one was found by DM-HH. Considering the average results, HH found seven better values and DM-HH

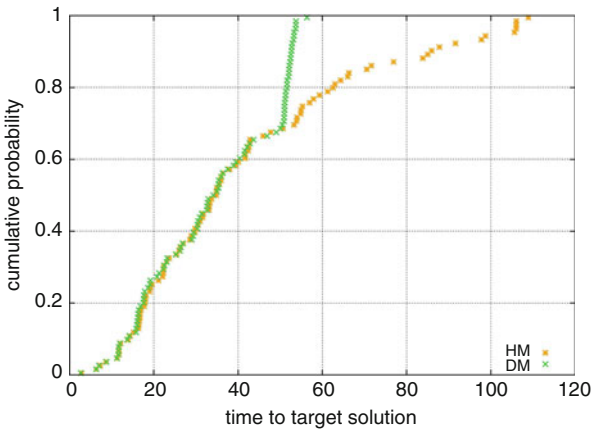


Fig. 28 Time-to-target plot for an easy target

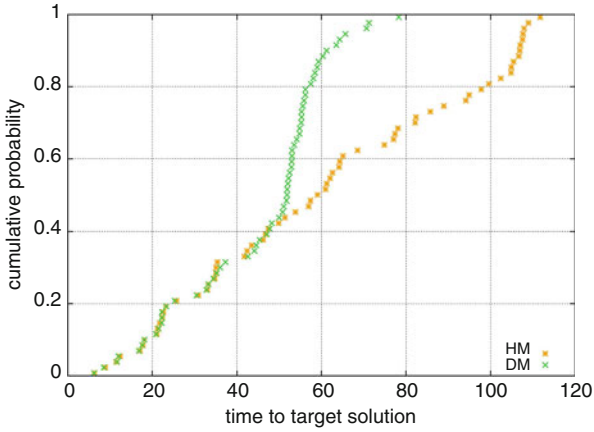


Fig. 29 Time-to-target plot for a difficult target

found four. Differently from ORLIB and RW classes, these results show that the HH strategy, for the FL1400 class, was able to obtain slightly better results than DM-HH.

However, related to time analysis, DM-HH always improved the HH performance. On average, DM-HH was 30.03% faster than HH with the standard deviation varying from 0.61 to 3.87.

Regarding the computational time, for all 113 instances, from ORLIB, RW, and FL1400 classes, the DM-HH proposal was, on average, 27.32% faster than the HH strategy. This result evidences that the DM-HH was able to speed up the original HH strategy, maintaining, and in some cases improving, the quality of the obtained solutions.

The plots presented in Figs. 28 and 29 show another comparison between HH and DM-HH strategies, based on *time-to-target* (TTT) plots [3], which are used to analyze the behavior of randomized algorithms.

They were generated by the executing HH and DM-HH 100 times (with different random seeds), until a target solution cost value was reached for a specific instance. In each run, if the target value was not found in 500 iterations, then the post-optimization was performed until the target value was found or the elite set used for the post-optimization procedure was not updated.

The instance rw1000-p25 was used as the test case, and two targets were analyzed: an easy target (value 24964) and a more difficult one (value 24923).

For the easy target, Fig. 28 shows that HH and DM-HH present similar behaviors until about 50 s when the probability for DM-HH to find the target value starts to be greater than for HH. This happens because, until the data mining procedure is executed in DM-HH, both strategies obtain the same solution in each iteration, but DM-HH starts to find the target value faster when the patterns are used.

For the difficult target, Fig. 29 shows that DM-HH behaves again better than HH. These plots indicate that DM-HH is able to reach difficult solutions faster than HH.

Conclusion

This chapter has presented some ideas to incorporate data mining in local search heuristics. Some applications developed by us have been described which have shown that memoryless heuristics can benefit from the use of data mining by obtaining better solutions in smaller computational times. Also, the results demonstrated that even memory-based heuristics were able to obtain benefits from using data mining by reducing the computational time to achieve good quality solutions.

These results encourage us to try to use data mining in other metaheuristics like genetic algorithms and tabu search, where we can also use the patterns to guide the search. We think of using data mining in exact algorithms like branch and bound, where we may set the value of some variables based on the patterns extracted from solutions previously found.

Analyzing the obtained results, it seems reasonable to expect that incorporating data mining techniques into SLS heuristics can be in general useful. Wolpert and Macread [55] presented a series of no free lunch theorems which establish that, for any algorithm, the high performance on a class of problems is offset by the low performance of another class. So they conclude that all black-box optimization algorithms perform exactly equally well. However, some subsequent studies [14, 30] have shown that these theorems apply to some specific classes of problems and that is expected that real-world problems are not of this particular class. Therefore, we expect that the good results obtained using the data mining techniques in the presented problems may be also achieved in other real-world problems.

Cross-References

- GRASP
- Iterated Local Search
- Multi-start Methods
- Variable Neighborhood Descent

References

1. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the ACM SIGMOD international conference on management of data, Washington, DC, pp 207–216
2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases, Santiago, pp 487–499
3. Aiex R, Resende MGC, Ribeiro CC (2007) TTTplots: a perl program to create time-to-target plots. Optim Lett 1:355–366
4. Aloise D, Ribeiro CC (2011) Adaptive memory in multistart heuristics for multicommodity network design. J Heuristics 17:153–179
5. Barbalho H, Rosseti I, Martins SL, Plastino A (2013) A hybrid data mining GRASP with path-relinking. Comput Oper Res 40:3159–3173
6. Beasley JE (1985) A note on solving large p-median problems. Eur J Oper Res 21:270–273
7. Breslau L, Diakonikolas I, Duffield N, Gu Y, Hajiaghayi M, Johnson DS, Karloff H, Resende MGC, Sen S (2011) Disjoint-path facility location: theory and practice. In: Proceedings of the thirteenth workshop of algorithm engineering and experiments (ALENEX11). SIAM, Philadelphia, pp 60–74
8. Campos V, Piñana E, Martí R (2011) Adaptive memory programming for matrix bandwidth minimization. Ann Oper Res 183:7–23
9. Chaovalitwongse W, Oliveira C, Chiarini B, Pardalos P, Resende MGC (2011) Revised GRASP with path-relinking for the linear ordering problem. J Comb Optim 22:1–22
10. Dahl G, Johannessen B (2004) The 2-path network problem. Networks 43:190–199
11. Eiben A, Smith J (2007) Introduction to evolutionary computing. Springer-Verlag, Berlin
12. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
13. Fleurent C, Glover F (1999) Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS J Comput 2:198–204
14. García-Martínez C, Rodriguez F, Lozano M (2012) Arbitrary function optimisation with metaheuristics. Soft Comput 16:2115–2133
15. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 39:653–684
16. Glover F, Laguna M, Martí R (2003) Scatter search and path relinking: advances and applications. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. International series in operations research & management science, vol 57. Springer, Boston, pp 1–35
17. Goethals B, Zaki MJ (2004) Advances in frequent itemset mining implementations: report on fimi'03. SIGKDD Explor Newsl 6:109–117
18. Gonçalves LB, Martins SL, Ochi LS (2010) Effective heuristics for the set covering with pairs problem. Int Trans Oper Res 17:739–751

19. Grahne G, Zhu J (2003) Efficiently using prefix-trees in mining frequent itemsets. In: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations, Melbourne, Florida, USA
20. Guerine M, Rosseti I, Plastino A (2014) Extending the hybridization of metaheuristics with data mining to a broader domain. In: Proceedings of the 16th international conference on enterprise systems, Lisboa, pp 395–406
21. Han J, Kamber M (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann, San Francisco
22. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the ACM SIGMOD international conference on management of data, Washington, DC, pp 1–12
23. Hansen P, Mladenović N (1997) Variable neighborhood search for the p-median. Locat Sci 5:207–226
24. Hansen P, Mladenović N, Perez-Brito D (2001) Variable neighborhood decomposition search. J Heuristics 7:335–350
25. Hassin R, Segev D (2005) The set cover with pairs problem. In: Sarukkai S, Sen S (eds) FSTTCS 2005: foundations of software technology and theoretical computer science. Lecture notes in computer science, vol 3821. Springer, Berlin/Heidelberg, pp 164–176
26. Hernández-Pérez H, Salazar-González JJ (2004) A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discret Appl Math 145:453–459
27. Hernández-Pérez H, Salazar-González JJ, Rodríguez-Martín I (2009) A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. Comput Oper Res 36:1639–1645. https://doi.org/10.1016/j.cor.2008.03.008
28. Hoos HH, Stützle T (2004) Tutorial(AAAI-04): stochastic local search: foundations and applications. Elsevier, Burlington. http://www.sls-book.net/Slides/aaai-04-tutorial.pdf
29. Hoos HH, Stützle T (2005) Stochastic local search: foundations and applications. Elsevier, San Francisco
30. Igel C, Toussaint M (2003) On classes of functions for which no free lunch results hold. Inf Process Lett IPL 86:317–321
31. Jourdan L, Dhaenens C, Talbi EG (2006) Using datamining techniques to help metaheuristics: a short survey. In: Almeida F, Blesa Aguilera M, Blum C, Moreno Vega JM, Pérez Pérez M, Roli A, Sampels M (eds) Hybrid metaheuristics. Lecture notes in computer science, vol 4030. Springer, Berlin/Heidelberg, pp 57–69
32. Kariv O, Hakimi SL (1979) An algorithmic approach to network location problems. II: the p-medians. SIAM J Appl Math 37:513–538
33. Laguna M, Martí R (1999) GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J Comput 11:44–52
34. Li B, Chen F, Yin L (2000) Server replication and its placement for reliable multicast. In: Proceedings of the 9th international conference on computer communication and networks, Las Vegas, pp 396–401
35. Lin S, Kernighan B (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
36. Lodi A, Allemand K, Liebling TM (1999) An evolutionary heuristic for quadratic 01 programming. Eur J Oper Res 119:662–670
37. Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics, vol. 57. Springer, Boston, pp 320–353
38. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100
39. Plastino A, Barbalho H, Santos LFM, Fuchshuber R, Martins SL (2014) Adaptive and multi-mining versions of the DM-GRASP hybrid metaheuristic. J. Heuristics 20:39–74
40. Plastino A, Fuchshuber R, Martins SL, Freitas AA, Salhi S (2011) A hybrid data mining metaheuristic for the p-median problem. Stat Anal Data Min 4:313–335
41. Reinelt G (1991) TSPLIB: a traveling salesman problem library. ORSA J Comput 3:376–384. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

42. Resende MGC, Ribeiro CC (2005) GRASP with path-relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers. Operations research/computer science interfaces series, vol 32. Springer, New York, pp 29–63
43. Resende MGC, Werneck RF (2003) On the implementation of a swap-based local search procedure for the p-median problem. In: Proceedings of the thirteenth workshop of algorithm engineering and experiments (ALENEX03). SIAM, Baltimore, Maryland, USA, pp 119–127
44. Resende MGC, Werneck RF (2004) A hybrid heuristic for the p-median problem. J Heuristics 10:59–88
45. Ribeiro CC, Resende MGC (2012) Path-relinking intensification methods for stochastic local search algorithms. J Heuristics 18:193–214
46. Ribeiro CC, Rosseti I (2007) Efficient parallel cooperative implementations of GRASP heuristics. Parallel Comput 33:21–35
47. Ribeiro MH, Plastino A, Martins SL (2006) Hybridization of grasp metaheuristic with data mining techniques. J Math Model Algorithms 5:23–41
48. Santos HG, Ochi LS, Marinho EH, Drummond LM (2006) Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem. Neurocomputing 70: 70–77
49. Santos LF, Martins SL, Plastino A (2008) Applications of the DM-GRASP heuristic: a survey. Int Trans Oper Res 15:387–416
50. Senne ELF, Lorena LAN (2000) Langrangean/surrogate heuristics for p-median problems. In: Laguna M, González-Velarde JL (eds) Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research. Operations research/computer science interfaces series, vol 32. Kluwer, Boston, pp 115–130
51. Taillard ED (2003) Heuristic methods for large centroid clustering problems. J Heuristics 9: 51–74
52. Tan PN, Steinback M, Kumar V (2015) Introduction to data mining, 2nd edn. Addison-Wesley, Boston, MA, USA
53. Tansel BC, Francis RL, Lowe TJ (1983) A hybrid heuristic for the p-median problem. J Heuristics 29:482–511
54. Witten IH, Frank E (2011) Data mining: practical machine learning tools and techniques with java implementations, 3rd edn. Morgan Kaufmann, Burlington
55. Wolpert DH, Macread WG (1997) No free lunch theorems for optimization. IEEE Trans Evolut Comput 1:67–82
56. Zaki MJ, Wagner Meira J (2014) Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press, New York

Evolution Strategies

4

Michael Emmerich, Ofer M. Shir, and Hao Wang

Contents

Introduction . . . 90
Classical Evolution Strategies . . . 92
Derandomized Evolution Strategies . . . 98
  First Level of Derandomization . . . 99
  Second Level of Derandomization: CMA-ES . . . 101
Theoretical Results . . . 104
Nonstandard Evolution Strategies . . . 105
  ES for Nonstandard Search Spaces . . . 105
  Niching and Multi-population ES . . . 106
  Noise Handling and Robust Optimization with ES . . . 107
  Multi-criterion and Constraint-Handling ES . . . 108
  Mirrored Sampling . . . 109
Benchmarks and Empirical Study . . . 110
  Experimental Settings . . . 110
  Results . . . 112
Conclusions . . . 115
Cross-References . . . 116
References . . . 116

M. Emmerich (✉)
Leiden, Netherlands
e-mail: m.t.m.emmerich@liacs.leidenuniv.nl

O. M. Shir
Computer Science Department, Tel-Hai College, Upper Galilee, Kiryat Shmona, Israel

MIGAL Galilee Research Institute, Upper Galilee, Kiryat Shmona, Israel
e-mail: ofersh@telhai.ac.il; ofers@migal.org.il

H. Wang
Leiden Institute of Advanced Computer Science, Leiden University, Leiden, Netherlands
e-mail: h.wang@liacs.leidenuniv.nl

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_13

Abstract

Evolution strategies are classical variants of evolutionary algorithms which are frequently used to heuristically solve optimization problems, in particular in continuous domains. In this chapter, a description of classical and contemporary evolution strategies will be provided. The review includes remarks on the history of evolution strategies and how they relate to other evolutionary algorithms. Furthermore, developments of evolution strategies for nonstandard problems and search spaces will also be summarized, including multimodal, multi-criterion, and mixed-integer optimization. Finally, selected variants of evolution strategies are compared on a representative set of continuous benchmark functions, revealing strength and weaknesses of the different variants.

Keywords

Evolution strategy · derandomization · CMA-ES · benchmarking · theory

Introduction

Evolution strategies (ESs) are a class of metaheuristics for optimization by means of (computer) experiments. They belong to the broader class of evolutionary algorithms and like other heuristics from this class mimic adaptive processes in biological evolution. The search in ESs is characterized by the alternating application of variation and selection operators. Recombination and mutation operators are the variation operators and create new individuals. The selection operator selects individuals from a population based on their corresponding fitness value, which is obtained by computing the objective function value in evolution strategies. The selected individuals form the next population and the process is repeated. A distinguishing feature of evolution strategies as compared to most other evolutionary algorithms is self-adaptive mutation operators, which are capable of adapting the shape of the mutation distribution according to the local topology of the landscape and thereby help ESs achieve maximal progress rates.

Before discussing technical details of the evolution strategies, it will be worthwhile to give a brief outline of their history: The idea of mimicking evolution in order to optimize technical systems arose in Germany, where it led to the development of evolution strategies by Ingo Rechenberg [50] and Hans-Paul Schwefel [57], and in the USA, where it led to the development of genetic algorithms [21,34] and evolutionary programming [20]. In the mid-1990s, these strategies were unified under the common umbrella of *evolutionary algorithms* (EAs) [5]. While all these heuristics share the idea to mimic evolution in computational algorithms, researchers in genetic algorithms and evolution strategies emphasized on different aspects of algorithm design and problem domains.

Since their invention in the 1960s by Rechenberg and Schwefel at the Technical University of Berlin, evolution strategies have been used to optimize real-world systems, typically in engineering design. The first application of an evolution

strategy was the design of optimal shapes in engineering such as nozzles and wing shape using physical experiments. Often the evolutionary design procedure discovered high-performing structures with surprising shapes that have never been considered by engineers before [7]. Starting from sequential stochastic hill climbing strategies, soon evolution strategies advanced to more sophisticated problem-solvers, and their main application domain became the treatment of black-box continuous optimization problems on the basis of computer models.

One important development introduced adaptive step sizes or mutation distributions. Although there were some precursors to the idea of step-size adaptation in stochastic search algorithms [56], the development of flexible and efficient adaptation schemes for mutation distributions became a major point of attention in ES research. This feature distinguished them from genetic algorithms which worked commonly with constant mutation strengths.

Different variants of the adaptation of the mutation parameters were developed, and the three mainstream variants are the control of a single step size by means of the following approaches:

- The so-called $1/5$-th success rule: the rate of generating successful mutations is monitored, and the step size is controlled to achieve success rate $1/5$, which is the optimal on spherical function.
- The mutative self-adaptation: it most closely resembles natural evolution, where the step size also undergoes the recombination, mutation, and natural selection [14].
- The derandomized self-adaptation (Hansen, Ostermeier, and Gawelczyk [30]): it cumulates the standardized steps and compares the length of the cumulative vector to the one obtained under random selection.

Their efforts to find efficient ways to control the shape of the mutation distribution culminated in the covariance matrix adaptation evolution strategy (CMA-ES) [29]. Due to its invariance properties, it is seen today by many researchers as the state-of-the-art evolution strategy when it comes to practically solving ill-conditioned optimization problems (cf. [24]).

A parallel development was the introduction of population-based (or multi-membered) evolution strategies. Here, the idea is to create an evolutionary algorithm that performs, as Hans-Paul Schwefel called it, a *collective hill climbing* [58] (a collection of search points, where each point performs a simple hill climb search). To categorize different population models, the notation of (μ, λ)- and $(\mu + \lambda)$-schemes was introduced, in which μ denotes the number of individuals in the parent population and λ the number of individuals in the offspring population. These multi-membered strategies are able to exploit the positive effects of recombination (crossover) and are more reliable in global optimization settings and on noisy problems than the early single-membered variants. Moreover, population-based algorithms could be executed in parallel and later could more easily be extended to advanced evolution strategies for solving multi-objective and multimodal optimization tasks.

Nowadays, evolution strategies are mainly used for the simulation-based optimization, i.e., computerized models that require parametric optimization. Evolution strategies are suitable for the optimization of non-smooth functions because they do not require derivatives. In general, the algorithmic paradigms of ESs can be extended to general metric search spaces, and mainstream variants of ESs address continuous optimization problems, as opposed to genetic algorithms [21], which are more typically used on binary search spaces.

Contemporary evolution strategies have shown to be competitive to other derivative-free optimization algorithms, both in theoretical studies [13, 65] and by benchmark comparisons on a large corpus of empirical problems [59]. Furthermore, their practical utility is underpinned by a large number of successful applications in engineering and systems optimization [6].

This chapter will give a brief introduction to classical and contemporary evolution strategies with a focus on mainstream variants of evolution strategies. Firstly, classical $(\mu, \lambda)-$ and $(\mu + \lambda)$ evolution strategies will be described in section "Classical Evolution Strategies". Then, in section "Derandomized Evolution Strategies" derandomization, techniques will be discussed, which is the distinguishing feature in the CMA-ES. Section "Theoretical Results" addresses theoretical findings on the convergence and reliability of evolution strategies. An overview on new developments and nonstandard evolution strategies is provided in section "Nonstandard Evolution Strategies". Moreover, this section covers adaptations of ESs that make them more suitable for multimodal optimization. Section "Benchmarks and Empirical Study" discusses empirical benchmarks used in this field and includes a comparative study of contemporary evolution strategies. Section "Conclusions" summarizes the main characteristics of ESs and highlights future research directions.

Classical Evolution Strategies

The typical application field of evolution strategies is (continuous) unconstrained optimization where the optimization problem is given by:

$$\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) \tag{1}$$

In the context of ESs, the function f can be a black-box function and usually function f is assumed to be nonlinear and have a minimum. Maximization problems can be brought into the standard form of Eq. 1 by simply flipping the sign of f. Standard implementations also allow to restrict the domain of decision variables to interval domains and to introduce constraints. For the sake of brevity, the extensions of ESs for constraint handling will be widely omitted in the following discussion.

Evolution strategies can be viewed as stochastic processes on a population of individuals from the space of individuals $\mathbb{I}$. An individual in evolution strategies typically comprises the following information:

- a d-tuple of values for the decision variables $x_1, \ldots, x_d$, representing a candidate solution for the optimization problem (see Eq. 1),
- a tuple of strategy parameters. The strategy parameters can, for instance, be the standard deviations used to generate the perturbations of variables in the mutation (step sizes) or the components of a covariance matrix used in the mutation. Strategy parameters can be adapted during evolution.
- a fitness value. It is typically based on the objective function value, and it may be altered by a penalty for constraint violations.

In some variants of the evolution strategies, the so-called (μ, κ, λ)- evolution strategies, the individual's age is also maintained, that is, the number of generations that an individual has survived.

Another basic data structure of an evolution strategy is a population. A population is a multi-set of individuals, i. e., of elements of $\mathbb{I}$. One distinguishes between the parent populations P_t consisting of μ individuals and the offspring populations consisting of λ individuals.

The basic algorithm of a (μ, λ)- evolution strategy and a $(\mu + \lambda)$- evolution strategy is outlined in Algorithm 1. The algorithm starts with initializing a population P_0 of μ parent individuals, for instance, by uniform random sampling in the feasible intervals for the objective variables $\vec{x}$. Then, the fitness values of P_0 are determined, and the best solution found in P_0 is identified and stored in the variables $\vec{x}_0^{best}$, f_0^{best}.

Algorithm 1: Evolution Strategy

input: initial population P_0 with μ evaluated individuals
$t \leftarrow 0$ {Generation counter}
$(\vec{x}_1^{best}, f_1^{best}) = \texttt{Update}(P_0)$ {Updates best found solution to best solution found in P_0.}
while termination criterion not fulfilled **do**
  $t \leftarrow t + 1$
  $R_t \leftarrow \texttt{Recombine}(P_{t-1})$ {Creates λ offspring from P_t using recombination operator.}
  $M_t \leftarrow \texttt{Mutate}(R_t)$ {Creates a mutant for each individual in R_t.}
  $C_t \leftarrow \texttt{Evaluate}_f(M_t)$ {Evaluates fitness function values.}
  $(\vec{x}_{t+1}^{best}, f_{t+1}^{best}) = \texttt{Update}(C_t, \vec{x}_t^{best}, f_t^{best})$ {Updates best found solution.}
  $P_t \leftarrow \texttt{Select}(C_t, P_{t-1})$ {Selects the μ best individuals from $C_t \cup P_{t-1}$ in case of a $(\mu + \lambda)$- selection, and from C_t in case of a (μ, λ)-selection.}
end while
return $\vec{x}^{best}$, f^{best}

Then, the following *generational loop* is executed until a termination criterion is met. Common termination criteria are stagnation of the search process or the excess of a maximally allowed duration for searching.

The search process in the generational loop is governed by two (stochastic) variation operators, *recombination* and *mutation*, and a deterministic *selection*

operator. The recombination operator, namely, $\texttt{Recombine} : \mathbb{I}^{\mu} \times \Omega \rightarrow \mathbb{I}^{\lambda}$, generates from the μ individuals in P_t an offspring population of λ individuals which are then mutated by the mutation operator, $\texttt{Mutate} : \mathbb{I}^{\mu} \times \Omega \rightarrow \mathbb{I}^{\mu}$. The mutated individuals are evaluated, and, if necessary, the best found solution $\vec{x}_t^{best}$, f_t^{best} gets updated. Then the parent population of the next round is determined by selecting the μ best solutions from

- in case of a (μ, λ)-ES the λ offspring individuals
- in case of a $(\mu+\lambda)$-ES the μ parents of the current generation and the λ offspring individuals,
- in case of a (μ, κ, λ)-ES from the μ parents who have not exceeded an age of κ and the λ offspring individuals

Finally the generation counter is increased, and the loop is continued or terminated (if the termination condition is met). After completing the loop, the best attained solution constitutes the output.

It appears to be a "*chicken-and-egg*" dilemma whether it makes more sense to start the evolution strategy with the generation of a parent population (with μ individuals), as suggested in Algorithm 1, or to do so at some other stage of the evolution, i.e., starting with an offspring population. The chosen representation has the advantage that the process that generates the subsequent parent populations $P_0, P_1, P_2, \ldots$ can be viewed as a memoryless stochastic process or more precisely a Markov process:

$$P_{t+1} = \texttt{Select}\left(\texttt{Mutate}\left(\texttt{Recombine}\left(P_t\right)\right), P_t\right). \tag{2}$$

This means that given P_t for some $t \geq 1$, the information of P_{t-1} is irrelevant in order to determine P_{t+1}; alternatively, in the terminology of stochastic processes, the state of P_{t+1} is conditionally independent of the state of P_{t-1} given P_t. This so-called Markov property makes it easier to analyze the behavior of the evolution strategy. In addition, P_t can be viewed as a checkpoint of the algorithm, and if the process stops, e.g., because of a computer crash, the process may resume by starting the loop with the last saved state of P_t.

The main loop of the evolution strategy is inspired by the principles of evolutionary adaptation in nature that were discovered in parallel by the naturalists Alfred Russel Wallace (1823–1913) and Charles Darwin (1809–1882). In brief, a population of individuals adapt to their environment by (random) variation and selection. The reason for variability in the population was unknown to these researchers. Only much later in the so-called modern synthesis, it was linked to the mutation and recombination of genes. The ES presented in Algorithm 1 does however by far not provide a complete model for evolution in nature. In fact, important driving forces of natural evolutionary processes such as the development of temporally stable species and coevolution cannot be modeled with this basic evolution strategy. On the other hand, by mimicking only the variation and selection

process, one can already achieve a potent and robust optimization heuristic, and the theoretical analysis of evolution strategies can provide new insights in the dynamics of natural evolution.

There are many options to instantiate the operators of an evolution strategy, and in the literature, a certain terminology is used to refer to standard choices. Next, the most common instantiations of operators will be discussed by following the structure of Algorithm 1. The first step is the *initialization* of P_0, where the starting population is set by the user, since this allows to resume the evolution from a checkpoint. However, it is also very common to view the initialization as an integral part of the evolution strategy. Initialization procedures vary, while common choices are either constant initialization by generating μ copies of a starting (seed) point, or random initialization, i.e., to initialize the decision variables randomly within their bounds. The initialization of strategy parameters can have a significant impact on the transient behavior of an evolution strategy. In case of step-size parameters, it is often recommended to set these to 5% of the search space size.

A more complex operator is the recombination operator. In the nomenclature, the number of individuals that participate in the creation of a single vector is called ρ. The notation $(\mu/\rho, \lambda)$- ES and, respectively, $(\mu/\rho + \lambda)$- ES makes this number explicit. The ρ individuals that participate in the recombination are drawn by independent uniformly random choices from the population.

Given ρ individuals, there are two common strategies to create an offspring vector – *intermediate* recombination and *dominant* recombination. The vector to be determined is commonly the vector of decision variables $\vec{x}$, but it can also include the vector of strategy parameters:

- *Intermediate recombination* determines the offspring by averaging the components of the parent individual. It can be applied on the object parameters and on the strategy parameters. Given a ρ-tuple parent vectors $(\vec{q}^{(1)}, \ldots, \vec{q}^{(\rho)}) \in (\mathbb{R}^d)^\rho$, it computes the resulting vector $\vec{r}$ by means of $r_j = \frac{1}{\rho}(\sum_{i=1}^{\rho} q_j^{(i)})$ for $i = 1, \ldots, d$.
- *Discrete (or dominant) recombination* sets the i-th position of the offspring vector $\vec{r}$ randomly to one of values of the parents. By drawing d uniform random numbers $u_j,\ j = 1, \ldots, d$ from the set $\{1, \ldots, \rho\}$, the offspring individual is set to $r_j = q_{u_j}$.

The terminology, *intermediate* and *dominant*, is lent from the theory of inheritance of biological traits by the botanist Gregor Mendel (1822–1884).

The *mutation* operator is seen as a main driving force of the evolutionary progress in ESs. Mutation adds a small random perturbation to each component of $\vec{x}$. The scaling of this perturbation is based on the strategy variables. A common case is to use individual so-called step-size parameters $\sigma_1 \in \mathbb{R}^+, \ldots, \sigma_d \in \mathbb{R}^+$. The mutation then performs as indicated in the following equation:

$$x_i' \leftarrow x_i + \sigma_i \cdot \mathcal{N}(0, 1), \quad i = 1, \ldots, d \tag{3}$$

Here, Normal(0, 1) denotes the generation of a standard normal distributed random number. The resulting distributions of random numbers have a standard deviation of σ_i, why the σ-variables are also termed as standard deviations of the mutation.

A distinguishing feature of ESs is the *self-adaptation* of the mutation's parameters. In case of d step sizes, the mutative self-adaptation lets the step sizes themselves undergo an evolutionary process. In continuous optimization, the parameters of the multivariate Gaussian distribution can be adapted. Three levels of adaptation can be devised: Firstly, it is possible to control only a single standard deviation that is used for all decision variables (possibly with a constant scaling factor). This is called isotropic self-adaptation. Then, in the so-called individual step-size adaptation for each decision variable, a different standard deviation for the mutation is maintained and adapted. Finally, it is also possible to learn the full covariance matrix of the multivariate Gaussian distribution that is used in the mutation. The different levels of mutation distribution adaptation are indicated in Fig. 1. As a rule of thumb, it can be stated that the more mutation parameters are to be adapted, the longer it takes to reach an optimal convergence behavior for a given model.

There are different strategies for controlling or adapting the mutation parameters:

- Step-size control based on the success rate: It has been shown that for the $(1 + 1)$-ES and on two important benchmark functions – sum of squares and corridor model – among all isotropic Gaussian distributions, the optimal standard deviation is obtained at a step size that yields approximately a success probability of $1/5$. Because the success probability can be assessed during execution, this allows for an effective step-size control of the $(1 + 1)$-ES.

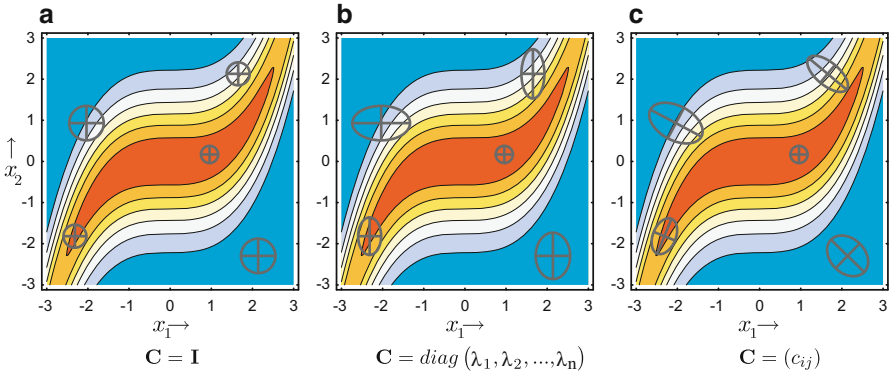


Fig. 1 Three levels of step-size control: isotropic (a), individual step sizes (b), and covariance matrix adaptation (c)

- Mutative step-size control: The idea is to make the parameters of the mutation distribution part of the individual and let it undergo an evolutionary process itself. Details of this strategy will be elaborated in this section.
- Derandomized step-size control: Here a more efficient adaptation of the mutation distribution is derived based on cumulative information from previous successful mutation steps. Derandomized self-adaptation uses arithmetic procedures that can no longer be considered as biomimetic. For the price of losing flexibility and simplicity, they gain efficiency in particular for unconstrained continuous optimization and allow practicable schemes for adapting a full covariance matrix of a mutation distribution. The history and details of derandomized evolution strategies will be elaborated on in the next section.

The classical self-adaptive mutation in evolution strategies is nowadays called mutative step-size control. For adapting the standard deviations of the mutation distribution, it augments the individual by a step-size vector and mutates the standard deviations of the mutation before it mutates the decision variables using these standard deviations:

$$N_{\text{global}} \leftarrow \mathcal{N}(0,1) \tag{4}$$

$$\sigma_i' \leftarrow \sigma_i \cdot \exp(\tau_{\text{local}}\mathcal{N}(0,1) + \tau_{\text{global}} N_{\text{global}}), \quad i = 1,\ldots,d \tag{5}$$

$$x_i' \leftarrow x_i + \sigma_i \cdot \mathcal{N}(0,1), \quad i = 1,\ldots,d \tag{6}$$

The parameters τ_{local} and τ_{global} are called local and global learning rates. The simplest form of the mutative step-size control exploits only a single step size σ for all the coordinates. In this case, Eqs. (5) and (6) simplify to

$$\sigma' \leftarrow \sigma \cdot \exp(\tau_{\text{global}} N_{\text{global}}), \quad i = 1,\ldots,d \tag{7}$$

$$x_i' \leftarrow x_i + \sigma' \cdot \mathcal{N}(0,1), \quad i = 1,\ldots,d \tag{8}$$

The utilization of mutative self-adaptation is valid when there are more offspring individuals than parent individuals. In this case, the step sizes with a greater value have a higher probability to be selected because they typically lead to better offspring individuals. Two commonly used default settings for parameters in the (μ, λ)-ES are:

- Small population size: $\mu = 1$, $\lambda = 7$, $\tau_{\text{global}} = 1.2$, single step size, $\rho = 1$.
- Large population size: $\mu = 15$, $\lambda = 100$, $\tau_{\text{local}} = 1.1$, $\tau_{\text{global}} = 1.2$, $\rho = \mu$, and intermediate recombination of decision variables and step-size parameters.

A larger population size is preferable if the optimization takes place on a rugged fitness function and a larger number of evaluations can be afforded.

Derandomized Evolution Strategies

Mutative step-size control tends to work well in the standard ES for the adaptation of a single global step size but shows poor performance when it comes to the individual step sizes or arbitrary normal mutations. Schwefel claimed that the adaptation of the strategy parameters in those cases is impossible within small populations [58] and suggested larger populations as a solution to the problem. Later on, Rudolph questioned the effectiveness of ES learning upon discarding past information and claimed that ES learning would benefit from introducing memory to the individuals [52]. Indeed, due to the crucial role that the mutation operator plays within ES, its mutative step-size control was investigated intensively. Especially, the disruptive effects to which mutative step-size control is subject were studied at several levels [29, 47] and are reviewed here:

- **Indirect selection**. By definition, the goal of the mutation operator is to apply a stochastic variation to the decision variables, which will increase the individual's selection probability. The selection of the *strategy parameters* setting is indirect, i.e., the vector of a successful mutation is not utilized to adapt the step-size parameters, but rather the parameters of the distribution that led to this mutation vector.
- **Realization of parameter variation**. Due to the sampling from a random distribution, the *realization* of the parameter variation does not necessarily reflect the nature of the strategy parameters. Thus, the difference de facto between good and bad strategy settings of strategy parameters is only reflected in the difference between their probabilities to be selected – which can be rather small. Essentially, this means that the selection process of the strategy parameters is *strongly disturbed*.
- The *strategy parameter change rate* is defined as the difference between strategy parameters of two successive generations. Hansen and Ostermeier [29] argue that the change rate is an important factor, as it gives an indication concerning the adaptation speed, and thus it has a direct influence on the performance of the algorithm. The principal claim is that *this change rate basically vanishes* in the standard ES. The *change rate* depends on the *mutation strength* to which the strategy parameters are subject. While aiming at attaining the maximal change rate, the latter is limited by an upper bound, due to the finite selection information that can be transferred between generations. Change rates that exceed the upper bound would lead to a stochastic behavior. Moreover, the mutation strength that obtains optimal change rate is typically smaller than the one that obtains good diversity among the mutants – a desired outcome of the mutation operator, often referred to as *selection difference*. Thus, the conflict between the objective of *optimal change rate* versus the objective of *optimal selection difference* cannot be resolved at the mutation strength level [47]. A possible solution to this conflict would be to *detach* the change rate from the mutation strength.

The so-called derandomized mutative step-size control aims to treat those disruptive effects, regardless of the search space dimensionality, population size, or any other characteristic parameters.

The concept of derandomized evolution strategies has been originally introduced by scholars at the Technical University of Berlin in the beginning of the 1990s. It was followed by the release of a new generation of successful ES variants by Hansen, Ostermeier, and Gawelczyk [28, 30, 46, 48].

The first versions of *derandomized ES algorithms* introduced a controlled global step size in order to monitor the individual step sizes by decreasing the stochastic effects of the probabilistic sampling. The selection disturbance was completely removed with later versions by omitting the adaptation of strategy parameters by means of probabilistic sampling. This was combined with individual information from the last generation (the successful mutations, i.e., of selected offspring) and then adjusted to *correlated mutations*. Later on, the concept of *adaptation by accumulated information* was introduced, aiming to use wisely the past information for the purpose of step-size adaptation. Rather than using the last generation's information alone, it was successfully generalized to a weighted average of the previous generations.

Note that the different derandomized ES variants strictly follow a $(1, \lambda)$ strategy, postponing the treatment of recombination or plus-strategies for later stages. Moreover, the different variants hold different numbers of strategy parameters to be adapted, and this is an important factor in the complexity of the optimization routine and in its learning rate. The different algorithms hold a number of strategy parameters scaling either linearly ($\mathcal{O}(d)$ parameters responsible for individual step-sizes) or quadratically ($\mathcal{O}(d^2)$ parameters responsible for arbitrary normal mutations) with the dimensionality d of the search space.

First Level of Derandomization

The so-called first level of derandomization targeted the following desired effects: (i) a degree of freedom with respect to the mutation strength of the strategy parameters, (ii) scalability of the *ratio* between the change rate and the mutation strength, and (iii) independence of population size with respect to the adaptation mechanism. The realization of the first level of derandomization can be reviewed through *three* particular derandomized ES variants:

DR1

The first derandomized attempt [46] coupled the successful mutations to the selection of decision parameters and learned the mutation step size as well as the scaling vector based upon the successful variation. The mutation step is formulated for the kth individual, $k = 1, \ldots, \lambda$:

$$\vec{x}^{(g+1)} = \vec{x}^{(g)} + \xi_k \delta^{(g)} \vec{\xi}^{k}_{\text{scal}} \vec{\delta}^{(g)}_{\text{scal}} \vec{z}_k \qquad \vec{z}_k \in \{-1, +1\}^d \tag{9}$$

Note that $\vec{z}_k$ is a random vector of ± 1, rather than a normally distributed random vector, while $\vec{\xi}^k_{\text{scal}} \sim \vec{\mathcal{N}}(0,1)^+$, i.e., distributed over the positive part of the normal distribution. The evaluation and selection are followed by the adaptation of the strategy parameters (subscripts *sel* refer to the selected individual):

$$\delta^{(g+1)} = \delta^{(g)} \cdot (\xi_{\text{sel}})^{\beta} \tag{10}$$

$$\vec{\delta}^{(g+1)}_{\text{scal}} = \vec{\delta}^{(g)}_{\text{scal}} \cdot \left(\vec{\xi}^{\text{sel}}_{\text{scal}} + b\right)^{\beta_{\text{scal}}} \tag{11}$$

DR2

The second derandomized ES variant [48] aimed to accumulate information about the correlation or anticorrelation of past mutation vectors in order to adapt the *global step size* as well as the *individual step sizes* – by introducing a *quasi-memory* vector. This accumulated information allowed omitting the stochastic element in the adaptation of the strategy parameters – updating them only by means of successful variations, rather than with random steps.

The mutation step for the kth individual, $k = 1, \ldots, \lambda$, reads

$$\vec{x}^{(g+1)} = \vec{x}^{(g)} + \delta^{(g)} \vec{\delta}^{(g)}_{\text{scal}} \vec{z}_k \qquad \vec{z}_k \sim \vec{\mathcal{N}}(0,1) \tag{12}$$

Introducing a quasi-memory vector $\vec{Z}$:

$$\vec{Z}^{(g)} = c\vec{z}_{\text{sel}} + (1-c)\,\vec{Z}^{(g-1)} \tag{13}$$

The adaptation of the strategy parameters according to the selected offspring:

$$\delta^{(g+1)} = \delta^{(g)} \cdot \left(exp\left(\frac{\|\vec{Z}^{(g)}\|}{\sqrt{d}\sqrt{\frac{c}{2-c}}} - 1 + \frac{1}{5d} \right) \right)^{\beta} \tag{14}$$

$$\vec{\delta}^{(g+1)}_{\text{scal}} = \vec{\delta}^{(g)}_{\text{scal}} \cdot \left(\frac{\left|\vec{Z}^{(g)}\right|}{\sqrt{\frac{c}{2-c}}} + b \right)^{\beta_{\text{scal}}}, \quad \left|\vec{Z}^{(g)}\right| = \left(|Z_1^{(g)}|, |Z_2^{(g)}|, \ldots, |Z_n^{(g)}| \right) \tag{15}$$

DR3

This third variant [30], usually referred to as the *Generation Set Adaptation* (GSA), considered the derandomization of arbitrary normal mutations for the first time, aiming to achieve invariance with respect to the scaling of variables and the rotation of the coordinate system. This naturally came with the cost of a quasi-memory matrix, $\mathbf{B} \in \mathbb{R}^{r \times d}$, setting the dimension of the strategy parameters space to $d^2 \leq r \leq 2d^2$. The adaptation of the global step size is *mutative* with stochastic variations, just like in the **DR1**.

The mutation step is formulated for the kth individual, $k = 1, \ldots, \lambda$:

$$\vec{x}^{(g+1)} = \vec{x}^{(g)} + \delta^{(g)} \xi_k \vec{y}_k \tag{16}$$

$$\vec{y}_k = c_m \mathbf{B}^{(g)} \cdot \vec{z}_k \qquad \vec{z}_k \sim \vec{\mathcal{N}}(0, 1) \tag{17}$$

The update of the *memory matrix* is formulated as

$$\begin{aligned} &\mathbf{B}^{(g)} = \left(\vec{b}_1^{(g)}, \ldots, \vec{b}_r^{(g)}\right) \\ &\vec{b}_1^{(g+1)} = (1-c) \cdot \vec{b}_1^{(g)} + c \cdot \left(c_u \xi_{\text{sel}} \vec{y}_{\text{sel}}\right), \quad \vec{b}_{i+1}^{(g+1)} = \vec{b}_i^{(g)} \end{aligned} \tag{18}$$

The step size is updated as follows:

$$\delta^{(g+1)} = \delta^{(g)} \left(\xi_{\text{sel}}\right)^{\beta} \tag{19}$$

Second Level of Derandomization: CMA-ES

Following a series of successful derandomized ES variants addressing the first level of derandomization, and a continuous effort at the Technical University of Berlin, the so-called covariance matrix adaptation (CMA) evolution strategy was released in 1996 [28], as a completely derandomized evolution strategy – the *fourth* generation of derandomized ES variants. The so-called second level of derandomization targeted the following effects: (i) The probability to regenerate the same mutation step is increased, (ii) the *change rate* of the strategy parameters is subject to explicit control, and (iii) strategy parameters are stationary when subject to random selection. The second level of derandomization was implemented by means of the CMA-ES.

The CMA-ES combines the robust mechanism of ES with powerful *statistical learning* principles, and thus it is sometimes subject to informal criticism for not being a genuine biomimetic evolution strategy. In short, it aims at satisfying the *maximum likelihood principle* by applying *principal component analysis* (PCA) [35] to the successful mutations, and it uses *cumulative global step-size adaptation*.

In the notation used here, the vector $\vec{m}$ represents the mean of the mutation distribution, but is also associated with the current solution-point, σ denotes the *global step size*, and the covariance matrix $\mathbf{C}$ determines the shape of the distribution ellipsoid:

$$\vec{x}^{\text{NEW}} \sim \mathcal{N}(\vec{m}, \sigma^2 \mathbf{C}) = \vec{m} + \sigma \cdot \mathcal{N}(\vec{0}, \mathbf{C}) = \vec{m} + \sigma \cdot \vec{z}$$

Two independent principles define the adaptation of the covariance matrix, $\mathbf{C}$, versus the adaptation of the global step size σ:

- The mean $\vec{m}$ and the covariance matrix $\mathbf{C}$ of the normal distribution are updated according to the *maximum likelihood principle*, such that good mutations are likely to appear again. $\vec{m}$ is updated such that

$$\mathcal{P}\left(\vec{x}_{\text{sel}}|\mathcal{N}\left(\vec{m}, \sigma^2\mathbf{C}\right)\right) \longrightarrow \max$$

  and $\mathbf{C}$ is updated such that

$$\mathcal{P}\left(\frac{\vec{x}_{\text{sel}} - \vec{m}_{\text{old}}}{\sigma}\middle|\mathcal{N}\left(\vec{0}, \mathbf{C}\right)\right) \longrightarrow \max$$

  considering the prior $\mathbf{C}$. This is implemented through the so-called covariance matrix adaptation (CMA) mechanism.
- σ is updated such that it is conjugate perpendicular to the consecutive steps of $\vec{m}$. This is implemented through the so-called cumulative step-size adaptation (CSA) mechanism.

Evolution Path

A straightforward way to update the covariance matrix would be to construct a $d \times d$ *matrix analogue* to the **DR2** mechanism (see Eq. 13), with the *outer-product* of the selected mutation vector $\vec{z}_{\text{sel}}$:

$$\mathbf{C} \longleftarrow (1 - c_{\text{cov}})\mathbf{C} + c_{\text{cov}}\vec{z}_{\text{sel}}\vec{z}_{\text{sel}}^T$$

However, to avoid discarding the sign information of $\vec{z}_{\text{sel}}$, the so-called evolution path is defined to accumulate the past information using an *exponentially weighted moving average*,

$$\vec{p}_c \propto \sum_{i=0}^{g}(1 - c_c)^{g-i}\vec{z}_{\text{sel}}^{(i)},$$

yielding the following update step for the covariance matrix:

$$\mathbf{C} \longleftarrow (1 - c_{\text{cov}})\mathbf{C} + c_{\text{cov}}\vec{p}_c\vec{p}_c^T$$

The Path Length Control

The covariance matrix update is not likely to simultaneously increase the variance in all directions, and thus a global step-size control is much needed to operate in parallel. The basic idea of the so-called path length control is to measure the length of the evolution path, which also constitutes the consecutive steps of $\vec{m}$, and adapt the step-size according to the following rationale: If the *evolution path* is longer than expected, the steps are likely parallel, and thus the step size should be increased; alternatively, if it is shorter than expected, the steps are probably antiparallel, and the step size should be decreased accordingly. That magnitude is defined as the expected

length of a normally distributed random vector. This evaluation is explicitly carried out by the *conjugate* evolution path:

$$\vec{p}_\sigma \propto \sum_{i=0}^{g} (1 - c_\sigma)^{g-i} \mathbf{C}^{(i)\,-\frac{1}{2}} \vec{z}_{\text{sel}}^{(i)}$$

where the *eigen-decomposition* of $\mathbf{C}$ is required in order to align all directions within the *rotated frame*. Then, the update of the step size depends on the comparison between $\|\vec{p}_\sigma\|$ and the expected length of a normally distributed random vector, $E\left[\|\mathcal{N}(0, \mathbf{I})\|\right]$:

$$\sigma \longleftarrow \sigma \cdot \exp\left(\frac{\|\vec{p}_\sigma\|}{E\left[\|\mathcal{N}(0, \mathbf{I})\|\right]} - 1\right)$$

The $(\mu_W; \lambda)$ Rank-μ CMA

The rank-μ covariance matrix adaptation [26] is an extension of the original update rule for larger population sizes. The idea is to use $\mu > 1$ vectors in order to update the covariance matrix $\mathbf{C}$ in each generation, based on *weighted intermediate recombination*. Let $\vec{x}_{i:\lambda}$ denote the ith ranked solution point, such that

$$f\left(\vec{x}_{1:\lambda}\right) \leq f\left(\vec{x}_{2:\lambda}\right) \leq \cdots \leq f\left(\vec{x}_{\lambda:\lambda}\right)$$

The updated *mean* is now defined as follows:

$$\vec{m} \leftarrow \sum_{i=1}^{\mu} w_i \vec{x}_{i:\lambda} = \vec{m} + \sigma \sum_{i=1}^{\mu} w_i \vec{z}_{i:\lambda} \equiv \langle \vec{x} \rangle_W$$

with a set of weights, $w_1 \geq w_2 \geq \cdots \geq w_\mu > 0$, $\sum_{i=1}^{\mu} w_i = 1$. The covariance matrix update is now formalized by means of rank-μ update, combined with the rank-one update:

$$\mathbf{C} \longleftarrow (1 - c_{\text{cov}})\mathbf{C} + \frac{c_{\text{cov}}}{\mu_{\text{cov}}} \vec{p}_c \vec{p}_c^{\,T} + c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i \vec{z}_{i:\lambda} \vec{z}_{i:\lambda}^{\,T}$$

The (μ_W, λ)-CMA-ES heuristic is summarized in Algorithm 2, with `initCMA()` referring to the parametric initialization procedure. It should be noted that a CMA variant, which resembles the DR2 and targets a vector of d individual step sizes (i.e., by means of a *diagonalized* covariance matrix), was released by the name of sep-CMA-ES [51]. Furthermore, the CMA-ES heuristic was simplified in the form of the so-called CMSA strategy [15] and was further improved for certain cases of global optimization [1].

Algorithm 2: (μ_W, λ)-CMA-ES

1: `initCMA()`
2: $g \leftarrow 0$
3: **repeat**
4: **for** $k = 1 \ldots \lambda$ **do**
5: $\vec{x}_k^{(g+1)} \leftarrow \langle \vec{x} \rangle_W^{(g)} + \sigma^{(g)} \cdot \vec{z}_k^{(g+1)}, \ \vec{z}_k^{(g+1)} \sim \mathcal{N}\left(\vec{0}, \mathbf{C}^{(g)}\right)$
6: $f_k^{(g+1)} \leftarrow \texttt{Evaluate}\left(\vec{x}_k^{(g+1)}\right)$
7: **end for**
8: $\langle \vec{x} \rangle_W^{(g+1)} \leftarrow \texttt{Select}\left(\vec{f}^{(g+1)}, \ \vec{x}_{1\ldots\mu:\lambda}^{(g+1)}\right)$
9: $\vec{p}_c^{(g+1)} \leftarrow \texttt{UpdatePath}\left(\vec{p}_c^{(g)}, \ \vec{z}_{1\ldots\mu:\lambda}^{(g+1)}\right)$
10: $\mathbf{C}^{(g+1)} \leftarrow \texttt{UpdateCov}\left(\mathbf{C}^{(g)}, \ \vec{p}_c^{(g+1)}, \ \vec{z}_{1\ldots\mu:\lambda}^{(g+1)}\right)$
11: $\mathbf{C}^{(g+1)} := \mathbf{R}^{(g+1)} \tilde{} ^{(g+1)} \left(\mathbf{R}^{(g+1)}\right)^T, \tilde{} ^{(g+1)} = \text{diag}\left(\lambda_{\max}^{(g+1)}, \ldots, \lambda_{\min}^{(g+1)}\right)$
12: $\vec{p}_\sigma^{(g+1)} \leftarrow \texttt{UpdateSPath}\left(\vec{p}_\sigma^{(g)}, \ \vec{z}_{1\ldots\mu:\lambda}^{(g+1)}\right)$
13: $\sigma^{(g+1)} \leftarrow \texttt{UpdateStep}\left(\sigma^{(g)}, \ \vec{p}_\sigma^{(g)}\right)$
14: $g \leftarrow g + 1$
15: **until** stopping criterion is met
16: **return** $\langle \vec{x} \rangle_W^{(g)}$

Theoretical Results

The theory of evolution strategies is traditionally focused on the questions of convergence dynamics.

Under relative mild conditions on the used mutation and recombination operators, complete global convergence in probability for $t \to \infty$ can be proven [54]. Basically, for continuous objective functions, it is sufficient to ascertain that for every ϵ-ball around the global minimizer the probability that the mutation operator samples a point in this region is positive, regardless of the starting point. This is given, for instance, by bounding the standard deviations of the mutation from below by a small positive value. This can be easily generalized to ES for discrete [53] or even mixed-integer optimization problems [43].

Of more practical relevance are results on the convergence dynamics, that is, the speed of convergence to the optimum and on local progress rates. Different approaches for analysis have been used, based on dynamical systems theory [13], stochastic process theory [54], and techniques from the asymptotic analysis of randomized algorithms [33]. It is a well-established result that the most self-adaptive ES variants, if parameterized correctly, achieve a linear convergence rate on convex quadratic problems, where the condition number of the matrix and the type of step-size adaptation determine the linear factor [14, 45]. The same can be found for problems with fitness proportional noise [2]. On the other hand, on sharp ridges

and plateaus and at the boundary of constraints, classical step-size adaptation tends to fail [38]. Also for problems with additive noise, the accuracy of the found results will be limited by the standard deviation of the noise [39]. The theory of ESs also revealed some insights regarding the manner in which different parameters are correlated with each other and devised guidelines concerning optimal parameter settings. Most prominently, the $1/5$-th success rule for step-size control in $(1 + 1)$-ES with isotropic mutation was developed based on theoretical studies on the sphere and the corridor model [50].

Also results on the effect of genetic drift and recombination in population-based evolution strategies are available: Beyer highlighted the so-called genetic repair effect that recombination has in ESs. When using recombination convergent behavior and optimal convergence, rates can be achieved with higher mutation step sizes. This increases the robustness in global optimization settings as it is more likely to escape from local optima. The genetic repair effect is stronger when intermediate recombination is used, as compared to discrete recombination. Dynamical systems analysis and Markov chain analysis on simple search landscapes revealed that it is rather impossible to simultaneously explore different local optima by means of a single population [11]. Even if the recombination operator is disabled, when different attractor basins share exactly the same geometry, the population tends to quickly concentrate on a single attractor only [55]. These findings gave incentive to the development of niching [60] and restart methods [4] that counteract this effect and prove to perform better on multimodal landscapes.

Nonstandard Evolution Strategies

The broad success of the family of evolution strategies provided the motivation to devise extended heuristics for treating problem instances that are beyond the canonical unconstrained single-objective, unimodal optimization formulation. Indeed, ES extensions to mixed-integer search spaces [8, 42], uncertainty handling [27], multimodal domains [60], and multi-objective Pareto optimization [31] were introduced in recent years. The goal of the current section is to provide an overview of those extensions.

ES for Nonstandard Search Spaces

A general framework for ES on nonstandard search spaces, termed metric-based evolutionary algorithms, was developed by Droste and Wiesmann [17]. They specified guidelines for instantiating of a mutation operator and recombination operators and exemplified the design method for the optimization of ordinary binary decision diagrams. Using similar guidelines, ESs for integer programming [53], mixed-integer programming [43], and graph-based optimization [18] were developed. Common guidelines on designing ESs for new types of solution spaces are:

1. Causal representation: Solutions should be represented in some metric search space such that relatively small changes with respect to the distance in the search space result on average in only relatively small changes of the objective function value.
2. Unimodal mutation distributions: Small mutations should occur more likely than large mutations.
3. Scalability of mutation: In order to implement self-adaptive mutation operators, it is essential that mutations can be scaled in terms of the average distance between parents and offspring.
4. Accessibility of points by mutation: By applying one or a chain of many mutations, it should be possible to reach every point in the search space, regardless of the starting point.
5. Unbiasedness of mutation: Mutations should not introduce a bias in the search. They should be symmetric and probability distributions with maximal entropy should be preferred.
6. Similarity to parents in recombination: The distance of an offspring to its parents should not exceed the distance of the parents to each other. Moreover, on average, the distance to all parents should be the same.

Following these design principles, one can expect generalized ESs to possess similar properties in comparison with standard ESs for continuous search spaces. However, a warning should be placed here regarding the functioning of step-size adaptation. In the theoretical derivation of optimal schemes of ES, often the fact is used that differentiable problems locally resemble quadratic or linear functions. This property is lost when it comes to discrete optimization, and therefore the generalization of such results requires some caution. On the other hand, it has been found that mutative self-adaptation of step sizes also works in nonstandard search spaces such as for the adaptation of mutation probabilities for binary vectors or for parameters of geometric distributions in mixed integer evolution strategies.

Niching and Multi-population ES

Given multimodal search landscapes with multiple basins of attraction that are of interest, targeting the simultaneous identification of several optima constitutes a challenge both at the theoretical and practical levels [44, 49, 60]. Within the domain of evolutionary computation, this challenge is typically treated by extending a given search heuristic into subpopulations of trial solutions that evolve in parallel to various solutions of the problem. This idea stems from the evolutionary concept of *organic speciation*, and the so-called *niching* techniques are the extension of EAs to speciation forming multiple subpopulations. The computational challenge in niching may be formulated as achieving an effective interplay between partitioning

the search space into niches occupied by stable subpopulations, by means of population diversity preservation, and exploiting the search in each niche by means of a highly efficient optimizer with local search capabilities [60]. A niching framework utilizing derandomized ES was introduced in [60], proposing the CMA-ES as a niching optimizer for the first time. The underpinning of that framework was the selection of a peak individual per subpopulation in each generation, followed by its sampling according to DES principles to produce the consecutive dispersion of search points. The *biological analogy* of this machinery is an *alpha male winning all the imposed competitions and dominating thereafter its ecological niche*, which then obtains all the sexual resources therein to generate its offspring.

A common utility for defining the landscape subdomain of each subpopulation is a so-called niche radius. A radius-based framework for niching, which employs derandomized ES heuristics, has been formulated and investigated [61] and has shown a broad success in tackling both synthetic and real-world multimodal optimization problems. In practice, this framework holds multiple derandomized ES populations, which conduct heuristic search in their radii-defined subdomains and independently update their mutation distributions and step sizes. Since the partitioning is enforced per each generation according to the niche radius parameter, and since no a priori knowledge is available on the global structure of the search landscape and the spatial distribution of its basins of attraction, an adaptive niche radius approach was devised to remedy this so-called niche radius presumption [62]. The main idea of ES niching with self-adaptive niche shape approaches is to exploit learned landscape information, as reflected by the evolving mutation distribution, to define the niches in a more accurate manner. Especially, a Mahalanobis CMA-ES niching heuristic was formulated, which carries out the distance calculations among the individuals based upon the Mahalanobis distance metric, by utilizing the evolving covariance matrices of the CMA mechanism. Such heuristic operation resulted in successful niching on landscapes with unevenly shaped optima, on which the fixed-radii approaches performed poorly [62].

On a related note, a multi-restart with increasing population size approach was developed with the CMA algorithm, namely, IPOP-CMA-ES [4]. This heuristic aims at attaining the global optimum, while possibly visiting local optima along the process and restarting the algorithm with a larger population size and a modified initial step size. It is thus not defined as a niching technique.

Noise Handling and Robust Optimization with ES

Robust optimization is concerned with identifying solvers that can also perform well when the input parameters and/or the objective function values are slightly perturbed on a systematic basis [10]. Most variants of ES are inherently suited to deal with noisy environments, and it was shown that a larger population size and the use of

recombination are beneficial in noisy settings [2,12]. Various techniques have shown to further improve performance on noisy objective functions. One technique is the so-called thresholding operator, which considers search points as improvement only when they introduce objective function improvements that exceed a certain threshold [9]. Moreover, it has been suggested to use the sample mean of multiple evaluations of the same individual (effective fitness) for evaluation. This increases the computation time, and therefore more efficient sampling schemes have been developed subsequently. Furthermore, a rank stability scheme was suggested to treat noise, exploiting the fact that ESs rather require a correct ranking rather than a correct objective function value [27].

Knowledge of second-order Hessian information at the optimum is desirable not only as a measure of system robustness to noise in the decision variables but also as a means for dimensionality reduction and for landscape characterization. Experimental optimization of quantum systems motivated the compilation of an automated method to efficiently retrieve the Hessian matrix about the global optimum *without derivative evaluations* from experimental measurements [63]. The study designed a heuristic to learn the Hessian matrix based upon the CMA-ES machinery, with necessary modification, by exploiting an inherent relation between the covariance matrix to the inverse Hessian matrix. It then corroborated this newly proposed technique, entitled forced optimal covariance adaptive learning (FOCAL), on noisy simulation-based optimization as well as on laboratory experimental quantum systems. The formal relation between the covariance matrix to the Hessian matrix is generally unknown, but has been a subject of active research. A recent study rigorously showed that accumulation of selected individuals carried the potential to reveal valuable information about the search landscape [64], e.g., as already practically utilized by derandomized ES variants. This theoretical study proved that a statistically constructed covariance matrix over selected decision vectors in the proximity of the optimum shared the same eigenvectors with the Hessian matrix about the optimum. It then provided an analytic approximation of this covariance matrix for a non-elitist multi-child $(1, \lambda)$ strategy, holding for a large population size λ.

For a comprehensive overview of contemporary noise handling and robust optimization in ESs, the reader is referred to the PhD dissertation of Kruisselbrink, which was also complemented by an empirical study of the most common variants. Also, theoretical limits in the precision of multi-evaluation schemes in the presence of additive noise are derived therein [39].

Multi-criterion and Constraint-Handling ES

In practical settings, the scenario of unconstrained optimization is not very common. Rather problems with multiple constraint functions and conflicting objective functions need to be solved.

In the optimization with constraints, it is mandatory to use alternative schemes for step-size adaptation for reasons that are explained in detail by Kramer and

Schwefel [38]. They also suggest alternative schemes that can better deal with the constraints.

Adaptations of the ESs are also required for optimization with multiple objective functions. In this case, it is common to search for a set of non-dominated solutions, the so-called Pareto set. A first proposal on using evolution strategies to approximate a Pareto front was made by Kursawe [40], a long time before the nowadays flourishing research field of evolutionary multi-criterion optimization was established. Today, three variants of the evolution strategy are used for optimization with multiple objectives:

- Pareto archived evolution strategy [37]: This classical multi-criterion optimization strategy uses an archive to maintain non-dominated points. The archive is updated based on the non-dominance and density of points.
- Predator prey evolution strategy [22, 41]: In this biomimetic strategy, individuals are distributed on a grid and a population of predator individuals is performing a random walk on the grid and triggers local selection. The predators select their prey based on different objective functions or combination strategies.
- Multi-objective CMA-ES [31]: This strategy seeks to improve contributions of individuals to the hypervolume indicator, which measures the size of the Pareto dominated space. Consequently, this strategy is well adapted to locate precise and regular representations of Pareto fronts for objective functions with complex shapes and correlated input variables. Another self-adaptation method, using local tournaments on hypervolume contributions, was suggested in [36] but so far received little attention.

Mirrored Sampling

The mirrored sampling technique is a derandomized mutation method. It was firstly introduced in [16] for non-elitist $(1, \lambda)$-ES and then extended to the (μ, λ)-ES [3]. The idea of mirrored sampling is to generate part (normally half) of the offspring population in a derandomized way. More specifically, a single mutation vector $\mathbf{z}$ is utilized to generate two offspring (rather than one in the standard ES) – one by adding $\mathbf{z}$ to the parent $\mathbf{x}$: $\mathbf{x} + \mathbf{z}$ and another by subtracting $\mathbf{z}$ from $\mathbf{x}$: $\mathbf{x} - \mathbf{z}$. The two offspring generated are *symmetric* or *mirrored* to the parental point. Mirrored sampling helps accelerating the convergence rate of evolution strategies, which is theoretically proven in [16].

When applied in (μ, λ)-CMA-ES with cumulative step-size adaptation, the mirrored sampling leads to a reduction of recombined mutation variance. Consequently, the step size is more than desirably reduced, and a premature convergence would occur. In order to solve this, the concept of pairwise selection is introduced [3], in which only the better offspring among the mirrored pair is allowed to possibly contribute to the weighted recombination. Then, it is assured that recombination will not use both elements of a mirrored pair at the same time.

Benchmarks and Empirical Study

Besides a detailed description of evolution strategies and their theoretical aspects, it would be intuitive and helpful to look at the empirical ability of ESs in solving the black-box problems. In the ES benchmarking, two difficulties arise. On one hand, it is hard to design a set of test functions that captures the problem characteristics encountered in the real-world applications. On the other hand, summarizing ES ability over a set of test functions would be not straightforward due to the fact that performance of evolution strategies largely varies on problems with different characteristics (e.g., separability). The black-box optimization benchmark [25] (BBOB) is devised to tackle these difficulties. The noiseless BBOB encompasses 24 noise-free real-parameter single-objective functions that are either separable, ill-conditioned, or multimodal. All the test functions are defined over $\mathbb{R}^d$, while the global optima for all the test functions are initialized in $[-5, 5]^d$ [19]. In addition, BBOB also introduces a proper measure to represent the performance of ESs for global optimization – the empirical cumulative distribution function (ECDF). ECDFs can be summarized over multiple test functions and represented graphically, increasing the accessibility of the benchmark results.

As opposed to the earlier work, in this empirical study, evolution strategies with different step-size adaptation strategies are considered, including the classical $1/5$-th success rule and mutative self-adaptation. Moreover the benchmark also covers a broad range of classical and contemporary strategies with derandomized self-adaptation. The evolution strategies tested are listed in the following:

- $(1 + 1)$-ES: one plus one elitist evolution strategy with $1/5$ success rule.
- $(15, 100)$-MSC-ES: mutative self-adaptation of individual step sizes.
- $(1, 7)$-MSC-ES: mutative self-adaptation of individual step sizes.
- DR2-ES: the derandomized evolution strategy using accumulated success mutation vector for step-size adaptation.
- $(\mu/\mu_{\mathrm{w}}, \lambda)$-CMA-ES: covariance matrix adaptation evolution strategy with weighted intermediate recombination.
- $(\mu/\mu_{\mathrm{w}}, \lambda_{\mathrm{m}})$-CMA-ES: CMA-ES with mirrored sampling and pairwise selection.
- IPOP-CMA-ES: a restart CMA-ES with increasing population size.
- $(1, \lambda)$-DR2-Niching: the niching approach based on the second derandomized ES variants.
- $(1, \lambda)$-CMA-Niching: CMA-ES niching with fixed niche radius.
- $(1 + \lambda)$-CMA-Niching: the elitist version.
- $(1, \lambda)$-Mahalanobis-CMA-Niching: niche shape adaptation using Mahalanobis distance.

Experimental Settings

The BBOB parameter settings of the experiment are the same for all the tested ES variants. The initial global step size σ is set to 1. The maximum number of

function evaluations is set to $10^4 \times d$. The initial solution vector (initial parent) is a uniformly distributed random vector restricted to the hyper-box $[-4, 4]^d$. The algorithms are tested for problems of different number of input variables d. These are $d \in \{2, 3, 5, 10, 20\}$.

The parameter settings for CMA-ES variants follow the suggested values in [23]; the reader is referred to it for details. For all the niching ES variants, the setting $\lambda = 10$ is used throughout all the test. The fixed niche radius calculation can be found in [60].

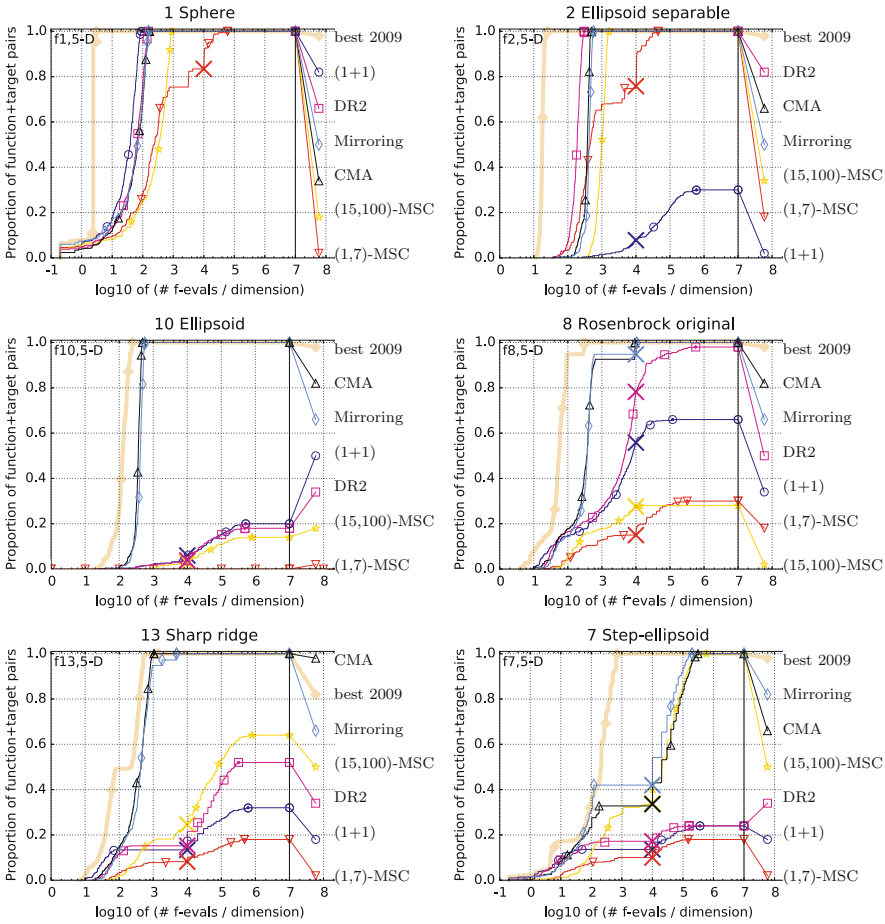


Fig. 2 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(1+1), ▽:(1,7)-MSC, ⋆:(15,100)-MSC, □:DR2, △:CMA, ◇:Mirroring

Results

BBOB automatically records the history of the fitness values found by the tested algorithm. The time is measured in numbers of function evaluations. The benchmark provides a postprocessing procedure to statistically estimate the empirical cumulative distribution function (ECDF) of running length (function evaluations) to reach the global optimum from the data. The ECDF of running length describes the distribution of necessary function evaluations a specific optimization algorithm follows in order to reach the global optimum in an experiment. An ECDF curve that inclines to distribute running length over smaller values indicates good performance of its corresponding algorithm. Thus, ECDFs characterize the performance of optimization algorithms and are used to present the benchmark results.

Instead of generating ECDFs for all 24 test functions in BBOB, several representative functions in BBOB are selected, which are listed as follows:

1. f_1 Sphere function.
2. f_2 Ellipsoidal function.
3. f_{10} Rotated ellipsoidal function.
4. f_8 Rosenbrock function.
5. f_{13} Sharp ridge function.
6. f_7 Step ellipsoidal function.
7. $f_{15} - f_{19}$ Multimodal function having weakly global structure.
8. $f_{20} - f_{24}$ Multimodal function having adequate global structure.

By aggregating the selected functions in function groups, it is possible to illustrate which ES variants perform better given a specific property (e.g., separable, isotropic, multimodal) by identifying functions with distinct properties. Rather than

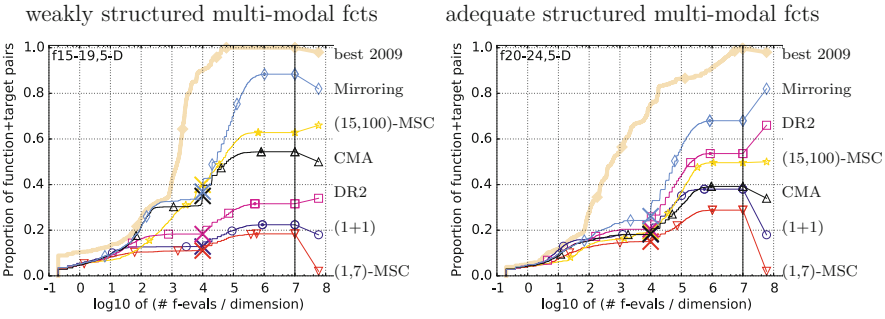


Fig. 3 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for two groups of multimodal functions in 5-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(1+1), ▽:(1,7)-MSC, ⋆:(15,100)-MSC, □:DR2, △:CMA, ◇:Mirroring

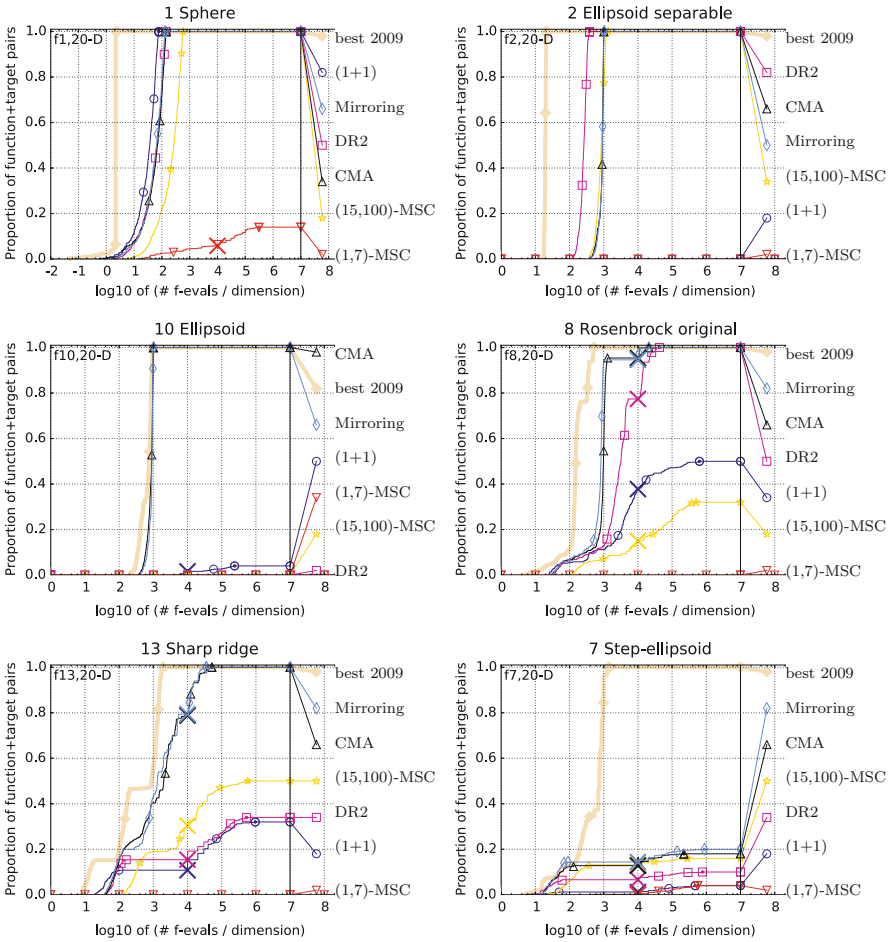


Fig. 4 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(1+1), ▽:(1,7)-MSC, ★:(15,100)-MSC, □:DR2, △:CMA, ◇:Mirroring

making a thorough competition, we aim for providing insights into the algorithms' strengths and weaknesses.

The results are presented in two parts. In the first part, all the ESs except IPOP-CMA-ES and the niching ES variants are compared on the aforementioned test functions and are excluded from the comparison. The purpose is to compare all the ESs which are inclining to local search. The results are depicted in Figs. 2, 3, 4, and 5. In 5D, the comparisons on the first 6 functions show that the standard

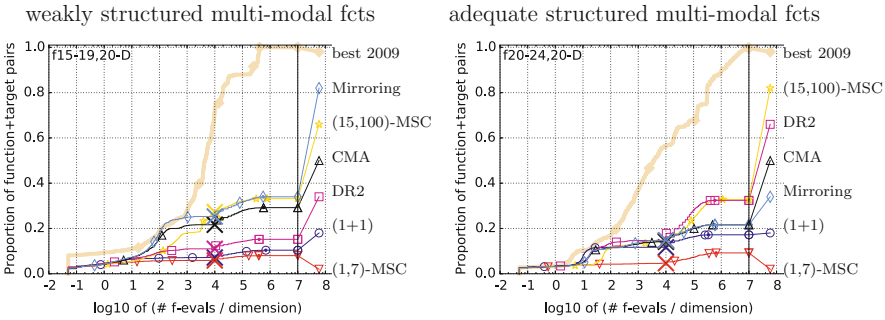


Fig. 5 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for two groups of multimodal functions in 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(1+1), ▽:(1,7)-MSC, ★:(15,100)-MSC, □:DR2, △:CMA, ◇:Mirroring

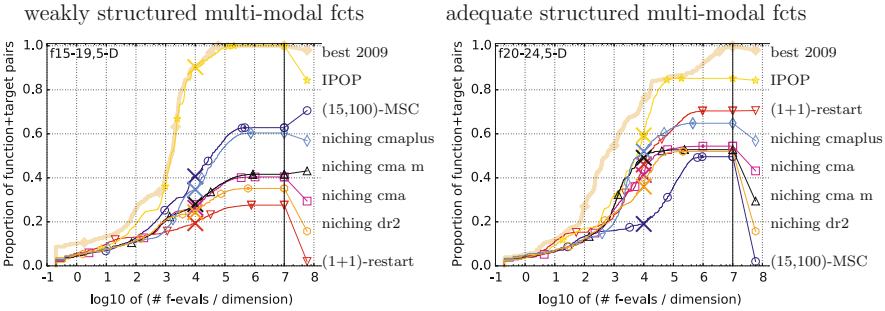


Fig. 6 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for two groups of multimodal functions in 5-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(15,100)-MSC, ▽:(1+1)-restart, ★:IPOP, □:niching_cma, △:niching_cma_ma, ◇:niching_cmaplus, ○:niching_dr2

CMA-ES and its mirrored sampling variant outperform the others in most functions. In general, the mutative self-adaptation ESs perform worse than the derandomized ES variants. The reason is that the MSC-ES normally exploits a much larger population size in order to adapt the covariance and thus consumes much more function evaluations. On the simple sphere function, the winner is (1 + 1)-ES, as expected from theory. In addition, the DR2-ES is equally good as CMA-ES. On the separable ellipsoid function, DR2-ES even outperforms CMA-ES because it efficiently adapts the uncorrelated mutations. On the non-separable functions (ellipsoid, Rosenbrock, sharp ridge and step-ellipsoid), CMA-ES and mirrored sampling significantly outperform the other ES variants. This is because CMA-ES is capable of adapting arbitrary mutations by means of the covariance matrix, while the other ES variants exploit either isotropic or axis-parallel mutation distributions. In 20D, the comparison roughly shows the same results as in 5D. Note that on

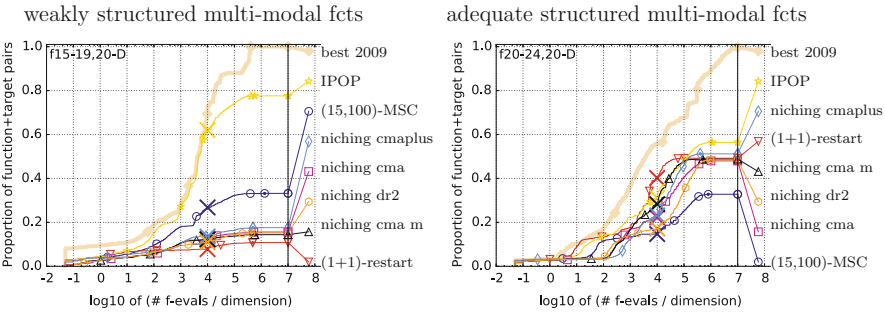


Fig. 7 Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 50 targets in $10^{[-8..2]}$ for two groups of multimodal functions in 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. Legend: ○:(15,100)-MSC, ▽:(1+1)-restart, ⋆:IPOP, □:niching_cma, △:niching_cma_ma, ◇:niching_cmaplus, ⬡:niching_dr2

separable ellipsoid and step-ellipsoid function, $(15, 100)$-MSC-ES could catch the convergence speed of CMA-ES variants.

In the second part, all the niching ES variants, IPOP-CMA-ES, restart $(1+1)$-ES and (15,100)-MSC-ES are compared on the multimodal functions. These algorithms are grouped for this comparison because they are better equipped for global search. The results are depicted in Figs. 6 (5D) and 7 (20D). On the weakly structured multimodal functions ($f_{15} - f_{19}$), IPOP-CMA-ES outperforms all the niching ES variants, $(1 + 1)$-ES with restart and $(15, 100)$-MSC-ES. $(1 + \lambda)$-CMA-Niching shows the best performance among all the niching ES variants tested. Surprisingly, (μ, λ)-MSC-ES performs well even compared to niching ES variants, which may be a consequence of its large population size. On the weakly structured multimodal functions, $(1 + \lambda)$-CMA-Niching could catch up with the performance of IPOP-CMA-ES both in 5D and 20D. (μ, λ)-MSC-ES performs quite poorly in this case. In addition, although it is a simple strategy, $(1+1)$-restart shows good results compared to IPOP-CMA-ES and niching ES. Evidently, the niching ES variants spend many function evaluations for maintaining the local optima and thus exhibits altogether poorer performance in terms of global convergence speed when compared to, e.g., IPOP-CMA-ES, which targets the accurate approximation of a single optimum and exploits much of its resources to achieve this.

Conclusions

It has been shown in this chapter that evolution strategies are a versatile class of stochastic search heuristics for optimization. There exists a rich body of theoretical results on ESs, including global convergence conditions, results showing linear convergence rates on high dimensional functions, and findings on the stability of subpopulations and the impact of recombination on global convergence reliability. Moreover, ESs are rank-based (order-invariant) and invariant to changes of the

coordinate system. The self-adaptation of the stochastic distribution is an important feature, too, as it frees the user from the burden of choosing the right parameters for mutation and it also makes highly precise approximation of optima possible.

This chapter highlighted mainstream variants of ESs for continuous optimization, including the $(1+1)$-ES with $1/5$-th success rule, the $(\mu, \lambda)-$ES with mutative step-size adaptation, and common variants of ES with different levels of derandomized step-size adaptation, namely, DR1, DR2, DR3, and CMA-ES. Moreover, common concepts of ESs for multimodal optimization were discussed.

All these strategies have been compared on different categories of functions. The empirical studies confirmed the superiority of covariance matrix adaptation techniques on ill-conditioned problems with correlated variables. However, if these problems do not govern the search difficulty, other evolution strategies can be highly competitive as well. Moreover, it was confirmed that multimodal optimization requires special adaptations to evolution strategies in order to achieve maximal performance.

Our literature review has shown that the algorithmic techniques developed for ES are not only fruitful in the domain of continuous optimization but can be applied to other problem classes as well. Here, the key is defining a metric representation of the search space and following a set of guidelines for the design of mutation and recombination operators, which were reviewed here in a rather informal manner.

Some prevalent topics for future research will be the integration of multiple criteria and constraints, although some first promising results are already available in this direction. Moreover, for nonstandard ES, the theoretical analysis needs to be advanced, in particular the study of convergence dynamics when the available time is limited. Finally, looking back to the original biological inspiration of evolution strategies, one might conjecture that nature has still many "tricks" in store that when well understood could lead to a further enhancement of ES-like search strategies. In this context, it will be interesting to follow recent trends in biological evolution theories [32], showing that a much broader set of mechanisms seem to govern organic evolution than those captured in the modern synthesis.

Cross-References

- ▸ Biased Random-Key Genetic Progamming
- ▸ Evolutionary Algorithms
- ▸ Genetic Algorithms
- ▸ Random-Key Genetic Algorithms

References

1. Akimoto Y, Sakuma J, Ono I, Kobayashi S (2008) Functionally specialized CMA-ES: a modification of CMA-ES based on the specialization of the functions of covariance matrix adaptation and step size adaptation. In: GECCO'08: proceedings of the 10th annual conference on genetic and evolutionary computation, New York. ACM, pp 479–486

2. Arnold DV (2002) Noisy optimization with evolution strategies, vol 8. Springer, Boston
3. Auger A, Brockhoff D, Hansen N (2011) Mirrored sampling in evolution strategies with weighted recombination. In: Proceedings of the 13th annual conference on genetic and evolutionary computation, GECCO'11, New York. ACM, pp 861–868
4. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of the 2005 congress on evolutionary computation CEC-2005, Piscataway. IEEE Press, pp 1769–1776
5. Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, Oxford
6. Bäck T, Emmerich M, Shir OM (2008) Evolutionary algorithms for real world applications [application notes]. IEEE Comput Intell Mag 3(1):64–67
7. Bäck T, Hammel U, Schwefel H-P (1997) Evolutionary computation: comments on the history and current state. IEEE Trans Evol Comput 1(1):3–17
8. Bäck T, Schütz M (1995) Evolution strategies for mixed integer optimization of optical multilayer systems. In: Evolutionary programming IV – proceeding of fourth annual conference on evolutionary programming. MIT Press, pp 33–51
9. Bartz-Beielstein T (2005) Evolution strategies and threshold selection. In: Hybrid metaheuristics. Springer, pp 104–115
10. Ben-Tal A, El Ghaoui L, Nemirovski A (2009) Robust optimization. Princeton series in applied mathematics. Princeton University Press, Princeton
11. Beyer H-G (1999) On the dynamics of EAs without selection. Found Genet Algorithms 5:5–26
12. Beyer H-G (2000) Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. Comput Methods Appl Mech Eng 186(2):239–267
13. Beyer H-G (2001) The Theory of Evolution Strategies. Springer, Berlin
14. Beyer H-G, Schwefel H-P (2002) Evolution strategies–a comprehensive introduction. Nat Comput 1(1):3–52
15. Beyer H-G, Sendhoff B (2008) Covariance matrix adaptation revisited – the CMSA evolution strategy. In: Parallel problem solving from nature – PPSN X. Lecture notes in computer science, vol 5199. Springer, Berlin, pp 123–132
16. Brockhoff D, Auger A, Hansen N, Arnold DV, Hohm T (2010) Mirrored sampling and sequential selection for evolution strategies. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G (eds) Proceedings of the 11th international conference on Parallel problem solving from nature: part I, PPSN'10. Springer, Berlin, pp 11–21
17. Droste S, Wiesmann D (2003) On the design of problem-specific evolutionary algorithms. In: Advances in evolutionary computing. Springer, Berlin, pp 153–173
18. Emmerich M, Grötzner M, Schütz M (2001) Design of graph-based evolutionary algorithms: a case study for chemical process networks. Evol Comput 9(3):329–354
19. Finck S, Hansen N, Ros R, Auger A (2010) Real-parameter black-box optimization benchmarking 2010: presentation of the noisy functions. Technical report 2009/21, Research Center PPE
20. Fogel LJ (1999) Intelligence through simulated evolution: forty years of evolutionary programming. Wiley, New York
21. Goldberg D (1989) Genetic algorithms in search, optimization, and machine learning. Addison Wesley, Reading
22. Grimme C, Schmitt K (2006) Inside a predator-prey model for multi-objective optimization: a second study. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. ACM, pp 707–714
23. Hansen N (2016) The CMA evolution strategy: a tutorial. arXiv preprint, arXiv:1604.00772. 4 Apr 2016
24. Hansen N, Arnold DV, Auger A (2015) Evolution strategies. Springer, Berlin/Heidelberg, pp 871–898
25. Hansen N, Auger A, Finck S, Ros R (2010) Real-parameter black-box optimization benchmarking 2010: experimental setup. Technical report RR-7215, INRIA
26. Hansen N, Kern S (1998) Evaluating the CMA evolution strategy on multimodal test functions. In: Parallel problem solving from nature – PPSN V. Lecture notes in computer science, vol 1498. Springer, Amsterdam, pp 282–291

27. Hansen N, Niederberger S, Guzzella L, Koumoutsakos P (2009) A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Trans Evol Comput 13(1):180–197
28. Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of the 1996 IEEE international conference on evolutionary computation, Piscataway. IEEE, pp 312–317
29. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
30. Hansen N, Ostermeier A, Gawelczyk A (1995) On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation. In: Proceedings of the sixth international conference on genetic algorithms (ICGA6), San Francisco. Morgan Kaufmann, pp 57–64
31. Igel C, Hansen N, Roth S (2007) Covariance matrix adaptation for multi-objective optimization. Evol Comput 15(1):1–28
32. Jablonka E, Lamb MJ (2005) Evolution in four dimensions. MIT Press, Cumberland
33. Jägersküpper J (2007) Algorithmic analysis of a basic evolutionary algorithm for continuous optimization. Theor Comput Sci 379(3):329–347
34. John H (1992) Holland, adaptation in natural and artificial systems. MIT Press, Cambridge
35. Jolliffe I (2002) Principal component analysis2nd edn. Springer, New York
36. Klinkenberg J-W, Emmerich MT, Deutz AH, Shir OM, Bäck T (2010) A reduced-cost SMS-EMOA using kriging, self-adaptation, and parallelization. In: Multiple criteria decision making for sustainable energy and transportation systems. Springer, Berlin/Heidelberg, pp 301–311
37. Knowles JD, Corne DW (2000) Approximating the nondominated front using the Pareto archived evolution strategy. Evol Comput 8(2):149–172
38. Kramer O, Schwefel H-P (2006) On three new approaches to handle constraints within evolution strategies. Nat Comput 5(4):363–385
39. Kruisselbrink J (2012) Evolution strategies for robust optimization. PhD thesis, Leiden University, Leiden
40. Kursawe F (1991) A variant of evolution strategies for vector optimization. In: Parallel problem solving from nature. Springer, Berlin/Heidelberg, pp 193–197
41. Laumanns M, Rudolph G, Schwefel H-P (1998) A spatial predator-prey approach to multi-objective optimization: a preliminary study. In: Parallel problem solving from nature – PPSN V. Springer, Berlin/Heidelberg, pp 241–249
42. Li R (2009) Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis. PhD thesis, Leiden University, Leiden
43. Li R, Emmerich MT, Eggermont J, Bäck T, Schütz M, Dijkstra J, Reiber JH (2013) Mixed integer evolution strategies for parameter optimization. Evol Comput 21(1): 29–64
44. Mahfoud S (1995) Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana Champaign
45. Meyer-Nieberg S, Beyer H-G (2007) Self-adaptation in evolutionary algorithms. In: Parameter setting in evolutionary algorithms. Springer, Berlin, pp 47–75
46. Ostermeier A, Gawelczyk A, Hansen N (1993) A derandomized approach to self adaptation of evolution strategies. Technical report TR-93-003, TU Berlin
47. Ostermeier A, Gawelczyk A, Hansen N (1994) A derandomized approach to self adaptation of evolution strategies. Evol Comput 2(4):369–380
48. Ostermeier A, Gawelczyk A, Hansen N (1994) Step-size adaptation based on non-local use of selection information. In: Parallel problem solving from nature – PPSN III. Lecture notes in computer science, vol 866. Springer, Berlin/Heidelberg, pp 189–198
49. Preuss M (2015) Multimodal optimization by means of evolutionary algorithms. Natural computing series. Springer International Publishing, Cham
50. Rechenberg I (1973) Evolutionsstrategie: optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Friedrich Frommann Verlag, Stuttgart-Bad Cannstatt

51. Ros R, Hansen N (2008) A simple modification in CMA-ES achieving linear time and space complexity. In: Parallel problem solving from nature – PPSN X. Lecture notes in computer science, vol 5199. Springer, Berlin/Heidelberg, pp 296–305
52. Rudolph G (1992) On correlated mutations in evolution strategies. In: Parallel problem solving from nature – PPSN II. Elsevier, Amsterdam, pp 105–114
53. Rudolph G (1994) An evolutionary algorithm for integer programming. In: Parallel problem solving from nature – PPSN III. Springer, Berlin/Heidelberg, pp 139–148
54. Rudolph G (1997) Convergence properties of evolutionary algorithms. Kovac, Hamburg
55. Schönemann L, Emmerich M, Preuß M (2004) On the extinction of evolutionary algorithm subpopulations on multimodal landscapes. Informatica (Slovenia) 28(4):345–351
56. Schumer M, Steiglitz K (1968) Adaptive step size random search. IEEE Trans Autom Control 13(3):270–276
57. Schwefel H-P (1965) Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Technische Universität, Berlin
58. Schwefel H-P (1987) Collective phenomena in evolutionary systems. In: Checkland P, Kiss I (eds) Problems of constancy and change – the complementarity of systems approaches to complexity, proceeding of 31st annual meeting, Budapest, vol 2. International Society for General System Research, pp 1025–1033
59. Schwefel H-PP (1993) Evolution and optimum seeking: the sixth generation. Wiley, New York
60. Shir OM (2008) Niching in derandomized evolution strategies and its applications in quantum control. PhD thesis, Leiden University, Leiden
61. Shir OM, Bäck T (2009) Niching with derandomized evolution strategies in artificial and real-world landscapes. Nat Comput Int J 8(1):171–196
62. Shir OM, Emmerich M, Bäck T (2010) Adaptive niche-radii and niche-shapes approaches for niching with the CMA-ES. Evol Comput 18(1):97–126
63. Shir OM, Roslund J, Whitley D, Rabitz H (2014) Efficient retrieval of landscape hessian: forced optimal covariance adaptive learning. Phys Rev E 89(6):063306
64. Shir OM, Yehudayoff A (2017) On the statistical learning ability of evolution strategies. In: Proceedings of the 14th ACM/SIGEVO conference on foundations of genetic algorithms, FOGA-2017. ACM Press, New York, pp 127–138
65. Teytaud O (2011) Lower bounds for evolution strategies. Theory Random Search Heuristics 1:327–354

Matheuristics

5

Martina Fischetti and Matteo Fischetti

Contents

Introduction . . . 122
General-Purpose MIP-Based Heuristics . . . 123
Local Branching . . . 124
Relaxation-Induced Neighborhood Search . . . 125
Polishing a Feasible Solution . . . 125
Proximity Search . . . 126
Application 1: Wind Farm Layout Optimization . . . 126
Choice of the MIP Model . . . 127
Choice of Ad Hoc Heuristics . . . 129
The Overall Matheuristic . . . 130
Computational Results . . . 131
Application 2: Prepack Optimization . . . 134
Mathematical Model . . . 135
Matheuristics . . . 136
Computational Experiments . . . 139
Application 3: Vehicle Routing . . . 142
The ASSIGN Neighborhood for TSP . . . 142
From TSP to DCVRP . . . 143
The Overall Matheuristic . . . 146
Conclusion . . . 150
Cross-References . . . 151
References . . . 151

M. Fischetti (✉)
Technical University of Denmark, Kongens Lyngby, Copenhagen, Denmark

Vattenfall BA Wind, Kongens Lyngby, Copenhagen, Denmark
e-mail: martina.fischetti@vattenfall.com

M. Fischetti
DEI, University of Padova, Padova, Italy
e-mail: matteo.fischetti@unipd.it

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_14

Abstract

As its name suggests, a matheuristic is the hybridization of mathematical programming with metaheuristics. The hallmark of matheuristics is the central role played by the mathematical programming model, around which the overall heuristic is built. As such, matheuristic is not a rigid paradigm but rather a concept framework for the design of mathematically sound heuristics. The aim of this chapter is to introduce the main matheuristic ideas. Three specific applications in the field of wind farm, packing, and vehicle routing optimization, respectively, are addressed and used to illustrate the main features of the method.

Keywords

Heuristics · Large scale neighborhood search · Local branching · Mathematical programming · Matheuristics

Introduction

The design of heuristics for difficult optimization problems is itself a heuristic process that often involves the following main steps.

After a clever analysis of the problem at hand and of the acceptable simplifications in its definition, one tries to set up an effective **mathematical programming** (MP) model and to solve it by a general-purpose piece of software—often a mixed-integer linear programming (MIP) solver. Due to the impressive improvement of general-purpose solvers in recent years, this approach can actually solve the instances of interest to proven optimality (or with an acceptable approximation) within a reasonable computing time, in which case of course no further effort is needed.

If this is not the case, one can insist on the MP approach and try to obtain better and better results by improving the model and/or by enhancing the solver by specialized features (cutting planes, branching, etc.). Or one can forget about MP and resort to ad hoc heuristics not based on the MP model. In this latter case, the MP model is completely disregarded or just used for illustrating the problem characteristics and/or for getting an off-line indication of the typical approximation error on a set of sample instances.

A third approach is however possible that consists in using the MP solver as a basic tool *within* the heuristic framework. This hybridization of MP with *metaheuristics* leads to the *matheuristic* approach, where the heuristic is built around the MP model. Matheuristics became popular in recent years, as witnessed by the publication of dedicated volumes and journal special issues [8, 19, 24] and by the dedicated sessions on MP and metaheuristic conferences.

Designing an effective heuristic is an art that cannot be framed into strict rules. This is particularly true when addressing a matheuristic, which is not a rigid paradigm but a concept framework for the design of mathematically sound heuristics. In this chapter, we will therefore try to illustrate some main matheuristic features with the help of different examples of application.

Section "General-Purpose MIP-Based Heuristics" describes powerful general-purpose MIP heuristics that can be used within the matheuristic framework. Interestingly, these heuristics can themselves be viewed as the first successful applications of the matheuristic idea of hybridizing MP and metaheuristics. Indeed, as noticed in [8], one of the very first illustrations of the power of the matheuristic idea is the general-purpose local branching [7] paradigm, where a black-box MIP solver is used to explore a solution neighborhood defined by invalid constraints added to the MIP model for the sake of easing its solution.

Section "Application 1: Wind Farm Layout Optimization" addresses the design of a matheuristic for wind farm optimization. This application is used to illustrate the importance of the choice of the MIP model: models that are weak in polyhedral terms can be preferred to tighter—but computationally much harder—models when heuristic (as opposed to exact) solutions are required.

Section "Application 2: Prepack Optimization" addresses a packing problem where the model is nonlinear, and the matheuristic is based on various ways to linearize it after a heuristic fixing of some variables.

Finally, section "Application 3: Vehicle Routing" is used to illustrate an advanced feature of matheuristics, namely, the solution of auxiliary MP models that describe a subproblem in the solution process. In particular, we address a vehicle routing problem and derive a matheuristic based on a set-partitioning MIP model asking for the reallocation of a subset of customer sequences subject to capacity and distance constraints.

The present chapter is based on previous published work; in particular, sections "General-Purpose MIP-Based Heuristics", "Application 1: Wind Farm Layout Optimization", "Application 2: Prepack Optimization", and "Application 3: Vehicle Routing" are based on [8, 11, 13, 15], respectively.

General-Purpose MIP-Based Heuristics

Heuristics for general-purpose MIP solvers form the basis of the matheuristic's toolkit. Their relevance for our chapter is twofold. On the one hand, they are invaluable tools for the solution of the subproblems tailored by the matheuristic when applied to a specific problem. On the other hand, they illustrate the benefits for a general-purpose MIP solver deriving from the use of metaheuristics concepts such as local search and evolutionary methods.

Modern MIP solvers exploit a rich arsenal of tools to attack hard problems. It is widely accepted that the solution of hard MIPs can take advantage from the solution of a series of auxiliary linear programs (LPs) intended to enhance the performance of the overall MIP solver. For example, auxiliary LPs may be solved to generate powerful disjunctive cuts or to implement a strong branching policy. On the other hand, it is a common experience that finding good-quality heuristic MIP solutions often requires a computing time that is just comparable to that needed to solve the LP relaxation. So, it makes sense to think of exact/heuristic MIP solvers where auxiliary MIPs (as opposed to LPs) are heuristically solved on the fly, with the aim

of bringing the MIP technology under the chest of the MIP solver itself. This leads to the idea of "translating into a MIP model" (*MIPping* in the jargon of [9]) some crucial decisions to be taken when designing a MIP-based algorithm.

We next describe the new generation of MIP heuristics that emerged in the late 1990s, which are based on the idea of systematically using a "black-box" external MIP solver to explore a solution neighborhood defined by invalid linear constraints. We address a generic MIP of the form

$$
\begin{aligned}
(MIP) \qquad & \min c^T x && (1)\\
& Ax \geq b, && (2)\\
& x_j \in \{0,1\},\ \forall j \in \mathcal{B}, && (3)\\
& x_j \text{ integer},\ \forall j \in \mathcal{G}, && (4)\\
& x_j \text{ continuous},\ \forall j \in \mathcal{C}, && (5)
\end{aligned}
$$

where A is an $m \times n$ input matrix and b and c are input vectors of dimension m and n, respectively. Here, the variable index set $\mathcal{N} := \{1, \ldots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$, where $\mathcal{B}$ is the index set of the 0-1 variables (if any), while sets $\mathcal{G}$ and $\mathcal{C}$ index the general integer and the continuous variables, respectively. Removing the integrality requirement on variables indexed by $\mathcal{I} := \mathcal{B} \cup \mathcal{G}$ leads to the so-called *LP relaxation*.

Local Branching

The *local branching* (LB) scheme of Fischetti and Lodi [7] appears to be one of the first general-purpose heuristics using a black-box MIP solver applied to subMIPs, and it can be viewed as a precursor of matheuristics. Given a *reference solution* $\bar{x}$ of a MIP with $\mathcal{B} \neq \emptyset$, one aims at finding an improved solution that is "not too far" from $\bar{x}$, in the sense that not too many binary variables need be flipped. To this end, one can define the k-opt neighborhood $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the MIP solutions satisfying the invalid *local branching constraint*

$$
\Delta(x, \bar{x}) := \sum_{j \in \mathcal{B}: \bar{x}_j = 0} x_j + \sum_{j \in \mathcal{B}: \bar{x}_j = 1} (1 - x_j) \leq k, \tag{6}
$$

for a small neighborhood radius k—an integer parameter typically set to 10 or 20. The neighborhood is then explored (possibly heuristically, i.e., with some small node or time limit) by means of a black-box MIP solver. Experimental results [10] show that the introduction of the local branching constraint typically has the positive effect of driving to integrality many component of the optimal solution of the LP relaxation, improving the so-called relaxation grip and hence the capability of the MIP solver to find (almost) optimal integer solutions within short computing times.

Of course, this effect is lost if parameter k is set to a large value—a mistake that would make local branching completely ineffective.

LB is in the spirit of local search metaheuristics and, in particular, of *large-neighborhood search* (LNS) [29], with the novelty that neighborhoods are obtained through "soft fixing," i.e., through invalid cuts to be added to the original MIP model. Diversification cuts can be defined in a similar way, thus leading to a flexible toolkit for the definition of metaheuristics for general MIPs.

Relaxation-Induced Neighborhood Search

The *relaxation-induced neighborhood search* (RINS) heuristic of Danna, Rothberg, and Le Pape [4] also uses a black-box MIP solver to explore a neighborhood of a given solution $\bar{x}$ and was originally designed to be integrated in a branch-and-bound solution scheme. At specified nodes of the branch-and-bound tree, the current LP relaxation solution x^* and the incumbent $\bar{x}$ are compared, and all integer-constrained variables that agree in value are fixed. The resulting MIP is typically easy to solve, as fixing reduces its size considerably, and often provides improved solutions with respect to $\bar{x}$.

Polishing a Feasible Solution

The *polishing* algorithm of Rothberg [27] implements an evolutionary MIP heuristic which is invoked at selected nodes of a branch-and-bound tree and includes all classical ingredients of genetic computation, namely:

- *Population*: A fixed-size population of feasible solutions is maintained. Those solutions are either obtained within the branch-and-bound tree (by other heuristics) or computed by the polishing algorithm itself.
- *Combination*: Two or more solutions (the parents) are combined with the aim of creating a new member of the population (the child) with improved characteristics. The RINS scheme is adopted, i.e., all variables whose value coincides in the parents are fixed, and the reduced MIP is heuristically solved by a black-box MIP solver within a limited number of branch-and-bound nodes. This scheme is clearly much more time-consuming than a classical combination step in evolutionary algorithms, but it guarantees feasibility of the child solution.
- *Mutation*: Diversification is obtained by performing a classical mutation step that (i) randomly selects a "seed" solution in the population, (ii) randomly fixes some of its variables, and (iii) heuristically solves the resulting reduced MIP.
- *Selection*: Selection of the two parents to be combined is performed by randomly picking a solution in the population and then choosing, again at random, the second parent among those solutions with a better objective value.

Proximity Search

Proximity search [10] is a "dual version" of local branching that tries to overcome the issues related to the choice of the neighborhood radius k. Instead of hard-fixing the radius, proximity search fixes the minimum improvement of the solution value and changes the objective function to favor the search of solutions at small Hamming distance with respect to the reference one.

The approach works in stages, each aimed at producing an improved feasible solution. As in LB or RINS, at each stage a reference solution $\bar{x}$ is given, and one aims at improving it. To this end, an explicit cutoff constraint

$$c^T x \leq c^T \bar{x} - \theta \tag{7}$$

is added to the original MIP, where $\theta > 0$ is a given tolerance that specifies the minimum improvement required. The objective function of the problem can then be replaced by the proximity function $\Delta(x, \bar{x})$ defined in (6), to be minimized. One then applies the MIP solver, as a black box, to the modified problem in the hope of finding a solution better than $\bar{x}$. Computational experience confirms that this approach is quite successful (at least, on some classes of problems), due to the action of the proximity objective function that improves the "relaxation grip" of the model.

A simple variant of the above scheme, called "proximity search with incumbent," is based on the idea of providing $\bar{x}$ to the MIP solver as a staring solution. To avoid $\bar{x}$ be rejected because of the cutoff constraint (7), the latter is weakened to its "soft" version

$$c^T x \leq c^T \bar{x} - \theta(1 - \xi) \tag{8}$$

while minimizing $\Delta(x, \bar{x}) + M\xi$ instead of just $\Delta(x, \bar{x})$, where $\xi \geq 0$ is a continuous slack variable and $M \gg 0$ is a large penalty.

Application 1: Wind Farm Layout Optimization

Green energy became a topic of great interest in recent years, as environmental sustainability asks for a considerable reduction in the use of fossil fuels. The *wind farm layout optimization problem* aims at finding an allocation of turbines in a given site so as to maximize power output. This strategic problem is extremely hard in practice, both for the size of the instances in real applications and for the presence of several nonlinearities to be taken into account. A typical nonlinear feature of this problem is the interaction among turbines, also known as wake effect. The wake effect is the interference phenomenon for which, if two turbines are located one close to another, the upwind one creates a shadow on the one behind. Interference is therefore of great importance in the design of the layout as it results into a loss of power production for the turbine downstream.

We next outline the main steps in the design of a sound matheuristic scheme for wind farm layout optimization that is able to address the large-size instances arising in practical applications.

Choice of the MIP Model

Different models have been proposed in the literature to describe interference. We will consider first a simplified model from the literature [5], where the overall interference is the sum of pairwise interferences between turbine pairs. The model addresses the following constraints:

(a) a minimum and maximum number of turbines that can be built is given;
(b) there should be a minimal separation distance between two turbines to ensure that the blades do not physically clash (turbine distance constraints);
(c) if two turbines are installed, their interference will cause a loss in the power production that depends on their relative position and on wind conditions.

Let V denote the set of possible positions for a turbine, called "sites" in what follows, and let

- N_{MIN} and N_{MAX} be the minimum and maximum number of turbines that can be built, respectively;
- D_{MIN} be the minimum distance between two turbines;
- $dist(i, j)$ be the Euclidean distance between sites i and j;
- I_{ij} be the interference (loss of power) experienced by site j when a turbine is installed at site i, with $I_{jj} = 0$ for all $j \in V$;
- P_i be the power that a turbine would produce if built (alone) at site i.

In addition, let $G_I = (V, E_I)$ denote the incompatibility graph with

$$E_I = \{[i, j] \in V \times V : dist(i, j) < D_{\text{MIN}}, i < j\}$$

and let $n := |V|$ denote the total number of sites. Two sets of binary variables are defined:

$$x_i = \begin{cases} 1 & \text{if a turbine is built at site } i; \\ 0 & \text{otherwise} \end{cases} \qquad (i \in V)$$

$$z_{ij} = \begin{cases} 1 & \text{if two turbines are built at both sites } i \text{ and } j; \\ 0 & \text{otherwise} \end{cases} \qquad (i, j \in V, i < j)$$

The model then reads

$$\max \sum_{i \in V} P_i x_i - \sum_{i \in V} \sum_{j \in V, i<j} (I_{ij} + I_{ji}) z_{ij} \tag{9}$$

$$\text{s.t.} \qquad N_{\text{MIN}} \leq \sum_{i \in V} x_i \leq N_{\text{MAX}} \tag{10}$$

$$x_i + x_j \leq 1 \qquad \forall [i, j] \in E_I \tag{11}$$

$$x_i + x_j - 1 \leq z_{ij} \qquad \forall i, j \in V, i < j \tag{12}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V \tag{13}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i < j \tag{14}$$

Objective function (9) maximizes the total power production by taking interference losses I_{ij} into account. Constraints (11) model pairwise site incompatibility. Constraints (12) force $z_{ij} = 1$ whenever $x_i = x_j = 1$; because of the objective function, this is in fact equivalent to imposing $z_{ij} = x_i x_j$.

The definition of the turbine power vector (P_i) and of interference matrix (I_{ij}) depends on the wind scenario considered, which greatly varies in time. Using statistical data, one can in fact collect a large number K of wind scenarios k, each associated with a pair (P^k, I^k) with a probability π_k, and define the average power and interference to be used in the model as:

$$P_i := \sum_{k=1}^{K} \pi_k P_i^k \qquad \forall i \in V \tag{15}$$

$$I_{ij} := \sum_{k=1}^{K} \pi_k I_{ij}^k \qquad \forall i, j \in V \tag{16}$$

While (9), (10), (11), (12), (13), and (14) turns out to be a reasonable model when just a few sites have to be considered (say $n \approx 100$), it becomes hopeless when $n \geq 1000$ because of the huge number of variables and constraints involved, which grows quadratically with n. Therefore, when facing instances with several thousand sites, an alternative (possibly weaker) model is required, where interference can be handled by a number of variables and constraints that grows just linearly with n. The model below is a compact reformulation of model (9), (10), (11), (12), (13), and (14) that follows a recipe of Glover [17] that is widely used, e.g., in the quadratic assignment problem [12, 32]. The original objective function (to be maximized), rewritten as

$$\sum_{i \in V} P_i x_i - \sum_{i \in V} \left(\sum_{j \in V} I_{ij} x_j \right) x_i \tag{17}$$

is restated as

$$\sum_{i \in V} (P_i x_i - w_i) \tag{18}$$

where

$$w_i := \left(\sum_{j \in V} I_{ij} x_j \right) x_i = \begin{cases} \sum_{j \in V} I_{ij} x_j & \text{if } x_i = 1 \\ 0 & \text{if } x_i = 0 \end{cases}$$

denotes the total interference caused by site i. Our compact model then reads

$$\max z = \sum_{i \in V} (P_i x_i - w_i) \tag{19}$$

$$\text{s.t.} \quad N_{\text{MIN}} \leq \sum_{i \in V} x_i \leq N_{\text{MAX}} \tag{20}$$

$$x_i + x_j \leq 1 \qquad \forall [i, j] \in E_I \tag{21}$$

$$\sum_{j \in V} I_{ij} x_j \leq w_i + M_i (1 - x_i) \qquad \forall i \in V \tag{22}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V \tag{23}$$

$$w_i \geq 0 \qquad \forall i \in V \tag{24}$$

where the big-M term $M_i = \sum_{j \in V : [i,j] \notin E_I} I_{ij}$ is used to deactivate constraint (22) in case $x_i = 0$, in which case $w_i = 0$ because of the objective function.

Choice of Ad Hoc Heuristics

A simple 1-opt heuristic can be designed along the following lines. At each step, we have an incumbent solution, say $\tilde{x}$, that describes the best-known turbine allocation ($\tilde{x}_i = 1$ if a turbine is built at site i, 0 otherwise), and a current solution x. Let

$$z = \sum_{i \in V} P_i x_i - \sum_{i \in V} \sum_{j \in V} I_{ij} \, x_i \, x_j$$

be the profit of the current solution, $\gamma = \sum_{i \in V} x_i$ be its cardinality, and define for each $j \in V$ the extra-profit δ_j incurred when flipping x_j, namely:

$$\delta_j = \begin{cases} P_j - \sum_{i \in V : x_i = 1} (I_{ij} + I_{ji}) & \text{if } x_j = 0; \\ -P_j + \sum_{i \in V : x_i = 1} (I_{ij} + I_{ji}) & \text{if } x_j = 1 \end{cases}$$

where we assume $I_{ij} = BIG$ for all incompatible pairs $[i, j] \in E_I$, and $BIG > \sum_{i \in V} P_i$ is a large penalty value, while $I_{ii} = 0$ as usual.

We start with $x = 0$, $z = 0$, and $\gamma = 0$ and initialize $\delta_j = P_j$ for all $j \in V$. Then, we iteratively improve x by a sequence of 1-opt moves, according to the following scheme. At each iteration, we look in $O(n)$ time for the site j with maximum $\delta_j + FLIP(j)$, where function $FLIP(j)$ takes cardinality constraints into account, namely

$$FLIP(j) = \begin{cases} -HUGE & \text{if } x_j = 0 \text{ and } \gamma \geq N_{MAX} \\ -HUGE & \text{if } x_j = 1 \text{ and } \gamma \leq N_{MIN} \\ +HUGE & \text{if } x_j = 0 \text{ and } \gamma < N_{MIN} \\ +HUGE & \text{if } x_j = 1 \text{ and } \gamma > N_{MAX} \\ 0 & \text{otherwise} \end{cases}$$

with $HUGE \gg BIG$ (recall that function $\delta_j + FLIP_j$ has to be maximized).

Once the best j has been found, say j^*, if $\delta_{j^*} + FLIP(j^*) > 0$, we just flip x_{j^*}; update x, z, and γ in $O(1)$ time; update all δ_j's in $O(n)$ time (through the parametric technique described in [11]); and repeat. In this way, a sequence of improving solutions is obtained, until a local optimal solution that cannot be improved by just one flip is found. To escape local minima, a simple perturbation scheme can be implemented; see again [11] for details.

A 2-opt heuristic can similarly be implemented to allow a single turbine to move to a better site—a move that requires flipping two variables. Each 2-opt exchange requires $O(n^2)$ time as it amounts to trying n 1-opt exchanges and to apply the best one.

The Overall Matheuristic

Our final approach is a mixture of ad hoc (1- and 2-opt) and general MIP (proximity search with incumbent) heuristics and works as shown in Algorithm 1.

At Step 2, the heuristics of section "Choice of Ad Hoc Heuristics" are applied in their "initial-solution" mode where one starts with $\tilde{x} = x = 0$ and aborts execution when 1-opt is invoked 10,000 consecutive times without improving $\tilde{x}$. At Step 4, instead, a faster "cleanup" mode is applied. As we already have a hopefully good incumbent $\tilde{x}$ to refine, we initialize $x = \tilde{x}$ and repeat the procedure until we count 100 consecutive 1-opt calls with no improvement of $\tilde{x}$. As to time-consuming 2-opt exchanges, they are applied with a certain frequency and in any case just before the final $\tilde{x}$ is returned.

Two different MIP models are used to feed the proximity-search heuristic at Step 6. During the first part of the computation, we use a simplified MIP model obtained from (19), (20), (21), (22), (23), and (24) by removing all interference constraints (22), thus obtaining a much easier problem. A short time limit is imposed for each call of proximity search when this simplified model is solved. In this way we aggressively drive the solution $\tilde{x}$ to increase the number of built turbines,

Algorithm 1: The overall matheuristic framework

1: read input data and compute the overall interference matrix (I_{ij});
2: apply ad hoc heuristics (1- and 2-opt) to get a first incumbent $\tilde{x}$;
3: **while** time limit permits **do**
4: apply quick ad hoc refinement heuristics (few iterations of 1- and 2-opt) to possibly improve $\tilde{x}$;
5: if $n > 2000$, randomly remove points $i \in V$ with $\tilde{x}_i = 0$ so as to reduce the number of candidate sites to 2000;
6: build a MIP model for the resulting subproblem and apply proximity search to refine $\tilde{x}$ until the very first improved solution is found (or time limit is reached);
7: **end while**
8: **return** $\tilde{x}$

without being bothered by interference considerations and only taking pairwise incompatibility (21) into account. This approach quickly finds better and better solutions (even in terms of the true profit), until either (i) no additional turbine can be built or (ii) the addition of new turbines does in fact *reduce* the true profit associated to the new solution because of the neglected interference. In this situation we switch to the complete model (19), (20), (21), (22), (23), and (24) with all interference constraints, which is used in all next executions of Step 6. Note that the simplified model is only used at Step 6, while all other steps of the procedure always use the true objective function that takes interference into full account.

Computational Results

The following alternative solution approaches were implemented in C language, some of which using the commercial MIP-solver IBM ILOG Cplex 12.5.1 [21]; because of the big-Ms involved in the models, all Cplex's codes use zero as integrality tolerance (`CPX_PARAM_EPINT` = 0.0).

(a) `proxy`: The matheuristic outlined in the previous section, built on top of Cplex with the following aggressive parameter tuning: all cuts deactivated, `CPX_PARAM_RINSHEUR` = 1, `CPX_PARAM_POLISHAFTERTIME` = 0.0, `CPX_PARAM_INTSOLLIM` = 2;
(b) `cpx_def`: The application of IBM ILOG Cplex 12.5.1 in its default setting, starting from the same heuristic solution $\tilde{x}$ available right after the first execution of Step 2 of Algorithm 1;
(c) `cpx_heu`: Same as `cpx_def`, with the following internal tuning intended to improve Cplex's heuristic performance: all cuts deactivated, `CPX_PARAM_RINSHEUR` = 100, `CPX_PARAM_POLISHAFTERTIME` = 20% of the total time limit;

(d) `loc_sea`: A simple heuristic not based on any MIP solver, that just loops on Steps 4 of Algorithm 1 and randomly removes installed turbines from the current best solution after 10,000 iterations without improvement of the incumbent.

For each algorithm, we recorded the best solution found within a given time limit.

In our view, `loc_sea` is representative of a clever but not oversophisticated metaheuristic, as typically implemented in practice, while `cpx_def` and `cpx_heu` represent a standard way of exploiting a MIP model once a good feasible solution is known.

Our test bed refers to an offshore 3,000 × 3,000 (m) square with $D_{\text{MIN}} = 400$ (m) minimum turbine separation, with no limit on the number of turbines to be built (i.e., $N_{\text{MIN}} = 0$ and $N_{\text{MAX}} = +\infty$). Turbines are all of Siemens SWT-2.3-93 type (rotor diameter 93 m), which produces a power of 0.0 MW for wind speed up to 3 m/s, of 2.3 MW for wind speed greater than or equal to 16 m/s, and intermediate values for winds in range 3–16 m/s according to a nonlinear power curve [30]. Pairwise interference (in MW) was computed using Jensen's model [22], by averaging 250,000+ real-world wind samples. Those samples were grouped into about 500 macro-scenarios to reduce the computational time spent for the definition of the interference matrix. A pairwise average interference of 0.01 MW or less was treated as zero. The reader is referred to [6] for details.

We generated five classes of medium-to-large problems with n ranging from 1,000 to 20,000. For each class, ten instances have been considered by generating n uniformly random points in the 3,000 × 3,000 square. (Although in the offshore case turbine positions are typically sampled on a regular grid, we decided to randomly generate them to be able to compute meaningful statistics for each value of n.)

In what follows, reported computing times are in CPU sec.s of an Intel Xeon E3-1220 V2 quad-core PC with 16GB of RAM and do not take Step 1 of Algorithm 1 into account as the interference matrix is assumed to be precomputed and reused at each run.

Computational results on our instances are given in Table 1, where each entry refers to the performance of a given algorithm at a given time limit. In particular, the left part of the table reports, for each algorithm and time limit, the *number of wins*, i.e., the number of instances for which a certain algorithm produced the best-known solution at the given time limit (ties allowed).

According to the table, `proxy` outperforms all competitors by a large amount for medium-to-large instances. As expected, `cpx_heu` performs better for instances with $n = 1{,}000$ as it is allowed to explore a large number of enumeration nodes for the original model and objective function. Note that `loc_sea` has a good performance for short time limits and/or for large instances, thus confirming its effectiveness, whereas `cpx_heu` is significantly better than `loc_sea` only for small instances and large time limits.

A different performance measure is given in the right-hand side part of Table 1, where each entry gives the *average optimality ratio*, i.e., the average value of the

Table 1 Number of times each algorithm finds the best-known solution within the time limit (wins) and optimality ratio with respect to the best-known solution—the larger, the better

		Number of wins				Optimality ratio			
n	Time limit (s)	proxy	cpx_def	cpx_heu	loc_sea	proxy	cpx_def	cpx_heu	loc_sea
1,000	60	6	1	3	0	0.994	0.983	0.987	0.916
	300	4	2	4	0	0.997	0.991	0.998	0.922
	600	7	3	7	0	0.997	0.992	0.997	0.932
	900	5	2	3	0	0.998	0.993	0.996	0.935
	1,200	5	1	5	0	0.998	0.992	0.997	0.939
	1,800	5	1	4	0	0.998	0.992	0.996	0.942
	3,600	4	2	5	0	0.998	0.995	0.997	0.943
5,000	60	9	6	6	5	0.909	0.901	0.901	0.904
	300	10	0	0	0	0.992	0.908	0.908	0.925
	600	10	0	10	0	0.994	0.908	0.994	0.935
	900	10	0	0	0	0.994	0.908	0.908	0.936
	1,200	10	0	0	0	0.994	0.908	0.925	0.939
	1,800	9	0	1	0	0.996	0.908	0.971	0.946
	3,600	5	0	5	0	0.996	0.932	0.994	0.948
10,000	60	9	9	8	10	0.914	0.913	0.914	0.914
	300	10	2	2	2	0.967	0.927	0.927	0.936
	600	10	0	10	0	0.998	0.928	0.998	0.944
	900	10	0	0	0	1.000	0.928	0.928	0.948
	1,200	10	0	0	0	1.000	0.928	0.928	0.951
	1,800	10	0	0	0	1.000	0.928	0.928	0.957
	3,600	9	0	0	1	1.000	0.928	0.928	0.964
15,000	60	9	10	9	9	0.909	0.912	0.911	0.909
	300	10	8	7	8	0.943	0.937	0.935	0.937
	600	10	0	10	0	0.992	0.939	0.992	0.942
	900	10	0	0	0	1.000	0.939	0.939	0.956
	1,200	9	0	0	1	1.000	0.939	0.939	0.959
	1,800	9	0	0	1	1.000	0.939	0.939	0.965
	3,600	9	0	0	1	1.000	0.939	0.939	0.972
20,000	60	9	9	9	10	0.901	0.902	0.901	0.902
	300	10	8	10	10	0.933	0.933	0.933	0.933
	600	9	0	9	1	0.956	0.935	0.956	0.941
	900	10	0	0	0	0.978	0.935	0.935	0.945
	1,200	10	0	0	0	0.991	0.935	0.935	0.950
	1,800	10	0	0	0	0.999	0.935	0.935	0.963
	3,600	10	0	0	0	1.000	0.935	0.935	0.971
ALL	60	42	35	35	34	0.925	0.922	0.922	0.909
	300	44	20	23	20	0.966	0.939	0.940	0.930
	600	46	3	46	1	0.987	0.941	0.987	0.938
	900	45	2	3	0	0.994	0.941	0.941	0.944
	1,200	44	1	5	1	0.997	0.940	0.945	0.947
	1,800	43	1	5	1	0.999	0.940	0.954	0.955
	3,600	36	2	10	2	0.999	0.946	0.959	0.959

ratio between the solution produced by an algorithm (on a given instance at a given time limit) and the best solution known for that instance—the closer to one, the better. It should be observed that an improvement of just 1% has a very significant economical impact due to the very large profits involved in the wind farm context. The results show that `proxy` is always able to produce solutions that are quite close to the best one. As before, `loc_sea` is competitive for large instances when a very small computing time is allowed, whereas `cpx_def` and `cpx_heu` exhibit a good performance only for small instances and are dominated even by `loc_sea` for larger ones.

Application 2: Prepack Optimization

Packing problems play an important role in industrial applications. In these problems, a given set of *items* has to be packed into one or more containers (*bins*) so as to satisfy a number of constraints and to optimize some objective function.

Most of the contributions from the literature are devoted to the case where all the items have to be packed into a minimum number of bins so as to minimize, e.g., transportation costs; within these settings, only loading costs are taken into account. The resulting problem is known as the *bin packing problem* and has been widely studied in the literature both in its one-dimensional version [25] and in its higher-dimensional variants [23].

We will next consider a different packing problem arising in inventory allocation applications, where the operational cost for packing the bins is comparable, or even higher, than the cost of the bins themselves. This is the case, for example, for warehouses that have to manage a large number of different customers (e.g., stores), each requiring a given set of items. Assuming that automatic systems are available for packing, the required workforce is related to the number of different ways that are used to pack the bins to be sent to the customers. To limit this cost, a hard constraint can be imposed on the total number of different box configurations that are used.

Prepacking items into box configurations has obvious benefits in terms of easier and cheaper handling, as it reduces the amount of material handled by both the warehouse and the customers. However, the approach can considerably reduce the flexibility of the supply chain, leading to situations in which the set of items that are actual shipped to each customer may slightly differ from the required one—at the expense of some cost in the objective function. In addition, an upper bound on overstocking is usually imposed for each store.

The resulting problem, known as *prepack optimization problem* (POP), was recently addressed in [20], where a real-world application in the fashion industry is presented, and heuristic approaches are derived using both constraint programming (CP) and MIP techniques.

Mathematical Model

In this section we briefly formalize POP and review the mathematical model introduced in [20]. We are given a set I of types of products and a set S of stores. Each store $s \in S$ requires an integer number r_{is} of products of type $i \in I$. Bins with different capacities are available for packing items: we denote by $K \subset Z_+$ the set of available bin capacities.

Bins must be completely filled and are available in an unlimited number for each type. A *box configuration* describes the packing of a bin, in terms of number of products of each type that are packed into it. We denote by NB the maximum number of box configurations that can be used for packing all products and by $B = \{1, \ldots, NB\}$ the associated set.

Products' packing into boxes is described by integer variables y_{bi}: for each product type $i \in I$ and box configuration $b \in B$, the associated variable y_{bi} indicates the number of products of type i that are packed into the b-th box configuration. In addition, integer variables x_{bs} are used to denote the number of bins loaded according to box configuration b that have to be shipped to store $s \in S$.

Understocking and overstocking of product i at store s are expressed by decisional variables u_{is} and o_{is}, respectively. Positive costs α and β penalize each unit of under- and overstocking, respectively, whereas an upper bound δ_{is} on the maximum overstocking of each product at each store is also imposed.

Finally, for each box configuration $b \in B$ and capacity value $k \in K$, a binary variable t_{bk} is introduced that takes value 1 if box configuration b corresponds to a bin of capacity k.

Additional integer variables used in the model are $q_{bis} = x_{bs}\, y_{bi}$ (number of items of type i sent to store s through boxes loaded with configuration b); hence, $\sum_{b \in B} q_{bis}$ gives the total number of products of type i that are shipped to store s.

A mixed-integer nonlinear programming (MINLP) model then reads:

$$\min \sum_{s \in S} \sum_{i \in I} (\alpha u_{is} + \beta o_{is}) \tag{25}$$

$$q_{bis} = x_{bs} y_{bi} \quad (b \in B; i \in I; s \in S) \tag{26}$$

$$\sum_{b \in B} q_{bis} - o_{is} + u_{is} = r_{is} \quad (i \in I; s \in S) \tag{27}$$

$$\sum_{i \in I} y_{bi} = \sum_{k \in K} k\, t_{bk} \quad (b \in B) \tag{28}$$

$$\sum_{k \in K} t_{bk} = 1 \quad (b \in B) \tag{29}$$

$$o_{is} \leq \delta_{is} \quad (i \in I; s \in S) \tag{30}$$

$$t_{bk} \in \{0, 1\} \quad (b \in B; k \in K) \tag{31}$$

$$x_{bs} \geq 0 \text{ integer} \quad (b \in B; s \in S) \tag{32}$$

$$y_{bi} \geq 0 \text{ integer} \quad (b \in B; i \in I) \tag{33}$$

The model is of course nonlinear, as the bilinear constraints (26) involve the product of decision variables. To derive a linear MIP model, the following standard technique can be used. Each x_{bs} variable is decomposed into its binary expansion using binary variables v_{bsl} ($l = 0, \ldots, L$), where L is easily computed from an upper bound on x_{bs}. When these variables are multiplied by y_{bi}, the corresponding product $w_{bisl} = v_{bsl}\, y_{bi}$ are linearized with the addition of suitable constraints.

Our final MIP is therefore obtained from (25), (26), (27), (28), (29), (30), (31), (32), and (33) by adding

$$x_{bs} = \sum_{l=0}^{L} 2^l v_{bsl} \quad (b \in B; s \in S) \tag{34}$$

$$v_{bsl} \in \{0, 1\} \quad (b \in B; s \in S; l = 0, \ldots, L) \tag{35}$$

and by replacing each nonlinear equation (26) with the following set of new variables and constraints:

$$q_{bis} = \sum_{l=0}^{L} 2^l w_{bisl} \quad (b \in B; i \in I; s \in S) \tag{36}$$

$$w_{bisl} \leq \overline{Y} v_{bsl} \quad (b \in B; i \in I; s \in S; l = 0, \ldots, L) \tag{37}$$

$$w_{bisl} \leq y_{bi} \quad (b \in B; i \in I; s \in S; l = 0, \ldots, L) \tag{38}$$

$$w_{bisl} \geq y_{bi} - \overline{Y}(1 - v_{bsl}) \quad (b \in B; i \in I; s \in S; l = 0, \ldots, L) \tag{39}$$

$$w_{bisl} \geq 0 \quad (b \in B; i \in I; s \in S; l = 0, \ldots, L) \tag{40}$$

where $\overline{Y}$ denotes an upper bound on the y variables.

In case all capacities are even, the following constraint—though redundant—plays a very important role in improving the LP bound of our MIP model:

$$\sum_{i \in I} (u_{is} + o_{is}) \geq 1 \quad \left(s \in S : \sum_{i \in I} r_{is} \text{ is odd} \right) \tag{41}$$

Matheuristics

The MIP model of the previous subsection is by far too difficult to be addressed by standard solvers. As a matter of fact, for real-world cases even the LP relaxation at each node turns out to be very time consuming. So we designed ad hoc heuristic

approaches to exploit the special structure of our MIP, following the matheuristic paradigm.

Each heuristic is based on the idea of iteratively solving a restricted problem obtained by fixing a subset of variables, so as to obtain a subproblem which is (reasonably) easy to solve by a commercial MIP solver, but still able to produce improved solutions.

Two kinds of heuristics can be implemented: constructive and refinement heuristics. Constructive heuristics are used to find a solution H starting from scratch. In a refinement heuristic, instead, we are given a heuristic solution $H = (x^H, y^H)$ that we would like to improve. We first fix some x and/or y variables to their value in H, thus defining a solution neighborhood $\mathcal{N}(H)$ of H. We then search $\mathcal{N}(H)$ by using a general-purpose MIP solver on the model resulting from fixing. If an improved solution is found within the given time limit, we update H and repeat; otherwise, a new neighborhood is defined in the attempt to escape the local optimum.

Fixing all *x* or *y* Variables

A first obvious observation is that our basic MINLP model (25), (26), (27), (28), (29), (30), (31), (32), and (33) reduces to a linear MIP if all the x (or all the y) variables are fixed, as constraints (26) trivially become linear. According to our experience, the resulting MIP (though nontrivial) is typically solved very quickly by a state-of-the-art solver, meaning that one can effectively solve a sequence of restricted MIPs where x and y are fixed, in turn, until no further improvement can be obtained.

Let $\mathcal{N}_y(H)$ and $\mathcal{N}_x(H)$ denote the solution neighborhoods of H obtained by leaving y or x free, i.e., when imposing $x = x^H$ or $y = y^H$, respectively.

A basic tool that we use in our heuristics is function $\texttt{REOPT}(S', y^H)$ that considers a store subset $S' \subseteq S$ and a starting y^H and returns the best solution H obtained by iteratively optimizing over the neighborhoods $\mathcal{N}_x(H)$, $\mathcal{N}_y(H)$, $\mathcal{N}_x(H)$, etc. after having removed all stores not in S', H being updated after each optimization.

Fixing *y* Variables For All but One Configuration

Another interesting neighborhood, say $\mathcal{N}_x(H, y_\beta)$, is obtained by leaving all x variables free and by fixing $y_{bi} = y^H_{bi}$ for all $i \in I$ and $b \in B \setminus \{\beta\}$ for a given $\beta \in B$. In other words, we allow for changing just one (out of NB) configuration in the current solution, and leave the solver the possibility to change the x variables as well.

In our implementation, we first define a random permutation $\{\beta_1, \ldots, \beta_{NB}\}$ of B. We then optimize, in a circular sequence, neighborhoods $\mathcal{N}_x(H, y_{\beta_t})$ for $t = 1, \ldots, NB, 1, \ldots$ Each time an improved solution is found, we update H and further refine it through function $\texttt{REOPT}(S, y^H)$. The procedure is stopped when there is no hope of finding an improved solution, i.e., after NB consecutive optimizations that do not improve the current H.

A substantial speedup can be obtained by heuristically imposing a tight upper bound on the x variables, so as to reduce the number $L + 1$ of binary variables

v_{bsl} in the binary expansion (34). An aggressive policy (e.g., imposing $x_{bs} \leq 1$) is however rather risky as the optimal solution could be cut off; hence, the artificial bounds must be relaxed if an improved solution cannot be found.

Working with a Subset of Stores

Our basic constructive heuristic is based on the observation that removing stores can produce a substantially easier model. Note that a solution $H' = (x', y')$ with a subset of stores can easily be converted into a solution $H = (x, y)$ of the whole problem by just invoking function `REOPT`(S, y').

In our implementation, we first define a store permutation $s_1, \ldots, s_{|S|}$ according to a certain criterion (to be discussed later). We then address, in sequence, the subproblem with store set $S_t = \{s_1, \ldots, s_t\}$ for $t = 0, \ldots, |S|$.

For $t = 0$, store set S_0 is empty and our MIP model just produces a y solution with random (possibly repeated) configurations.

For each subsequent t, we start with the best solution $H_{t-1} = (x^{t-1}, y^{t-1})$ of the previous iteration and convert it into a solution $H_t = (x^t, y^t)$ of the current subproblem through the refining function `REOPT`(S_t, y^{t-1}). Then we apply the refinement heuristics described in the previous subsection to H_t, reoptimizing one configuration at a time in a circular vein. To reduce computing time, this latter step can be skipped with a certain frequency—except of course in the very last step when $S_t = S$.

Each time a solution $H_t = (x^t, y^t)$ is found, we quickly compute a solution $H = (x, y)$ of the overall problem through function `REOPT`(S, y^t) and update the overall incumbent where all stores are active.

As to store sequence $s_1, \ldots, s_{|S|}$, we have implemented three different strategies to define it. For each store pair a, b, let the dissimilarity index $dist(a, b)$ be defined as the distance between the two demand vectors $(r_{ia} : i \in I)$ and $(r_{ib} : i \in I)$.

- `random`: The sequence is just a random permutation of the integers $1, \ldots, |S|$;
- `most_dissimilar`: We first compute the two most dissimilar stores (a, b), i.e., such that $a < b$ and $dist(a, b)$ is a maximum and initialize $s_1 = a$. Then, for $t = 2, \ldots, |S|$, we define $S' = \{s_1, \ldots, s_{t-1}\}$ and let

$$s_t = \text{argmax}_{a \in S \setminus S'} \{ \min\{dist(a, b) : b \in S'\}$$

- `most_similar`: This is just the same procedure as in the previous item, with max and min operators reverted.

The rational of the `most_dissimilar` policy is to attach first a "core problem" defined by the pairwise most dissimilar stores (those at the beginning of the sequence). The assumption here is that our method performs better in its first iterations (small values of t) as the size of the subproblem is smaller, and we have plenty of configurations to accommodate the initial requests. The "similar stores" are therefore addressed only at a later time, in the hope that the found configurations will work well for them.

A risk with the above policy is that the core problem becomes soon too difficult for our simple refining heuristic, so the current solution in not updated after the

very first iterations. In this respect, policy `most_similar` is more conservative: given for granted that we proceed by subsequently refining a previous solution with one less store, it seems reasonable not to inject too much innovation in a single iteration—as `most_dissimilar` does when it adds a new store with very different demands with respect to the previous ones.

Computational Experiments

The heuristics described in the previous section have been implemented in C language. IBM ILOG CPLEX 12.6 [21] was used as MIP solver. Reported computing times are in CPU seconds of an Intel Xeon E3-1220 V2 quad-core PC with 16GB of RAM. For each run a time limit of 900 s (15 m) was imposed.

Four heuristic methods have been compared: the three construction heuristics `random`, `most_dissimilar` and `most_similar` of section "Working with a Subset of Stores," plus

- `fast_heu`: The fast refinement heuristic of section "Fixing y Variables For All but One Configuration" applied starting from a null solution $x = 0$.

All heuristics are used in a multi-start mode, e.g., after completion they are just restarted from scratch until the time limit is exceeded. At each restart, the internal random seed is not reset; hence, all methods natively using a random permutation (namely, `fast_heu` and `random`) will follow a different search path at each run as the permutations will be different. As to `most_dissimilar` and `most_similar`, after each restart the sequence $s_1, \ldots, s_{|S|}$ is slightly perturbed by five random pair swaps. In addition, after each restart the CPLEX's random seed is changed so as to inject diversification among runs even within the MIP solver.

Due to their heuristic nature, our methods—though deterministic—exhibit a large dependency on the initial conditions, including the random seeds used both within our code and in CPLEX. We therefore repeated several times each experiment, starting with different (internal/CPLEX) random seeds at each run, and also report average figures.

In case all capacities are even (as it is the case in our tesbed), we compute the following trivial lower bound based on constraint (41)

$$LB := \min\{\alpha, \beta\} \cdot \left| \left\{ s \in S : \sum_{i \in I} r_{is} \text{ is odd} \right\} \right| \tag{42}$$

and abort the execution as soon as we find a solution whose value meets the lower bound.

Test Bed

Our test bed coincides with the benchmark proposed in [20] and contains a number of subinstances of a real-world problem (named `AllColor58`) with 58 stores that require 24 (= 6 × 4) different items: T-shirts available in six different sizes and four different colors (black, blue, red, and green). The available box capacities are 4, 6,

8, and 10. Finally, each item has a given overstock limit (0 or 1) for all stores but no understock limits, and the overstock and understock penalties are $\beta = 1$ and $\alpha = 10$, respectively.

Note that our testing environment is identical to that used in [20] (assuming the PC used are substantially equivalent), so our results can fairly be benchmarked against those therein reported.

Comparison Metric

To better compare the performance of our different heuristics, we also make use of an indicator recently proposed by [1, 2] and aimed at measuring the trade-off between the computational effort required to produce a solution and the quality of the solution itself. In particular, let $\tilde{z}_{opt}$ denote the optimal solution value and let $z(t)$ be the value of the best heuristic solution found at a time t. Then, a *primal gap function* p can be computed as

$$p(t) = \begin{cases} 1 & \text{if no incumbent found until time } t \\ \gamma(z(t)) & \text{otherwise} \end{cases} \tag{43}$$

where $\gamma(\cdot) \in [0, 1]$ is the *primal gap*, defined as follows

$$\gamma(z) = \begin{cases} 0 & \text{if } |\tilde{z}_{opt}| = |z| = 0, \\ 1 & \text{if } \tilde{z}_{opt} \cdot z < 0, \\ \frac{z - \tilde{z}_{opt}}{\max\{|\tilde{z}_{opt}|, |z|\}} & \text{otherwise.} \end{cases} \tag{44}$$

Finally, the *primal integral* of a run until time $t_{\max}$ is defined as

$$P(t_{\max}) = \frac{\int_0^{t_{\max}} p(t)\, \mathrm{d}t}{t_{\max}} \tag{45}$$

and is actually used to measure the quality of primal heuristics—the smaller $P(t_{\max})$, the better the expected quality of the incumbent solution if we stopped computation at an arbitrary time before $t_{\max}$.

Computational Results

We addressed the instances provided in [20], with the aim of benchmarking our matheuristics against the methods therein proposed. Results for the easiest cases involving only $NB = 4$ box configurations (namely, instances Black58, Blue58, Red58, and Green58) are not reported as the corresponding MIP model can be solved to proven optimality in less than one second by our solver—thus confirming the figures given in [20].

Tables 2 and 3 report the performance of various heuristics in terms of solution value and time and refer to a single run for each heuristic and for each instance.

Table 2 is taken from [20], where a two-phase hybrid metaheuristic was proposed. In the first phase, the approach uses a memetic algorithm to explore the solution space and builds a pool of interesting box configurations. In the

Table 2 Performance of CPLEX and LNS heuristics from [20]. Single run for each instance. Times in CPU seconds (time limit of 900 s)

		CPLEX		LNS	
Instance	*NB*	Value	Time (s)	Value	Time (s)
BlackBlue10	7	66	7	21	16
BlackBlue58	7	525	43	174	74
AllColor10	14	202	49	89	293
AllColor58	14	1828	273	548	900

Table 3 Performance of matheuristics. Single run for each instance. Times in CPU seconds (time limit of 900 s)

						`most`		`most`	
		`fast_heu`		`random`		`dissimilar`		`similar`	
Instance	LB	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
BlackBlue10	10	10	1	10	2	10	1	10	1
BlackBlue58	58	58	4	58	3	58	2	58	4
AllColor10	6	6	29	17	82	17	734	17	379
AllColor58	42	141	71	273	722	614	105	53	66

second phase, a box-to-store assignment problem is solved to choose a subset of configurations from the pool—and to decide how many boxes of each configuration should be sent to each store. The box-to-store assignment problem is modeled as a (very hard in practice) MIP and heuristically solved either by a commercial solver (CPLEX) or by a sophisticated large-neighborhood search (LNS) approach.

Table 3 reports the performance of our four matheuristics, as well as the lower bound value LB computed through (42)—this latter value turned out to coincide with the optimal value for all instances under consideration in the present subsection.

Comparing Tables 2 and 3 shows that matheuristics outperform the LNS heuristics analyzed in [20]. In particular, `fast_heu` is able to find very good solutions (actually, an optimal one in 3 out of 4 cases) within very short computing times. For the largest instance (AllColor58), however, `most_similar` qualifies as the best heuristic both in terms of quality and speed.

To get more reliable information about the matheuristics' performance, we ran them 100 times for each instance, with different random seeds, and took detailed statistics on each run. Table 4 reports, for each instance and for each heuristic, the average completion time (time), the average time to find its best solution (time_best), the primal integral after 900 s (pint, the lower the better), and the number of provably optimal solutions found (#opt) out of the 100 runs. Note that, for all instances, a solution matching the simple lower bound (42) was eventually found by at least one of our heuristics. The 100-run statistics confirm that `fast_heu` is very effective in all cases, though it is outperformed by `most_similar` for the largest instance AllColor58 with respect to the #opt criterion. The results suggest that a hybrid method running `fast_heu` and `most_similar` (possibly in parallel) qualifies a robust heuristic with a very good performance for all instances.

Table 4 Average performance (out of 100 runs) of our heuristics

Instance	Heuristic	Time (s)	Time_best (s)	Pint	#opt
BlackBlue10	`fast_heu`	1.08	1.08	0.34	100
	`random`	1.44	1.44	0.27	100
	`most_dissimilar`	1.26	1.26	0.25	100
	`most_similar`	1.25	1.25	0.29	100
BlackBlue58	`fast_heu`	4.61	4.61	9.88	100
	`random`	6.40	6.40	10.11	100
	`most_dissimilar`	2.76	2.76	9.13	100
	`most_similar`	5.62	5.62	15.42	100
AllColor10	`fast_heu`	71.82	71.82	3.81	100
	`random`	600.29	304.06	18.63	36
	`most_dissimilar`	704.33	241.12	19.69	27
	`most_similar`	626.15	302.40	20.54	26
AllColor58	`fast_heu`	900.00	332.43	328.20	0
	`random`	874.87	329.59	562.95	2
	`most_dissimilar`	893.48	323.93	545.47	1
	`most_similar`	859.86	287.50	404.29	1

Application 3: Vehicle Routing

In this section we address the NP-hard ***distance-constrained capacitated vehicle routing problem*** (DCVRP) that can be defined as follows. We are given a central depot and a set of $n-1$ customers, which are associated with the nodes of a complete undirected graph $G = (V, E)$ where $|V| = n$, node 1 representing the depot. Each edge $[i, j] \in E$ has an associated finite cost $c_{ij} \geq 0$. Each node $j \in V$ has a request $d_j \geq 0$ ($d_1 = 0$ for depot node 1). Customers need to be served by k cycles (*routes*) passing through the depot, where k is fixed in advance. Each route must have a total duration (computed as the sum of the edge costs in the route) not exceeding a given limit D and can visit a subset S of customers whose total request $\sum_{j \in S} d_j$ does not exceed a given capacity C. The problem then consists of finding a feasible solution covering exactly once all the nodes $v \in V \setminus \{1\}$ and having a minimum overall cost; see, e.g., [3, 31].

We will next outline the refinement matheuristic for DCVRP proposed in [15].

The ASSIGN Neighborhood for TSP

Sarvanov and Doroshko (SD) investigated in [28] the so-called ASSIGN neighborhood for the pure Traveling Salesman Problem (TSP), i.e., for the problem of finding a min-cost Hamiltonian cycle in a graph. Given a certain TSP solution (viewed as node sequence $< v_1 = 1, v_2, \cdots, v_n >$), the neighborhood contains all the $\lfloor n/2 \rfloor !$ TSP solutions that can be obtained by permuting, in any possible way, the nodes in

even position in the original sequence. In other words, any solution $(\psi_1, \psi_2, \cdots, \psi_n)$ in the neighborhood is such that $\psi_i = v_i$ for all odd i. An interesting feature of the neighborhood is that it can be explored exactly in polynomial time, though it contains an exponential number of solutions. Indeed, for any given starting solution, the min-cost TSP solution in the corresponding ASSIGN neighborhood can be found efficiently by solving a min-cost assignment problem on a $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ matrix; see, e.g., [18]. Starting from a given solution, an improving heuristic then consists of exploring the ASSIGN neighborhood according to the following two phases:

- *node extraction*, during which the nodes in even position (w.r.t. the current solution) are removed from the tour, thus leaving an equal number of "free holes" in the sequence;
- *node reinsertion*, during which the removed nodes are reallocated to the available holes in an optimal way by solving a min-sum assignment problem.

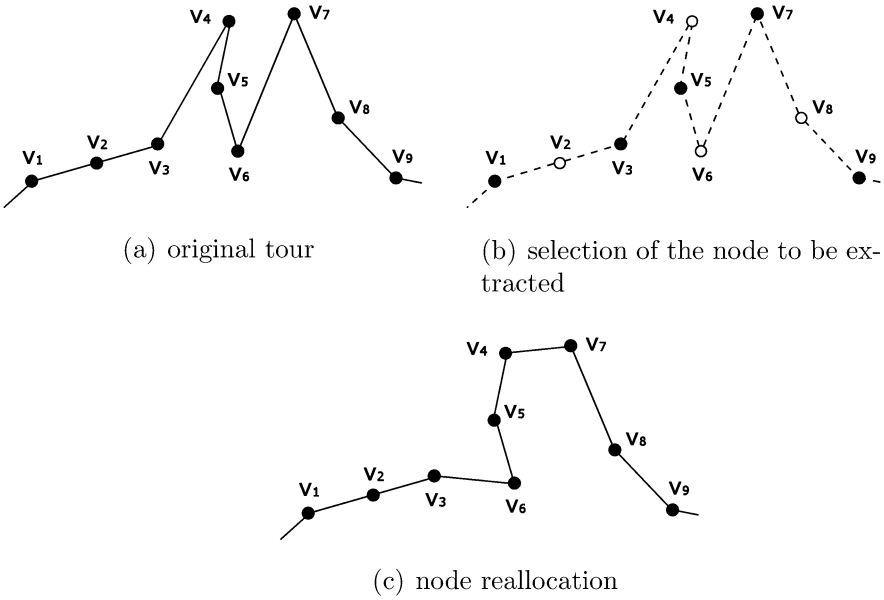
The simple example in Fig. 1 gives an illustration of the kind of "improving moves" involved in the method. The figure draws a part of a tour, corresponding to the node sequence $\langle v_1, v_2, \cdots, v_9 \rangle$. In the node extraction phase, the nodes in even position v_2, v_4, v_6 e v_8 are removed from the sequence, whereas all the other nodes retain their position. In Fig. 1b the black nodes represent the fixed ones, while the holes left by the extracted nodes are represented as white circles. If we use symbol "–" to represent a free hole, the sequence corresponding to Fig. 1b is therefore $\langle v_1, -, v_3, -, v_5, -, v_7, -, v_9 \rangle$. The second step of the procedure, i.e., the optimal node reallocation, is illustrated in Fig. 1c, where nodes v_4 and v_6 swap their position, whereas v_2 and v_8 are reallocated as in the original sequence. This produces the improved part of tour $\langle v_1, v_2, v_3, v_6, v_5, v_4, v_7, v_8, v_9 \rangle$.

In the example, the same final tour could have been constructed by a simple 2-opt move. However, for more realistic cases, the number of possible reallocation is exponential in the number of extracted nodes; hence, the possible reallocation patterns are much more complex and allow, e.g., for a controlled worsening of some parts of the solution which are compensated by large improvement in other parts.

From TSP to DCVRP

One can conjecture that the ASSIGN neighborhood would work well if applied to VRP problems. Indeed, due to the presence of several routes and of the associated route constraints, in VRP problems the node sequence is not the only issue to be considered when constructing a good solution: an equally important aspect of the optimization is to find a balanced distribution of the nodes between the routes. In this respect, heuristic refinement procedures involving complex patterns of node reallocations among the routes likely are quite effective in practice.

We can therefore extend the SD method to DCVRP so as to allow for more powerful move patterns, while generalizing its basic scheme so as to get rid of the



(a) original tour

(b) selection of the node to be extracted

(c) node reallocation

Fig. 1 A simple example of node extraction and reallocation

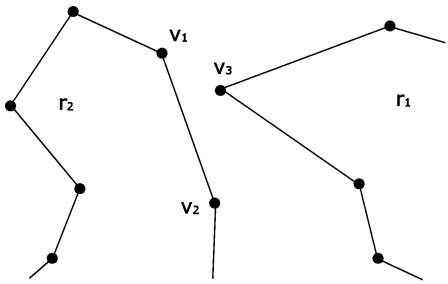


Fig. 2 The assignment of node v_3 to route r_1 is nonoptimal

too simple min-sum assignment model for node reallocation in favor of a more flexible reallocation model based on the (heuristic) solution of a more complex MIP model. The resulting matheuristic will be introduced, step by step, with the help of some illustrative examples.

Let us consider Fig. 2, where a nonoptimal part of a VRP solution is depicted. It is clear that the position of node v_3 is not very clever, in that inserting v_3 between node v_1 and v_2 is likely to produce a better solution (assuming this new solution is not infeasible because of the route constraints). Even if v_3 were an even position node, however, this move would be beyond the possibility of the pure SD method, where the extracted nodes can only be assigned to a hole left free by the removal of another node—while no hole between v_1 e v_2 exists which could accommodate v_3. The example then suggests a first extension of the basic SD method, consisting of removing the 1-1 correspondence between extracted nodes and empty holes. We

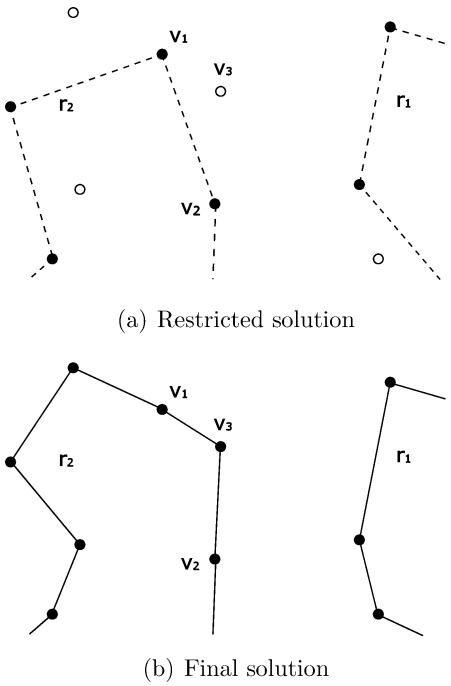


Fig. 3 Improving the solution depicted in Fig. 2

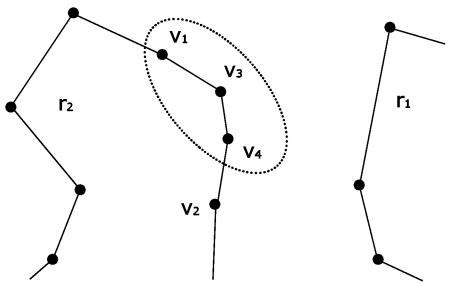


Fig. 4 Removing a sequence of nodes to better reallocate them (possibly in a different order)

therefore consider the concept of *insertion point*: after having extracted the selected nodes, we construct a *restricted solution* through the remaining nodes, obtained from the original one by short-cutting the removed nodes. All the edges in the restricted solution are then viewed as potential insertion points for the extracted nodes. In the example, removing v_3 but not v_1 and v_2 would produce the restricted solution depicted in Fig. 3a, where all dashed edges are possible insertion points for the extracted nodes—this allows the method to produce the solution in Fig. 3b.

A second important extension comes from a more flexible node extraction scheme that allows for the removal of a *sequence* of nodes; see Fig. 4 for an illustration. Once a sequence of nodes has been extracted, one can use a heuristic procedure to generate *new* sequences through the extracted nodes, to be allocated to different insertion points. To be more specific, starting from the extracted node

sequences, one can create new *derived sequences* that combine the extracted nodes in a different way and consider the possibility of assigning each derived sequence to a different insertion point. Of course, one never knows in advance which are the best sequences to be used, so all the (original and derived) sequences should be available when solving the reallocation problem.

The above considerations imply the use of a reallocation model which goes far beyond the scope of the original one, which is based on the solution of an easy min-cost assignment problem. Indeed, the new reallocation model becomes a MIP that receives as input the set of insertion points along with a (typically large) set of node sequences through the extracted nodes and provides an (almost) optimal allocation of at most one sequence to each insertion point, with the constraint that each extracted node has to belong to one of the allocated sequences, while fulfilling the additional constraints on the capacity and distance constraints on the routes. This model will be described in more detail in the next section.

The Overall Matheuristic

Here is a possible implementation of the ideas outlined in the previous section, leading to the so-called *selection, extraction, recombination, and reallocation* (SERR) matheuristic.

(i) (*Initialization*). Apply a fast heuristic method to find a first (possibly infeasible, see below) DCVRP solution.
(ii) (*Selection*). Apply one of the available criteria (to be described later) to determine the nodes to be extracted—the nodes need not be consecutive, and any node subset qualifies as a valid choice.
(iii) (*Extraction*). Extract the nodes selected in the previous step and construct the corresponding restricted DCVRP solution obtained by short-cutting the extracted nodes. All edges in the restricted solution are put in the list $\mathcal{I}$ of the available insertion points.
(iv) (*Recombination*). The node sequences extracted in the previous step (called *basic* in the sequel) are initially stored in a *sequence pool*. Thereafter, heuristic procedures (to be described later) are applied to derive new sequences through the extracted nodes, which are added to the sequence pool. During this phase, dual information derived from the LP relaxation of the reallocation model can be used to find new profitable sequences—the so-called *pricing* step. Each sequence s in the final pool is then associated with a (heuristically determined) subset I_s of the available insertion points in $\mathcal{I}$. For all basic sequences s, we assume that $\mathcal{I}_s$ contains (among others) the *pivot* insertion point associated to s in the original tour, so as to make it feasible to retrieve the original solution by just reallocating each basic sequence to the associated pivot insertion point.
(v) (*Reallocation*). A suitable MIP (to be described later in greater details) is set up and solved heuristically through a general-purpose MIP solver. This model has a binary variable x_{si} for each pair (s, i), where s is a sequence in the

pool and $i \in \mathcal{I}_s$, whose value 1 means that s has to be allocated to i. The constraints in the MIP stipulate that each extracted node has to be covered by exactly one of the selected sequences s, while each insertion point i can be associated to at most one sequence. Further constraints impose the capacity and distance constraints in each route. Once an (almost) optimal MIP solution has been found, the corresponding DCVRP solution is constructed and the current best solution is possibly updated (in which case each route in the new solution is processed by a 3-opt [26] exchange heuristic in the attempt of further improving it).

(vi) (*Termination*). If at least one improved DCVRP solution has been found in the last n iterations, we repeat from step (ii); otherwise the method terminates.

Finding a Starting Solution

Finding a DCVRP solution that is guaranteed to be feasible is an NP-hard problem; hence, we have to content ourselves with the construction of solutions that, in some hard cases, may be infeasible—typically because the total-distance-traveled constraint is violated for some routes. In this case, the overall infeasibility of the starting solution can hopefully be driven to zero by a modification of the SERR recombination model where the capacity and distance constraints are treated in a soft way through the introduction of highly penalized slack variables.

As customary in VRP problems, we assume that each node is assigned a coordinate pair (x, y) giving the geographical position of the corresponding customer/depot in a two-dimensional map.

One option for the initialization of the current solution required at step (i) of the SERR method is to apply the classical two-phase method of Fisher and Jaikumar (FJ) [14]. This method can be implemented in a very natural way in our context in that it is based on a (heuristic) solution of a MIP whose structure is close to that of the reallocation model.

According to computational experience, however, the solution provided by the FJ heuristic is sometimes "too balanced," in the sense that the routes are filled so tightly that leave not enough freedom to the subsequent steps of our SERR procedure. Better results are sometimes obtained starting from a less-optimized solution whose costs significantly exceed the optimal cost as, e.g., the one obtained by using a simplified *SWEEP* method [16].

A second possibility is instead to start from an extremely good solution provided by highly effective (and time-consuming) heuristics or metaheuristics, in the attempt of improving this solution even further.

Node selection criteria

At each execution of step (ii), one of the following selection schemes is applied.

- scheme RANDOM-ALTERNATE: This criterion is akin to the SD one and selects in some randomly selected routes all the nodes in even position, while in the remaining routes the extracted nodes are those in odd position—the position parity being determined by visiting each route in a random direction.

- scheme SCATTERED: Each node had a uniform probability of 50% of being extracted; this scheme allows for the removal of consecutive nodes, i.e., of route subsequences.
- scheme NEIGHBORHOOD: Here one concentrates on a seed node, say v^*, and removes the nodes v with a probability that is inversely proportional to the distance c_{vv^*} of v from v^*.

Schemes RANDOM-ALTERNATE and SCATTERED appear particularly suited to improve the first solutions, whereas the NEIGHBORHOOD scheme seems more appropriate to deal with the solutions available after the first iterations.

Reallocation Model

Given the sequences stored in the pool and the associated insertion points (defined through the heuristics outlined in the next subsection), our aim is to reallocate the sequences so as to find a feasible solution of improved cost (if any). To this end, we need to introduce some additional notation.

Let $\mathcal{F}$ denote the set of the extracted nodes, $\mathcal{S}$ the sequence pool, and $\mathcal{R}$ the set of routes r in the restricted solution. For any sequence $s \in \mathcal{S}$, let $c(s)$ be the sum of the costs of the edges in the sequence, and let $d(s)$ be the sum of the requests d_j associated with the *internal* nodes of s.

For each insertion point $i \in \mathcal{I}$, we define the extra-cost γ_{si} for assigning sequence s (in its best possible orientation) to the insertion point i. For each route $r \in \mathcal{R}$ in the restricted solution, let $\mathcal{I}(r)$ denote the set of the insertion points (i.e., edges) associated with r, while let $\tilde{d}(r)$ and $\tilde{c}(r)$ denote, respectively, the total request and distance computed for route r—still in the restricted tour. As already mentioned, our MIP model is based on the following decision variables:

$$x_{si} = \begin{cases} 1 \text{ if sequence } s \text{ is allocated to the insertion point } i \in \mathcal{I}_s \\ 0 \text{ otherwise} \end{cases} \tag{46}$$

The model then reads:

$$\sum_{r \in \mathcal{R}} \tilde{c}(r) + \min \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}_s} \gamma_{si} x_{si} \tag{47}$$

subject to:

$$\sum_{s \in v} \sum_{i \in \mathcal{I}_s} x_{si} = 1 \quad \forall v \in \mathcal{F} \tag{48}$$

$$\sum_{s \in \mathcal{S}: i \in \mathcal{I}_s} x_{si} \leq 1 \quad \forall i \in \mathcal{I} \tag{49}$$

$$\tilde{d}(r) + \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}_s \cap \mathcal{I}(r)} d(s) x_{si} \leq C \quad \forall r \in \mathcal{R} \tag{50}$$

$$\tilde{c}(r) + \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}_s \cap \mathcal{I}(r)} \gamma_{si} x_{si} \leq D \quad \forall s \in \mathcal{S},\ r \in \mathcal{R} \tag{51}$$

$$0 \leq x_{si} \leq 1 \ \text{ integer} \quad \forall s \in \mathcal{S},\ i \in \mathcal{I}_s \tag{52}$$

The objective function, to be minimized, gives the cost of the final DCVRP solution. Indeed, each objective coefficient gives the MIP cost of an inserted sequence, including the linking cost, minus the cost of the "saved" edge in the restricted solution. Constraints (48) impose that each extracted node belongs to exactly one of the selected sequences, i.e., that it is covered exactly once in the final solution. Note that, in the case of triangular costs, one could replace $=$ by $\geq$ in (48), thus obtaining a typically easier-to-solve MIP having the structure of a set-covering (instead of set-partitioning) problem with side constraints. Constraints (49) avoid a same insertion point be used to allocate two or more sequences. Finally, constraints (50) and (51) impose that each route in the final solution fulfills the capacity and distance restriction, respectively.

In order to avoid to overload the model by an excessive number of variables, a particular attention has to be paid to reduce the number of sequences and, for each sequence, the number of the associated insertion points.

Node Recombination and Construction of Derived Sequences

This is a very crucial step in the SERR method. It consists not just of generating new "good" sequences through the extracted nodes, but it also associates each sequence to a clever set of possible insertion points that can conveniently accommodate it. Therefore, one has two complementary approaches to attack this problem: (a) start from the insertion points and, for each insertion point, try to construct a reasonable number of new sequences which are likely to "fit well" or (b) start from the extracted nodes and try to construct new sequences of small cost, no matter the position of the insertion points. The following two-phase method turned out to be a good strategy in practice.

In the first phase, the sequence pool is initialized by means of the original (basic) sequences, and each of them is associated to its corresponding (pivot) insertion point. This choice guarantees that the current DCVRP solution can be reconstructed by simply selecting all the basic sequences and putting them back in their pivot insertion point. Moreover, when the NEIGHBORHOOD selection scheme is used, a further set of sequences is generated as follows. Let v^* be the extracted seed node, and let $N(v^*)$ contain v^* plus the, say, k closest extracted nodes ($k = 4$ in our implementation). A complete enumerative scheme is applied to generate all the sequences through $N(v^*)$ that are added to the pool. This choice is intended to increase the chances of locally improving the current solution, by exploiting appropriate sequences to reallocate the nodes in $N(v^*)$ in an optimal way.

The second phase is only applied for the NEIGHBORHOOD and SCATTERED selection schemes and corresponds to a pricing loop based on the dual information available after having solved the LP relaxation of the current reallocation model. The reader is again referred to [15] for details.

Examples

Extensive computational results for SERR matheuristic have been reported [15] and are omitted because of space. We only report in Figs. 5 and 6 a plot of the incumbent SERR solution for some instances from the literature. Computing time is given in CPU seconds of an old AMD Athlon XP 2400+ PC with 1 GByte RAM, using ILOG Cplex 8.0 as MIP solver. The figures show that SERR is able to significantly improve the starting solution in the early part of its computation.

Conclusion

Contamination of metaheuristics with mathematical programming leads to the concept of "matheuristics." The result is a general approach to design mathematically sound heuristics. In this chapter we presented the main ideas underlying matheuristics, and used some case studies to illustrate them. For each application, we described the specific problem at hand, the mathematical programming model

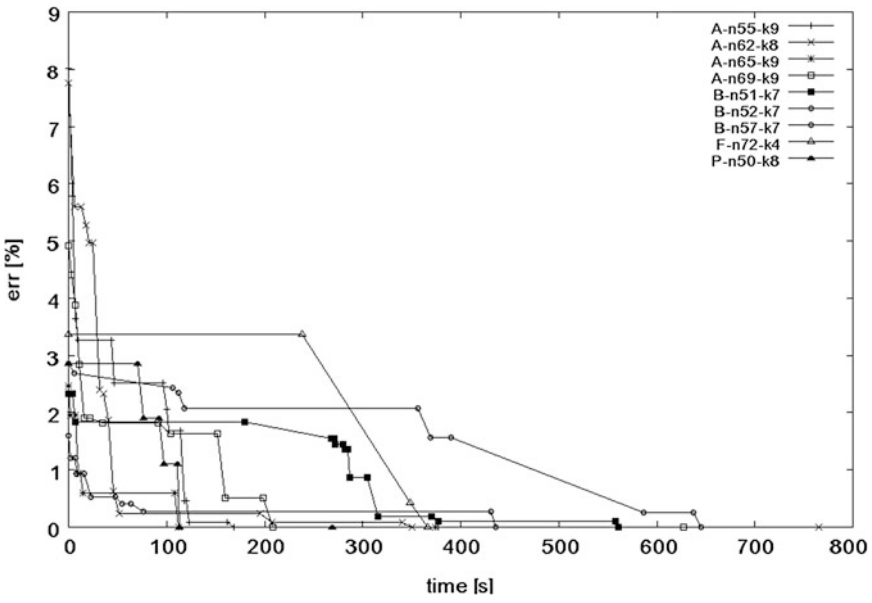


Fig. 5 Time evolution of the SERR solution for various CVRP instances, with FJ [14] initial solution

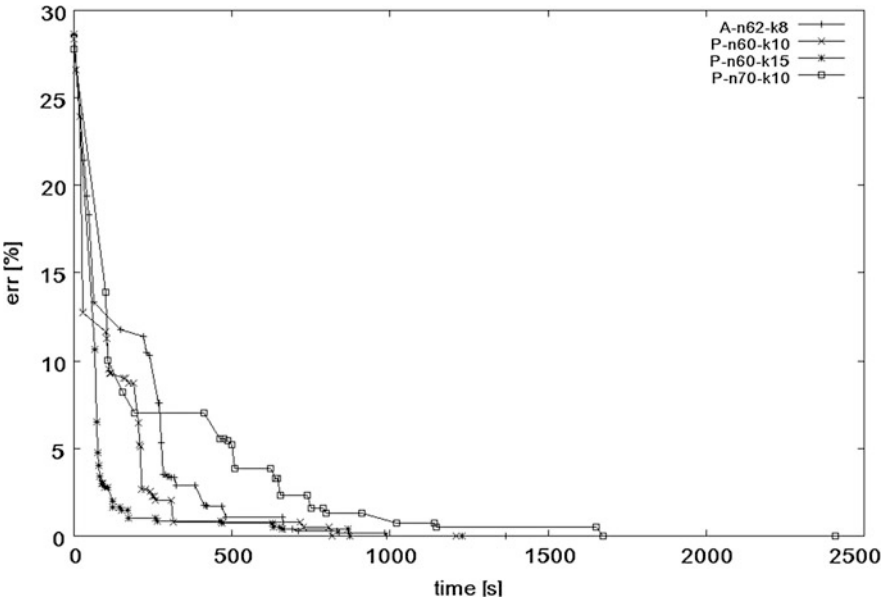


Fig. 6 Time evolution of the SERR solution for various CVRP instances, with SWEEP [16] initial solution

that formalizes it, and the way the model—or a simplification of it—can be used to produce heuristic (as opposed to exact) solutions in an effective way.

Cross-References

- ▸ Adaptive and Multilevel Metaheuristics
- ▸ Constraint-Based Local Search
- ▸ Iterated Local Search

References

1. Achterberg T, Berthold T, Hendel G (2012) Rounding and propagation heuristics for mixed integer programming. In: Operations research proceedings 2011, Zurich, pp 71–76
2. Berthold T (2013) Measuring the impact of primal heuristics. Oper Res Lett 41(6):611–614
3. Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. Combinatorial optimization. Wiley, New York
4. Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Math Program 102:71–90
5. Donovan S (2005) Wind farm optimization. In: Proceedings of the 40th annual ORSNZ conference, Wellington, pp 196–205
6. Fischetti M (2014) Mixed-integer models and algorithms for wind farm layout optimization. Master's thesis, University of Padova. http://tesi.cab.unipd.it/45458/1/tesi_Fischetti.pdf

7. Fischetti M, Lodi A (2003) Local branching. Math Program 98:23–47
8. Fischetti M, Lodi A (2011) Heuristics in mixed integer programming. In: Cochran JJ (ed) Wiley encyclopedia. Volume 8 of operations research and management science. John Wiley & Sons, Hoboken, pp 738–747
9. Fischetti M, Lodi A, Salvagnin D (2010) Just MIP it! In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics. Volume 10 of annals of information systems. Springer, Boston, pp 39–70
10. Fischetti M, Monaci M (2014) Proximity search for 0–1 mixed-integer convex programming. J Heuristics 6(20):709–731
11. Fischetti M, Monaci M (2016) Proximity search heuristics for wind farm optimal layout. J Heuristics 22(4):459–474
12. Fischetti M, Monaci M, Salvagnin D (2012) Three ideas for the quadratic assignment problem. Oper Res 60(4):954–964
13. Fischetti M, Monaci M, Salvagnin D (2015, to appear) Mixed-integer linear programming heuristics for the prepack optimization problem. Discret Optim
14. Fisher ML, Jaikumar R (1981) A generalized assignment heuristic for vehicle routing. Networks 11:109–124
15. De Franceschi R, Fischetti M, Toth P (2006) A new ILP-based refinement heuristic for vehicle routing problems. Math Program 105(2–3):471–499
16. Gillett BE, Miller LR (1974) A heuristic algorithm for the vehicle dispatch problem. Oper Res 22:340–349
17. Glover F (1975) Improved linear integer programming formulations of nonlinear integer problems. Manage Sci 22:455–460
18. Gutin G, Yeo A, Zverovitch A (2007) Exponential neighborhoods and domination analysis for the TSP. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations. Volume 12 of combinatorial optimization. Springer, Boston, pp 223–256
19. Hansen P, Maniezzo V, Voß S (2009) Special issue on mathematical contributions to metaheuristics editorial. J Heuristics 15(3):197–199
20. Hoskins M, Masson R, Melanon GG, Mendoza JE, Meyer C, Rousseau L-M (2014) The prepack optimization problem. In: Simonis H (ed) Integration of AI and OR techniques in constraint programming. Volume 8451 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 136–143
21. IBM ILOG (2014) CPLEX User's Manual
22. Jensen NO (1983) A note on wind generator interaction. Technical report, Riso-M-2411(EN), Riso National Laboratory, Roskilde
23. Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: a survey. Eur J Oper Res 141:241–252
24. Maniezzo V, Stützle T, Voß S (eds) (2010) Matheuristics – hybridizing metaheuristics and mathematical programming. Volume 10 of annals of information systems. Springer, Boston
25. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Chichester
26. Rego C, Glover F (2007) Local search and metaheuristics. In: Gutin G, Punnen A (eds) The traveling salesman problem and its variations. Volume 12 of combinatorial optimization. Springer, Boston, pp 309–368
27. Rothberg E (2007) An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS J Comput 19:534–541
28. Sarvanov VI, Doroshko NN (1981) The approximate solution of the travelling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time (in Russian). In: Software: algorithms and programs. Mathematical Institute of the Belorussian Academy of Sciences, Minsk, pp 11–13
29. Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget J-F (eds) Principles and practice of constraint programming CP98. Volume 1520 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 417–431

30. SIEMENS AG. SWT-2.3-93 Turbine, Technical Specifications. http://www.energy.siemens.com
31. Toth P, Vigo D (2002) An overview of vehicle routing problems. In: The vehicle routing problem. SIAM monographs on discrete mathematics and applications, Philadelphia
32. Xia Y, Yuan YX (2006) A new linearization method for quadratic assignment problem. Optim Methods Softw 21:803–816

Multi-start Methods

6

Rafael Martí, Jose A. Lozano, Alexander Mendiburu, and Leticia Hernando

Contents

Introduction . . . 156
Global Optimization . . . 158
Combinatorial Optimization . . . 159
Estimating the Number of Local Optima . . . 162
  Methods Proposed in the Metaheuristics Arena . . . 163
  Methods Proposed in the Field of Statistics . . . 164
Experiments . . . 165
  Experimental Design . . . 165
  Results . . . 167
Conclusions . . . 173
Cross-References . . . 173
References . . . 174

Abstract

Multi-start procedures were originally conceived as a way to exploit a local or neighborhood search procedure, by simply applying it from multiple random initial solutions. Modern multi-start methods usually incorporate a powerful form

R. Martí (✉)
Statistics and Operations Research Department, University of Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

J. A. Lozano · L. Hernando
Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Donostia, Spain
e-mail: ja.lozano@ehu.es; leticia.hernando@ehu.es

A. Mendiburu
Intelligent Systems Group, Department of Computer Architecture and Technology, University of the Basque Country, Donostia, Spain
e-mail: alexander.mendiburu@ehu.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_1

of diversification in the generation of solutions to help overcome local optimality. Different metaheuristics, such as GRASP or tabu search, have been applied to this end. This survey briefly sketches historical developments that have motivated the field and then focuses on modern contributions that define the current state of the art. Two classical categories of multi-start methods are considered according to their domain of application: global optimization and combinatorial optimization. Additionally, several methods are reviewed to estimate the number of local optima in combinatorial problems. The estimation of this number can help to establish the complexity of a given instance, and also to choose the most convenient neighborhood, which is especially interesting in the context of multi-start methods. Experiments on three well-known combinatorial optimization problems are included to illustrate the local optima estimation techniques.

Keywords
Metaheuristics · Multi-start methods · Local optima estimation

Introduction

Heuristic search procedures that aspire to find globally optimal solutions to hard optimization problems usually require some type of diversification to overcome local optimality. One way to achieve diversification is to restart the procedure from a new solution once a region has been explored. This chapter describes the best known multi-start methods for solving optimization problems.

The first multi-start efforts described in the literature rely on methods used in statistics and calculus as instances of utilizing repeated constructions to produce a preferred candidate, although such methods were not used to solve optimization problems. In our context, early proposals can be found in the domains of heuristic scheduling [9, 34] and the traveling salesman problem [21]. Multi-start global optimization algorithms, on the other hand, were introduced in the 1980s for bound constraint optimization problems. The well-known Monte Carlo random restart approaches simply evaluate the objective function at randomly generated points [38]. The probability of success approaches one as the sample size tends to infinity under very mild assumptions about the objective function. Many algorithms have been proposed that combine the Monte Carlo method with local search procedures [26].

The restart mechanism inherent of a multi-start design has been superimposed on many different search methods. Once a new solution has been generated, a variety of options can be used to improve it, ranging from a simple greedy routine to a complex metaheuristic such as tabu search [18] or GRASP [37]. Note that the former is based on identifying and recording specific types of information (attributes) to exploit in future constructions, while the latter is based on order statistics of sampling and on generating unconnected solutions. Martí et al. [30] presented a detailed description

of these two methodologies within the multi-start framework, in the context of combinatorial optimization.

A basic multi-start procedure simply applies a procedure that can be called `ConstructSolution` multiple times, returning the best solution found over all starts. The constructed solution is typically improved with the `LocalSearch` procedure. These two procedures, also called phases, are alternated until a stopping criterion is satisfied. Then, each global iteration produces a solution (usually a local optima) and the best overall is the algorithm's output. This is illustrated in Algorithm 1 for a minimization problem.

Algorithm 1: Pseudo-code for multi-start algorithm

```
procedure MultiStart
f* ← ∞;
while stopping criterion not satisfied do
    Construct feasible solution:
        S ← ConstructSolution;
        S ← LocalSearch(S);
    if f(S) < f* then
        S* ← S;
        f* ← f(S);
return S*;
```

As it is well known in the heuristic community, the performance of the local search-based algorithms strongly depends on the properties that the neighborhood imposes on the search space. One of the most important properties is the number of local optima. Given an instance and a neighborhood, the estimation of the number of local optima can help to both measure the instance complexity and to choose the most efficient neighborhood. This chapter reviews and tests several methods to estimate the number of local optima in combinatorial optimization problems.

This chapter is focused on studying the different strategies and methods for generating solutions to launch a succession of local searches for global optimum in the context of two domains: global optimization and combinatorial optimization. It is organized as follows. Section "Global Optimization" describes the developments and strategies applied in the context of global (nonlinear) optimization. Section "Combinatorial Optimization" introduces notation for combinatorial optimization and provides descriptions for solution construction procedures and multi-start algorithms. The difficulty of finding the global optima of a combinatorial problem is evaluated through the estimation of its number of local optima in section "Estimating the Number of Local Optima". Specifically, methods proposed in the literature are first reviewed, and then, the associated experiments are described in section "Experiments", where statistical tests are applied to draw significant conclusions. Concluding remarks are given in section "Conclusions".

Global Optimization

As mentioned above, many algorithms have been proposed in the 1980s that combine the Monte Carlo method with local search procedures [25], being the multi-level single linkage the most relevant. In general terms, the probability of success approaches one as the sample size tends to infinity under very mild assumptions about the objective function. The convergence for random restart methods is studied in [38], where the probability distribution used to choose the next starting point can depend on how the search evolves. Some extensions of these methods seek to reduce the number of complete local searches that are performed and increase the probability that they start from points close to the global optimum [31].

From a theoretical point of view, Hu et al. [24] study the combination of the gradient algorithm with random initializations to find a global optimum. Efficacy of parallel processing, choice of the restart probability distribution, and number of restarts are studied for both discrete and continuous models. The authors show that the uniform probability is a good choice for restarting procedures.

Hickernell and Yuan Hickernell and Yuan [23] present a multi-start algorithm for unconstrained global optimization based on *quasirandom samples*. Quasirandom samples are sets of deterministic points, as opposed to random points, that are evenly distributed over a set. The algorithm applies an inexpensive local search (steepest descent) on a set of quasirandom points to concentrate the sample. The sample is reduced replacing worse points with new quasirandom points. Any point that is retained for a certain number of iterations is used to start an efficient complete local search. The algorithm terminates when no new local minimum is found after several iterations. An experimental comparison shows that the method performs favorably with respect to other global optimization procedures.

Tu and Mayne [40] describe a multi-start with a clustering strategy for constrained optimization problems. It is based on the characteristics of nonlinear constrained global optimization problems and extends a strategy previously tested on unconstrained problems. In this study, variations of multi-start with clustering are considered including a simulated annealing procedure for sampling the design domain and a quadratic programming (QP) sub-problem for cluster formation. The strategies are evaluated by solving 18 nonlinear mathematical problems and six engineering design problems. Numerical results show that the solution of a one-step QP sub-problem helps predict possible regions of attraction of local minima and can enhance robustness and effectiveness in identifying local minima without sacrificing efficiency. In comparison with other multi-start techniques, the strategies of this study are superior in terms of the number of local searches performed, the number of minima found, and the number of function evaluations required.

Ugray et al. [41] propose OptQuest/Multi-start or OQMS, a heuristic designed to find global optima for pure and mixed integer nonlinear problems with many constraints and variables, where all problem functions are differentiable with respect to the continuous variables. It uses OptQuest, a commercial implementation of scatter search developed by OptTek Systems, Inc., to provide starting points for any gradient-based local NLP solver. This solver seeks a local solution from a subset

of these points, holding discrete variables fixed. Computational results include 155 smooth NLP and MINLP problems, most with both linear and nonlinear constraints, coded in the GAMS modeling language. Some are quite large for global optimization, with over 100 variables and many constraints. Global solutions to almost all problems are found in a small number of local solver calls, often one or two. An improved version of OQMS is proposed in Ugray et al. [42] in terms of the filters to apply the NLP solver.

More recently, Kaucic [27] presents a multi-start particle swarm optimization algorithm for the global optimization of a function subject to bound constraints. The procedure consists of three main steps. In the initialization phase, an opposition learning strategy is performed to improve the search efficiency. Then, a variant of the adaptive velocity based on the differential operator enhances the optimization ability of the particles. Finally, a re-initialization strategy based on two diversity measures for the swarm is act in order to avoid premature convergence and stagnation. The algorithm is evaluated on a set of 100 global optimization test problems. Comparisons with other global optimization methods show its robustness and effectiveness.

Combinatorial Optimization

Boese et al. [4] analyze relationships among local minima from the perspective of the best local minimum, finding convex structures in the cost surfaces. Based on the results of that study, they propose a multi-start method where starting points for greedy descent are adaptively derived from the best previously found local minima. In the first step, adaptive multi-start (AMS) heuristics generate r random starting solutions and run a greedy descent method from each one to determine a set of corresponding random local minima. In the second step, *adaptive starting solutions* are constructed based on the local minima obtained so far and improved with a greedy descent method. This improvement is applied several times from each adaptive starting solution to yield corresponding *adaptive local minima*. The authors test this method for the traveling salesman problem and obtain significant speedups over previous multi-start implementations. Hagen and Kahng [20] apply this method for the iterative partitioning problem.

Moreno et al. [33] propose a stopping rule for the multi-start method based on a statistical study of the number of iterations needed to find the global optimum. The authors introduce two random variables that together provide a way of estimating the number of global iterations needed to find the global optima: the number of initial solutions generated and the number of objective function evaluations performed to find the global optima. From these measures, the probability that the incumbent solution is the global optimum is evaluated via a normal approximation. Thus, at each global iteration, this value is computed, and if it is greater than a fixed threshold, the algorithm stops; otherwise, a new solution is generated. The authors illustrate the method using the median p-hub problem.

One of the most well-known multi-start methods is the greedy adaptive search procedure (GRASP), which was introduced by Feo and Resende [15]. It was first used to solve set covering problems [14]. Each GRASP iteration consists of constructing a trial solution and then applying a local search procedure to find a local optimum (i.e., the final solution for that iteration). The construction step is an adaptive and iterative process guided by a greedy evaluation function. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is adaptive because the element chosen at any iteration in a construction is a function of previously chosen elements (i.e., the method is adaptive in the sense of updating relevant information from one construction step to the next). At each stage, the next element to be added to the solution is randomly selected from a candidate list of high-quality elements according to the evaluation function. Once a solution has been obtained, it is typically improved by a local search procedure. The improvement phase performs a sequence of moves toward a local optimum solution, which becomes the output of a complete GRASP iteration.

Hagen and Kahng [20] implement an adaptive multi-start method for a VLSI partitioning optimization problem where the objective is to minimize the number of signals sent between components. The method consists of two phases. It first generates a set of random starting points and performs the iterative (local search), thus determining a set of local minimum solutions. Then it constructs adaptive starting points derived from the best local minimum solutions found so far. The authors add a preprocessing cluster module to reduce the size of the problem. The resulting clustering adaptive multi-start (CAMS) method is fast and stable and improves upon previous partitioning results reported in the literature.

Fleurent and Glover [16] propose some adaptive memory search principles to enhance multi-start approaches. The authors introduce a template of a constructive version of tabu search based on both, a set of elite solutions and the intensification strategies based on identifying *strongly determined and consistent variables* according to the following definitions:

- *Strongly determined variables* are those whose values cannot be changed without significantly eroding the objective function value or disrupting the values of other variables.
- A *consistent variable* is defined as one that receives a particular value in a significant portion of good solutions.

The authors propose the inclusion of memory structures within the multi-start framework as it is done with tabu search. Computational experiments for the quadratic assignment problem show that these methods improve significantly over previous multi-start methods like GRASP and random restart that do not incorporate memory-based strategies.

Patterson et al. [35] introduce a multi-start framework called *adaptive reasoning technique* (ART), based on memory structures. The authors implement the short-term and long-term memory functions, proposed in the tabu search framework,

to solve the capacitated minimum spanning tree problem. ART is an iterative, constructive solution procedure that implements learning methodologies on top of memory structures. ART derives its success from being able to learn about and modify the behavior of a primary greedy heuristic. The greedy heuristic is executed repeatedly, and for each new execution, constraints that prohibit certain solution elements from being considered by the greedy heuristic are introduced in a probabilistic fashion. The active constraints are held in a short-term memory, while a long-term memory holds information regarding the constraints that were in the active memory for the best set of solutions.

Glover [17] approaches the multi-start framework from a different perspective. The author views multi-start methods as an extreme version of the *strategic oscillation* approach. Strategic oscillation is a mechanism used in tabu search to allow the process to visit solutions around a "critical boundary," by approaching such a boundary from both sides.

Braysy et al. [5] propose a multi-start local search heuristic for the vehicle routing problem with time windows. The objective in this problem is to design least cost routes for a fleet of identical capacitated vehicles to service geographically scattered customers within pre-specified service time windows. The suggested method uses a two-phase approach. In the first phase, a fast construction heuristic is used to generate several initial solutions. Then, injection trees, an extension of the well-known ejection chain approach [18], are used to reduce the number of routes. In the second phase, two new improvement heuristics, based on CROSS-exchanges [39], are applied for distance minimization. The best solution identified by the algorithm is post-optimized using a threshold accepting post-processor with both intra-route and inter-route improvement heuristics. The resulting hybrid method is shown to be fast, cost-effective, and highly competitive.

Mezmaz et al. [32] hybridize the multi-start framework with a model in which several evolutionary algorithms run simultaneously and cooperate to compute better solutions (called *island model*). They propose a solution method in the context of multi-objective optimization on a computational grid. The authors point out that although the combination of these two models usually provides very effective parallel algorithms, experiments on large-size problem instances are often stopped before convergence is achieved. The full exploitation of the cooperation model needs a large amount of computational resources and the management of the fault tolerance issue.

Under the template of a typical multi-start metaheuristic, [28] propose Meta-RaPS (metaheuristic for randomized priority search), in which several randomization methods and memory mechanisms are present. With the set covering problem (SCP) as the application problem, it is found that these randomization and memory-based methods work well for Meta-RaPS.

Beausoleil et al. [3] consider a multi-objective combinatorial optimization problem called extended knapsack problem. By applying multi-start search and path-relinking their solving method rapidly guides the search toward the most balanced zone of the Pareto-optimal front (the zone in which all the objectives are equally important).

Essafi et al. [13] propose a multi-start ant-based heuristics for a machine line balancing problem. The proposed procedure is a constructive algorithm that assigns operations sequentially to stations. The algorithm builds a feasible solution step by step according to a greedy function that computes the contribution of each unassigned operation to the partial solution under construction based on operational time and weights. The selection of the operation to be added is performed with a roulette wheel mechanism based on the typical ant probability distribution (pheromones of previous assignments). The proposed heuristic is applied to solve a real industry problem.

Dhouib et al. [10] propose a multi-start adaptive threshold accepting algorithm (MS-TA) to find multiple Pareto-optimal solutions for continuous optimization problems. Threshold accepting methods (TAs) are deterministic and faster variants of the well-known simulated annealing algorithms, in which every new move is accepted if it is not much worse than the old one. A multi-start technique is applied in this paper to the TA algorithm to allow more diversifications.

Villegas et al. [43] propose two hybrid algorithms for the single truck and trailer routing problem. The first one is based on GRASP and variable neighborhood descent (VND), while the second one is an evolutionary local search (ELS). In the first one, large tours are constructed with a randomized nearest neighbor method with a restricted candidate list that ignores capacity constraints and trailer-point selection. VND is applied to improve these initial solutions obtained with GRASP. In the second one, a multi-start evolutionary search is applied starting from an initial solution (giant tour). The best solution found is strongly perturbed to obtain different solutions from which the search is restarted. The perturbation is managed by a mutation operator. The results of the computational experiments on a set of 32 randomly generated instances also unveil the robustness of the proposed metaheuristics, all of them achieving gaps to best known solutions of less than 1 % even in the worst case. Among the proposed methods, the multi-start evolutionary local search is more accurate and faster and scales better than the GRASP/VND.

Estimating the Number of Local Optima

As previously seen, multi-start algorithms depend on a neighborhood defined in the search space. Furthermore, the same multi-start algorithm can produce different results in the same instance when it is used with two different neighborhoods. Although there are several characteristics of a neighborhood that influence the behavior of a multi-start algorithm, probably, the most relevant is the number of local optima it generates.

The knowledge about the number of local optima that a neighborhood generates in an instance of a combinatorial optimization problem (COP) can have a high impact on the choice of the multi-start algorithm used to solve the instance. On the first hand, different neighborhoods can generate a dramatically different number of local optima. Of course, as a general rule, the higher the size of the neighbor-

hood, the lower the number of local optima. However, given two neighborhoods of the same size complexity (number of local solutions in the same order of magnitude), the one that generates a lower number of local optima will always be preferred.

Given an instance of a COP and a neighborhood, the exact calculation of the number of local optima is impractical, except for extremely low dimensions ($n \leq 14$). Furthermore, this exact calculation requires, in most of the cases, the exhaustive inspection of every solution in the search space, which makes this approach useful. Therefore, statistical estimation methods are required to have an approximation of the number of local optima of a landscape. Note, however, that the representation of such number is in itself an additional challenge. When the dimension of a problem is high (for instance, $n \geq 200$) and the size of the search space is exponential in n, the number of local optima is usually so high that cannot be accurately represented in the computer. The alternative choice to represent the proportion of solutions that are local optima deals with similar issues as this number is too small to be accurately represented. Therefore, the estimation of the number of local optima of an instance of a COP is only plausible for moderate values of the problem size.

In spite of the useful information that the knowledge of the number of local optima can provide in order to choose the best algorithm for solving a COP instance, there have not been many proposals in the literature. This can probably be due to the difficulty of the estimation problem. These proposals are mainly divided into three groups (a good review and a comparison of several proposals can be found in [22]). A first group is based on the hypothesis that the instances have been generated uniformly at random [1, 2, 19]. The second group mainly includes proposals that have been developed in the metaheuristics community. Finally, a third group includes those that were not designed with the objective of estimating the number of local optima, but were adapted to this problem.

In order to estimate the number of local optima, all the methods share some steps of the process. All of them obtain uniformly at random a sample of size M from the search space. Departing from these solutions and applying a greedy local search algorithm, they finally obtain a sample of local optima (notice that in this sample some local optima could be repeated several times). In addition to that, the concept of basin of attraction is presented in all the estimation methods. Basically, a solution belongs to the basin of attraction of a local optimum if, after applying a greedy local search to that solution, it finishes in the corresponding local optimum. Notice that, in case of injective functions, the basins of attraction represent a partition of the search space.

Methods Proposed in the Metaheuristics Arena

Most of the methods proposed in the metaheuristics field are mainly due to [11, 12, 36]. They basically developed two kinds of methods: a first group where they use confidence intervals to provide lower bounds (with high probability) for the number of local optima and a second group using bias-correcting methods.

In the first group, by assuming that the sizes of the basins of attraction of all the local optima are the same, they manage to get lower bounds by means of several methods:

- *First repetition time*: A random solution is taken from the search space, and a greedy local search algorithm is applied to reach a local optimum. This process continues until a local optimum is repeated for the first time. This number of initial solutions needed until a local optimum is seen twice is the value used to make the estimation. Note that in this method the sample size is not fixed.
- *Maximal first repetition time*: This method is similar to the previous one, except that the sample size is fixed beforehand and the maximal first repetition time (the maximum number of solutions between the first reoccurrence of a local optimum) is used to carry out the estimation.
- *Schnabel census*: In order to make the estimation, the authors take into account in this case the probability that r different local optima are discovered from a sample of size M.

The main drawback of the previous methods is that they are strongly biased by the assumption that all the local optima have the same size of basin of attraction. In order to overcome this bias, a couple of very well-known methods in the statistical literature are proposed: Jackknife and Bootstrap. These are nonparametric methods that, departing from a biased estimation, try to correct the bias:

- *Jackknife method*: It consists in calculating M new estimations by leaving each time one of the current solutions out from the original sample. These new M estimations are used to modify the original estimation decreasing its bias.
- *Bootstrap method*: This resampling technique consists in obtaining several new samples from the original sample. Each sample has the same size as the original one and is obtained by uniformly at random sampling solutions with replacement from the original sample. On the contrary to the previous method, it assumes a probabilistic model in the basins of attraction to make a new estimation for each resampling.

Methods Proposed in the Field of Statistics

The community working in metaheuristics quickly realized that the methods used by biologists and ecologists to calculate the number of species in a population could be easily adapted to the problem of calculating the number of local optima. Particularly four nonparametric methods [6–8] were adapted to estimate the number of local optima of instances of COPs [22]. Although they are nonparametric methods, they are based on particular sampling models. An important consideration is that all these methods assume an infinite population, in our case an infinite number of local solutions. This assumption does not impose a big constraint because of the large cardinality of the search space of the COPs:

- *Chao1984 method*: It is based on multinomial sampling. The estimator is the result of adding to the number of local optima obtained from the sample a term that depends only on the number of local optima seen once and twice in the sample.
- *Chao&Lee methods*: These two methods are also based on multinomial sampling. They include the idea of sample coverage (sum of the proportions of the basins of attraction of the local optima observed in the sample). They distinguish between the local optima observed many times in the sample and those observed a few number of times in the sample. So the estimators are the sum of the number of local optima observed many times and some terms that depend on the sample coverage and the number of local optima found a few times, trying to compensate for the local optima that have small basins of attraction.
- *Chao&Bunge method*: It is based on a mixed Poisson sampling model and is closely related to the previous estimators. This method bases the estimation of unobserved local optima on a formula that depends on the expected proportion of duplicates in the sample. It also makes the distinction between the local optima observed many times in the sample and those observed a few number of times in the sample.

Experiments

The accuracy of the different estimators presented in the previous section has been tested on instances of three different common problems: permutation flowshop scheduling problem (PFSP), linear ordering problem (LOP), and quadratic assignment problem (QAP). Using these datasets, the methods can be tested over a wide set of instances with different characteristics. The number of local optima for different neighborhoods is already known for some of these instances, which allows to evaluate the accuracy of the different estimations. A comparison of the different methods is given, together with recommendations of the methods that provide the best estimations. Based on the results found in [22], the three first methods presented in the previous section are removed for the analysis, that is, first repetition time, maximal first repetition time, and Schnabel census.

Experimental Design

The instances used in the experiments are taken from three well-known benchmarks: 5 instances of the PFSP obtained from the Taillard's benchmark, 5 instances of the LOP taken from the LOLIB benchmark [29], and 5 instances of the QAP chosen from the QAPLIB.

The flowshop scheduling problem can be stated as follows: there are b jobs to be scheduled in c machines. A job consists of c operations and the j-th operation of each job must be processed on machine j for a specific processing time without interruption. Jobs are processed in the same order on different machines. The

objective of the PFSP is to find the order in which the jobs have to be scheduled on the machines, minimizing the total flow time.

In the LOP, given a matrix $B = [b_{ij}]_{nxn}$ of numerical entries, a simultaneous permutation of the rows and columns of B has to be found, such that the sum of the entries above the main diagonal is maximized (or equivalently, the sum of the entries below the main diagonal is minimized).

The QAP is the problem of allocating a set of facilities to a set of locations, with a cost function associated to the distance and the flow between the facilities. The objective is to assign each facility to a location such that the total cost is minimized, given two $n \times n$ input matrices with real values $\mathbf{H} = [h_{ij}]$ and $\mathbf{D} = [d_{kl}]$, where h_{ij} is the flow between facility i and facility j and d_{kl} is the distance between location k and location l.

For the three problems, instances of permutation size $n = 13$ are considered, so the search space is of size $|\Omega| = 13! \approx 6.23 \cdot 10^9$. In the case of the PFSP, instances have 13 jobs and 5 machines. Notice that a first step consists of evaluating all the solutions of the search space to know in advance the exact number of local optima. Therefore, working with higher permutation sizes becomes impracticable.

The local optima of the instances have been calculated according two commonly used operators: the 2-exchange and the insert neighborhoods. Denoting by $\boldsymbol{\pi} = \pi_1\pi_2 \ldots \pi_n$ a solution (permutation of size n) of the search space, the 2-exchange neighborhood considers that two solutions are neighbors if one is generated by swapping two elements of the other:

$$N_S(\pi_1\pi_2 \ldots \pi_n) = \left\{(\pi'_1\pi'_2 \ldots \pi'_n) \mid \pi'_k = \pi_k, \forall k \neq i, j, \pi'_i = \pi_j, \pi'_j = \pi_i, i \neq j\right\}.$$

Two solutions are neighbors under the insert neighborhood (N_I) if one is the result of moving an element of the other one to a different position:

$$N_I(\pi_1\pi_2 \ldots \pi_n) = \left\{(\pi'_1\pi'_2 \ldots \pi'_n) \mid \pi'_k = \pi_k, \forall k<i \text{ and } \forall k>j, \pi'_k = \pi_{k+1}, \forall i \leq k<j, \pi'_j = \pi_i\right\}$$
$$\cup \left\{(\pi'_1\pi'_2 \ldots \pi'_n) \mid \pi'_k = \pi_k, \forall k<i \text{ and } \forall k>j, \pi'_i = \pi_j, \pi'_k = \pi_{k-1}, \forall i<k \leq j\right\}.$$

The different methods for estimating the number of local optima are applied to the 5 instances of each of the considered problems (PFSP, LOP, and QAP) using both neighborhoods (N_S and N_I). So, a total of 30 different landscapes are considered. These two neighborhoods create different situations for the estimations. As the insert neighborhood explores at each step more solutions than the 2-exchange neighborhood, the number of local optima obtained when using the first neighborhood is probabilistically lower than when assuming the second one. However, as will be shown, due to the intrinsic nature of the QAP, given an instance of this problem, the number of local optima when using the insert neighborhood is much higher than when considering the 2-exchange neighborhood.

The estimation methods *Jackknife*, *Bootstrap*, *Chao1984*, *Chao&Lee1*, *Chao&Lee2*, and *Chao&Bunge* are applied to each of the landscapes 10 times,

and the average estimation value of the 10 repetitions is reported. Methods are applied with different sample sizes: $M = 100, 500, 1000, 5000$.

Results

Our aim is to compare the accuracy of the different methods and their relation to the sample size. Results are analyzed according to the type of problem and the neighborhood used, and the effect of the sample size on the methods using different sample sizes is studied. Table 1 reports the average estimations (of the 10 repetitions) obtained with the different methods and the different sample sizes for all the instances of the three problems when using both neighborhoods. It also indicates the real number of local optima of each instance. v denotes the real number of local optima of the instances with the corresponding neighborhood.

Table 1 shows that the method that provides the best results more times is *Chao&Lee2*. In fact, in general, the estimations, given by those methods that come from the field of statistics, provide better results than *Bootstrap* and *Jackknife*, above all, when the number of local optima is considerably high.

A statistical analysis was carried out to compare the estimations obtained with the different methods and three different scenarios. In the first one, the estimations are grouped in three sets according to the type of problem: PFSP, LOP, or QAP. The second scenario considers two different sets that contain the estimations of the instances when using the neighborhoods N_S or N_I. In the last scenario, the estimations are grouped in four sets, according to the parameter $M = 100, 500, 1000, 5000$. A nonparametric Friedman's test with level of significance $\alpha = 0.05$ is used to test if there are statistical significant differences among the estimations provided by the six methods in the different scenarios. It provides a ranking of the methods while also giving an average rank value for each method. Statistical differences are found in all the cases, and then a post hoc test is applied to perform all pairwise comparisons. Particularly, the Holm's procedure is applied fixing the level of significance to $\alpha = 0.05$.

Table 2 shows the ranking obtained for the methods with the Friedman's test in the first scenario, that is, for the instances of the PFSP (first pair of columns), LOP (the pair of columns in the middle), and QAP (last pair of columns). The lower the rank, the worse the performance of the method is. So, the methods are ordered from best to worst. Therefore, the best methods in the three groups are *Chao&Lee2*, *Chao&Lee1*, and *Chao1984*. However, pairwise significant differences are not found between *Chao&Lee1* and *Chao1984*, *Chao1984* and *Bootstrap*, and *Bootstrap* and *Chao&Bunge*, for the PFSP instances, and between *Bootstrap* and *Chao&Bunge*, *Bootstrap* and *Jackknife*, and *Jackknife* and *Chao&Bunge*, for the LOP and the QAP instances.

In Table 3, the ranking for the methods is shown, but this time for the second scenario, that is, when grouping the estimations for the instances using the 2-exchange neighborhood N_S (the first pair of columns), and the insert neighborhood N_I (the last pair of columns). In the first case, the Holm's procedure states that

Table 1 Mean of the estimations obtained for all the instances of the PFSP, LOP, and QAP, under the 2-exchange and the insert neighborhoods

		Method	M=100	M=500	M=1000	M=5000	M=100	M=500	M=1000	**M=5000**
PFSP	**2-exchange**		Instance 1. $v = 2386$				Instance 2. $v = 8194$			
		Jackknife	158.20	517.30	744.80	1371.20	185.50	709.50	1139.10	2733.60
		Bootstrap	155.90	562.80	862.00	1786.30	173.20	706.50	1199.00	3281.70
		Chao1984	400.80	692.80	874.70	1405.30	1148.60	1438.50	1751.30	3105.50
		Chao&Lee1	397.30	725.70	932.00	1401.90	1196.50	1544.50	1884.70	3223.90
		Chao&Lee2	**482.40**	**917.90**	1185.30	1565.30	**1302.50**	**2132.40**	**2605.80**	**4020.70**
		Chao&Bunge	236.50	822.80	**2304.40**	**1901.50**	494.40	390.50	660.30	14054.20
	Insert		Instance 1. $v = 134$				Instance 2. $v = 923$			
		Jackknife	54.30	95.40	104.00	173.40	125.10	317.30	435.90	**955.40**
		Bootstrap	66.60	**124.30**	**141.40**	231.20	129.00	374.60	535.90	1253.50
		Chao1984	62.40	100.10	103.20	170.60	256.60	389.90	482.80	997.40
		Chao&Lee1	65.80	95.00	100.20	**165.80**	247.70	408.90	488.20	978.10
		Chao&Lee2	**84.00**	103.90	104.60	169.60	**371.80**	545.50	583.90	1092.90
		Chao&Bunge	250.80	121.30	109.20	171.80	112.20	**1202.90**	**1108.50**	1341.40
LOP	**2-exchange**		Instance 1. $v = 9969$				Instance 2. $v = 3355$			
		Jackknife	183.00	732.10	1185.50	3256.30	176.20	642.40	980.70	1946.70
		Bootstrap	171.30	723.00	1237.20	3754.40	167.50	662.00	1073.30	2458.60
		Chao1984	923.50	1633.00	1939.20	4073.00	810.70	1001.10	1225.40	2007.00
		Chao&Lee1	935.30	1612.50	2043.00	4580.20	740.40	1003.10	1319.30	2033.40
		Chao&Lee2	**987.30**	**2104.70**	**2860.80**	**6487.20**	**904.80**	1185.50	1677.20	2308.20
		Chao&Bunge	349.80	399.90	680.10	2040.20	144.30	**3188.10**	**2642.10**	**2950.40**
	Insert		Instance 1. $v = 1113$				Instance 2. $v = 60$			
		Jackknife	124.40	335.20	472.70	1093.50	51.90	**60.50**	61.50	137.10
		Bootstrap	128.80	391.80	578.00	1431.90	66.90	82.00	77.70	169.70
		Chao1984	224.00	396.30	528.70	**1121.70**	53.20	57.70	60.40	133.70
		Chao&Lee1	235.60	445.10	556.90	1137.30	52.30	57.60	59.60	**133.40**
		Chao&Lee2	351.70	595.80	**693.60**	1285.00	**57.30**	57.70	**59.70**	133.60
		Chao&Bunge	**1111.70**	**859.80**	8551.50	1630.70	67.70	57.70	**59.70**	133.60
QAP	**2-exchange**		Instance 1. $v = 18720$				Instance 2. $v = 3472$			
		Jackknife	197.10	958.80	1811.60	6594.90	171.70	598.50	910.60	1889.80
		Bootstrap	180.00	873.60	1678.60	6740.80	164.80	628.60	1009.20	2369.00
		Chao1984	2000.70	9847.80	8007.20	9818.00	609.40	870.40	1191.10	2009.20
		Chao&Lee1	**2025.00**	9914.80	7973.80	9750.90	616.20	934.40	1247.50	2026.20
		Chao&Lee2	**2025.00**	**10323.30**	**8296.00**	10683.60	**679.90**	1188.50	1652.60	2351.40
		Chao&Bunge	1050.00	3752.20	128363.10	**17782.40**	260.10	**1330.50**	**4564.80**	**3386.60**
	Insert		Instance 1. $v = 4615326$				Instance 2. $v = 1501175$			
		Jackknife	198.70	995.70	1991.60	9821.40	198.70	995.30	1988.20	9795.60
		Bootstrap	180.80	892.70	1787.00	8851.40	180.80	892.50	1785.20	8837.20
		Chao1984	580.10	93351.20	210869.40	226925.80	580.10	45585.50	196680.90	199572.40
		Chao&Lee1	**585.00**	**93575.00**	**211316.70**	226706.30	**585.00**	41316.70	179342.60	199911.00
		Chao&Lee2	**585.00**	**93575.00**	**211316.70**	**245043.10**	**585.00**	**45980.20**	**198268.50**	**213030.90**
		Chao&Bunge	340.00	46925.00	105933.30	26271.90	340.00	16916.30	74087.70	34635.20

M=100	**M=500**	M=1000	**M=5000**	**M=100**	**M=500**	M=1000	**M=5000**	**M=100**	**M=500**	M=1000	**M=5000**
Instance 3. $v = 1997$				Instance 4. $v = 5119$				Instance 5. $v = 192$			
142.70	397.50	590.40	1098.50	167.40	575.20	886.60	1920.50	77.10	131.80	153.00	**183.10**
143.60	452.90	688.60	1428.00	161.40	609.20	982.40	2376.60	90.70	170.80	**208.80**	240.10
331.70	517.30	774.30	1153.30	544.30	843.40	1161.60	2122.60	94.00	136.20	151.60	178.30
340.50	546.50	769.10	1146.60	510.90	889.20	1242.60	2194.20	90.10	133.30	147.70	175.00
519.70	**737.80**	1030.10	1301.30	587.60	1145.70	1679.00	2689.10	108.80	146.70	155.00	177.60
204.60	3789.80	**2399.70**	**1695.90**	**793.20**	**2103.80**	**1887.90**	**6446.60**	**258.10**	**173.30**	162.10	178.60
Instance 3. $v = 506$				Instance 4. $v = 190$				Instance 5. $v = 14$			
87.50	208.80	265.10	**520.40**	72.10	127.80	150.60	300.30	12.70	13.30	**14.20**	**14.00**
99.40	252.10	339.90	703.20	85.30	163.00	**199.70**	415.00	16.00	14.50	15.50	14.30
138.40	253.80	277.20	536.60	94.60	138.10	154.10	290.90	11.60	12.90	13.10	**14.00**
128.30	263.50	285.70	522.30	83.50	133.90	147.10	**287.10**	12.30	13.40	**14.20**	**14.00**
178.10	350.40	331.20	570.80	**101.40**	152.20	157.00	296.60	12.50	13.70	14.70	**14.00**
74.70	**367.70**	**498.50**	652.70	281.00	**212.60**	169.80	303.80	**12.80**	**14.10**	15.10	**14.00**
Instance 3. $v = 4732$				Instance 4. $v = 3227$				Instance 5. $v = 6810$			
172.80	615.90	921.00	2068.10	136.30	394.10	587.80	1274.40	185.00	746.60	1248.20	2995.20
164.80	638.90	1019.00	2539.90	138.50	443.90	685.60	1607.50	172.70	734.40	1291.00	3578.20
842.70	1013.50	1211.00	2321.60	259.90	562.90	765.70	1428.50	1884.60	1523.20	1970.00	3297.30
813.00	1070.80	1299.90	2369.00	257.40	615.10	792.10	1463.30	1757.50	1589.10	2157.70	3485.10
1068.70	**1450.40**	**1744.20**	**2914.70**	**328.00**	**935.00**	1088.80	1785.30	**1966.40**	**1960.20**	2868.70	**4263.30**
156.90	27501.80	558.90	7544.60	121.70	714.80	**2946.60**	**4654.60**	666.80	4991.50	**7635.90**	10894.20
Instance 3. $v = 12$				Instance 4. $v = 9$				Instance 5. $v = 67$			
9.60	12.40	12.60	19.30	7.80	9.40	**9.00**	20.60	49.10	63.80	**60.90**	108.90
11.80	13.40	14.40	20.20	9.70	13.00	10.20	23.00	**64.00**	86.00	78.70	128.30
8.80	**11.70**	**12.10**	**18.20**	7.20	**9.00**	8.90	**20.20**	51.00	**65.40**	59.90	107.30
9.10	13.20	12.60	20.10	**9.20**	9.20	8.90	20.30	47.80	60.40	58.70	**106.20**
9.30	14.20	12.60	22.30	10.40	9.40	8.90	20.40	51.30	61.30	59.10	106.60
9.20	14.80	12.40	19.00	8.20	9.40	8.90	20.50	57.00	61.80	59.10	106.80
Instance 3. $v = 62$				Instance 4. $v = 173568$				Instance 5. $v = 563$			
27.40	42.30	48.80	57.40	198.70	992.00	1965.20	9280.00	94.40	204.50	264.00	368.00
33.00	**54.30**	**66.00**	73.10	180.90	891.60	1771.50	8532.00	105.20	254.10	338.80	**494.60**
39.90	48.70	47.50	**58.00**	580.10	48561.00	49112.70	51951.40	139.00	227.10	286.40	373.90
32.30	44.10	49.50	54.90	**585.00**	**48783.40**	**49601.10**	51848.80	157.80	224.70	278.60	351.90
39.80	50.90	53.30	56.00	**585.00**	**48783.40**	**49601.10**	53432.60	257.00	266.60	320.90	365.80
55.20	130.20	66.70	56.50	340.00	24529.10	25050.50	**174841.50**	**261.30**	**521.50**	**457.40**	378.10
Instance 3. $v = 579275$				Instance 4. $v = 6712090$				Instance 5. $v = 83240$			
199.00	987.00	1947.40	9108.50	**199.00**	997.70	1997.30	9955.30	193.70	884.40	1616.80	5935.70
181.00	889.90	1758.20	8408.30	181.00	894.50	1791.90	8929.10	179.00	829.50	1548.00	6141.30
100.00	43655.70	33049.70	54343.30	100.00	25200.50	225000.60	938528.00	2902.10	3923.60	4892.70	10856.90
100.00	38924.50	32904.70	58780.20	100.00	**25275.00**	**225250.00**	933604.60	**2950.10**	3925.40	5521.50	12057.10
100.00	**45088.10**	**36647.70**	**83214.20**	100.00	**25275.00**	**225250.00**	**976729.70**	**2950.10**	**4803.90**	**8319.40**	**18835.30**
100.00	12960.90	8107.80	4677.70	100.00	12850.00	113000.00	304803.40	1499.90	458.70	857.40	3354.70

Table 2 Average rankings of the methods according to the type of problem

PFSP		LOP		QAP	
Method	Ranking	Method	Ranking	Method	Ranking
Chao&Lee2	4.85	*Chao&Lee2*	4.83	*Chao&Lee2*	4.77
Chao&Lee1	3.74	*Chao&Lee1*	4.08	*Chao&Lee1*	3.97
Chao1984	3.51	*Chao1984*	3.78	*Chao1984*	3.60
Bootstrap	3.37	*Chao&Bunge*	2.82	*Jackknife*	2.97
Chao&Bunge	3.05	*Jackknife*	2.78	*Bootstrap*	2.88
Jackknife	2.47	*Bootstrap*	2.69	*Chao&Bunge*	2.81

Table 3 Average rankings of the methods according to the neighborhood

N_S		N_I	
Method	Ranking	Method	Ranking
Chao&Lee2	5.22	*Chao&Lee2*	4.41
Chao&Lee1	3.94	*Chao&Lee1*	3.92
Chao1984	3.61	*Chao1984*	3.66
Bootstrap	3.12	*Jackknife*	3.26
Chao&Bunge	2.88	*Chao&Bunge*	2.91
Jackknife	2.23	*Bootstrap*	2.84

Table 4 Average rankings of the methods according to the sample size

$M = 100$		$M = 500$		$M = 1000$		$M = 5000$	
Method	Ranking	Method	Ranking	Method	Ranking	Method	Ranking
Chao&Lee2	4.72	*Chao&Lee2*	5.01	*Chao&Lee2*	5.01	*Chao&Lee2*	4.54
Chao&Lee1	3.87	*Chao&Lee1*	4.06	*Chao&Lee1*	3.97	*Chao&Lee1*	3.83
Chao1984	3.65	*Chao1984*	3.63	*Chao1984*	3.57	*Chao1984*	3.67
Bootstrap	3.19	*Bootstrap*	2.93	*Bootstrap*	3.00	*Chao&Bunge*	3.48
Jackknife	3.12	*Chao&Bunge*	2.69	*Chao&Bunge*	2.93	*Bootstrap*	2.81
Chao&Bunge	2.46	*Jackknife*	2.67	*Jackknife*	2.52	*Jackknife*	2.67

significant differences exist among each pair of methods. Nevertheless, for the second neighborhood, significant differences are not found between *Chao&Lee1* and *Chao1984* and between *Chao&Bunge* and *Bootstrap*. It is observed in this table that, also in this scenario, the best estimations are provided by *Chao&Lee2*, *Chao&Lee1*, and *Chao1984*.

Finally, Table 4 shows the ranking obtained for the methods when the estimations are separated according to the sample size used by the methods $M = 100, 500, 1000, 5000$. Again, the three best methods are *Chao&Lee2*, *Chao&Lee1*, and *Chao1984*. Significant differences are not found between *Chao&Lee1* and *Chao1984* and between *Bootstrap* and *Jackknife*, when using $M = 100$. For $M = 500$ there are not significant differences among Bootstrap, *Chao&Bunge*, and *Jackknife*, while significant differences are not found between *Bootstrap* and *Chao&Bunge* when $M = 1000$. Finally, when taking $M = 5000$, there are

not significant differences among *Chao&Lee1*, *Chao1984*, and *Chao&Bunge* and between *Jackknife* and *Bootstrap*.

Although the statistical analysis gives a global picture of the performance of the methods, and *Chao&Lee2* provides the best solutions with significant differences between this method and the remaining methods, it is also relevant to consider some aspects that are not reflected in the hypothesis tests. The closeness of the estimations provided by the methods to the value to estimate is the most important factor. Obviously, there are methods that estimate better than others, but it does not mean that the estimations provided by the best methods are close enough to the real value. In order to check if the methods provide useful estimations, and so as to add more information to that deduced from Table 1, the average errors of the estimations with respect to the real number of local optima are calculated. Tables 5, 6, and 7 show the average relative errors and the standard deviations (in brackets) grouped by the neighborhood used and the sample size, for all the instances of the PFSP, LOP, and QAP, respectively.

A general conclusion drawn from Tables 5, 6, and 7 is that, in general terms, the estimations improve as the sample size grows, and they are worse as the number of local optima increases. Specifically, as shown in Table 1, the instances of the LOP with the insert neighborhood have a low number of local optima, and Table 6 shows that the average error is lower than in the rest of the cases. Moreover, the instances of the QAP with the insert neighborhood have a high number of local optima, and the errors obtained for those instances (in Table 7) are higher than for the rest. It is remarkable that, for sample size $M = 100$ and $M = 500$, the estimations are far from the real value. However, there is one method which behaves different and does not follow the general lines. This is the *Chao&Bunge* method. This estimation method does not necessarily improve as the sample size grows,

Table 5 Average relative errors and standard deviations (in brackets) of the estimations provided by the different methods for the instances of the PFSP, according to the neighborhood and the sample size M

		$M = 100$	$M = 500$	$M = 1000$	$M = 5000$
2-exchange	*Jackknife*	0.88 (0.14)	0.74 (0.22)	0.66 (0.24)	0.44 (0.22)
	Bootstrap	0.87 (0.17)	0.69 (0.30)	0.61 (0.28)	0.38 (0.15)
	Chao1984	0.79 (0.15)	0.68 (0.21)	0.60 (0.22)	0.42 (0.20)
	Chao&Lee1	0.79 (0.14)	0.67 (0.20)	0.60 (0.20)	0.42 (0.19)
	Chao&Lee2	**0.74 (0.18)**	**0.60 (0.20)**	**0.51 (0.19)**	0.35 (0.16)
	Chao&Bunge	0.89 (0.26)	1.06 (2.23)	0.84 (1.12)	**0.28 (0.38)**
Insert	*Jackknife*	0.60 (0.28)	0.39 (0.22)	0.30 (0.18)	0.19 (0.23)
	Bootstrap	0.58 (0.26)	0.28 (0.23)	0.21 (0.16)	0.54 (0.40)
	Chao1984	0.53 (0.23)	0.34 (0.19)	0.28 (0.17)	0.19 (0.21)
	Chao&Lee1	0.54 (0.24)	0.34 (0.18)	0.29 (0.15)	**0.17 (0.19)**
	Chao&Lee2	**0.45 (0.22)**	**0.25 (0.15)**	0.24 (0.14)	0.23 (0.20)
	Chao&Bunge	0.92 (1.40)	0.50 (0.78)	**0.20 (0.18)**	0.32 (0.24)

Table 6 Average relative errors and standard deviations (in brackets) of the estimations provided by the different methods for the instances of the LOP, according to the neighborhood and the sample size M

		$M = 100$	$M = 500$	$M = 1000$	$M = 5000$
2-exchange	*Jackknife*	0.96 (0.01)	0.87 (0.04)	0.81 (0.06)	0.56 (0.08)
	Bootstrap	0.97 (0.01)	0.87 (0.04)	0.79 (0.06)	0.47 (0.12)
	Chao1984	0.83 (0.14)	0.79 (0.06)	0.73 (0.06)	0.52 (0.07)
	Chao&Lee1	0.84 (0.12)	0.78 (0.05)	0.71 (0.07)	0.49 (0.06)
	Chao&Lee2	**0.80 (0.15)**	**0.71 (0.07)**	**0.62 (0.09)**	**0.37 (0.05)**
	Chao&Bunge	0.95 (0.06)	2.38 (8.21)	1.12 (1.44)	0.51 (0.47)
Insert	*Jackknife*	0.33 (0.30)	0.18 (0.27)	0.16 (0.22)	0.77 (0.49)
	Bootstrap	**0.28 (0.32)**	0.37 (0.21)	0.26 (0.17)	1.05 (0.61)
	Chao1984	0.35 (0.25)	**0.18 (0.24)**	0.14 (0.20)	**0.72 (0.47)**
	Chao&Lee1	0.35 (0.28)	0.19 (0.23)	0.14 (0.19)	0.75 (0.48)
	Chao&Lee2	0.33 (0.33)	**0.18 (0.24)**	**0.12 (0.15)**	0.82 (0.51)
	Chao&Bunge	0.49 (0.74)	0.31 (0.70)	1.39 (4.95)	0.83 (0.37)

Table 7 Average relative errors and standard deviations (in brackets) of the estimations provided by the different methods for the instances of the QAP, according to the neighborhood and the sample size M

		$M = 100$	$M = 500$	$M = 1000$	$M = 5000$
2-exchange	*Jackknife*	0.87 (0.17)	0.75 (0.25)	0.67 (0.28)	0.50 (0.29)
	Bootstrap	0.84 (0.20)	0.69 (0.33)	0.62 (0.34)	0.44 (0.32)
	Chao1984	0.77 (0.24)	0.57 (0.21)	0.53 (0.19)	0.41 (0.19)
	Chao&Lee1	0.78 (0.20)	0.56 (0.21)	0.53 (0.19)	0.42 (0.19)
	Chao&Lee2	**0.73 (0.25)**	**0.52 (0.22)**	**0.49 (0.19)**	0.38 (0.19)
	Chao&Bunge	0.91 (0.40)	0.87 (1.41)	1.89 (5.45)	**0.27 (0.39)**
Insert	*Jackknife*	1.00 (0.00)	1.00 (0.00)	1.00 (0.01)	0.98 (0.03)
	Bootstrap	1.00 (0.00)	1.00 (0.00)	1.00 (0.01)	0.98 (0.03)
	Chao1984	**0.99 (0.02)**	**0.96 (0.04)**	**0.93 (0.06)**	0.89 (0.04)
	Chao&Lee1	**0.99 (0.02)**	0.97 (0.03)	0.94 (0.05)	0.89 (0.04)
	Chao&Lee2	**0.99 (0.02)**	**0.96 (0.04)**	**0.93 (0.06)**	**0.86 (0.06)**
	Chao&Bunge	1.00 (0.01)	0.99 (0.01)	0.98 (0.03)	0.98 (0.03)

and the standard deviation presented in most of the cases is really high compared with other methods. It seems that *Chao&Bunge* is a very unstable method. The instability of this method is a consequence of the variability on the estimation of the parameter that represents the expected proportion of duplicates in the sample [22]. This estimation is unreliable when the sample has many local optima that are seen only once, but there is a small number of local optima seen a low (but higher than one) number of times. These particularities are commonly found when the sample

size is small with respect to the number of local optima or even when the variance of the sizes of the attraction basins of the local optima is high.

As a conclusion from this analysis, we highly recommend the use of the *Chao&Lee2* method to estimate the number of local optima. According to the hypothesis test, this estimation method provides the best results. Although in the statistical analysis it is not reflected, the *Chao&Bunge* method gives also good estimations in a high number of occasions; however, its instability provokes that it is not trusted. Therefore, we recommend to execute both methods independently. If the results provided are close, *Chao&Bunge* is usually the choice; otherwise, select *Chao&Lee2*.

Conclusions

The objective of this study has been to extend and advance the knowledge associated to implementing multi-start procedures. Unlike other well-known methods, it has not yet become widely implemented and tested as a metaheuristic itself for solving complex optimization problems. Previous efforts have been classified according to its domain of applicability: global and combinatorial optimization.

In this chapter, different methods have been reviewed for estimating the number of local optima of instances of combinatorial optimization problems. Additionally, some methods in the optimization field have been compared with methods previously used for estimating the number of species in a population in the field of statistics. The methods have been applied to instances of three well-known problems in combinatorial optimization and two classical neighborhoods. The main conclusions are that, in general, the higher the sample used by the methods, the more precise the estimations, and the higher the number of local optima, the worse the estimations provided. Based on the results observed through the experiments, we recommend using the *Chao&Lee2* method or applying it together with the *ChaoBunge* method for a more accurate local optima estimation.

Cross-References

▸ GRASP
▸ Restart Strategies
▸ Tabu Search
▸ Theory of Local Search

Acknowledgments This work has been partially supported by the Spanish Ministerio de Economía y Competitividad (codes TIN2016-78365-R, TIN2015-65460, and TIN2013-41272P), the Basque Government (IT-609-13 program), and Generalitat Valenciana (project Prometeo 2013/049).

References

1. Albrecht A, Lane P, Steinhofel K (2008) Combinatorial landscape analysis for k-SAT instances. In: IEEE congress on evolutionary computation, CEC 2008, Hong Kong. IEEE World congress on computational intelligence, pp 2498–2504
2. Albrecht A, Lane P, Steinhofel K (2010) Analysis of local search landscapes for k-SAT instances. Math Comput Sci 3(4):465–488
3. Beausoleil R, Baldoquin G, Montejo R (2008) A multi-start and path relinking methods to deal with multiobjective knapsack problems. Ann Oper Res 157:105–133
4. Boese K, Kahng A, Muddu S (1994) A new adaptive multi-start technique for combinatorial global optimisation. Oper Res Lett 16:103–113
5. Braysy O, Hasle G, Dullaert W (2004) A multi-start local search algorithm for the vehicle routing problem with time windows. Eur J Oper Res 159:586–605
6. Chao A (1984) Nonparametric estimation of the number of classes in a population. Scand J Stat 11(4):265–270
7. Chao A, Bunge J (2002) Estimating the number of species in a stochastic abundance model. Biometrics 58(3):531–539
8. Chao A, Lee SM (1992) Estimating the number of classes via sample coverage. J Am Stat Assoc 87(417):210–217
9. Crowston WB, Glover F, Thompson GL, Trawick JD (1963) Probabilistic and parametric learning combinations of local job shop scheduling rules. Technical report 117, Carnegie-Mellon University, Pittsburgh
10. Dhouib S, Kharrat A, Chabchoub H (2010) A multi-start threshold accepting algorithm for multiple objective continuous optimization problems. Int J Numer Methods Eng 83:1498–1517
11. Eremeev AV, Reeves CR (2002) Non-parametric estimation of properties of combinatorial landscapes. In: Cagnoni S, Gottlieb J, Hart E, Middendorf M, Raidl G (eds) Applications of evolutionary computing. Lecture notes in computer science, vol 2279. Springer, Berlin/Heidelberg, pp 31–40
12. Eremeev AV, Reeves CR (2003) On confidence intervals for the number of local optima. In: Proceedings of EvoWorkshops 2003, Essex, pp 224–235
13. Essafi M, Delorme X, Dolgui A (2010) Balancing lines with CNC machines: a multi-start and based heuristic. CIRP J Manuf Sci Technol 2:176–182
14. Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71
15. Feo T, Resende M (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
16. Fleurent C, Glover F (1999) Improved constructive multi-start strategies for the quadratic assignment problem using adaptive memory. INFORMS J Comput 11:198–204
17. Glover F (2000) Multi-start and strategic oscillation methods – principles to exploit adaptive memory. In: Laguna M, Gonzalez-Velarde J (eds) Computing tools for modeling optimization and simulation. Kluwer Academic, Boston, pp 1–25
18. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Boston
19. Grundel D, Krokhmal P, Oliveira C, Pardalos P (2007) On the number of local minima for the multidimensional assignment problem. J Combin Optim 13:1–18
20. Hagen L, Kahng A (1997) Combining problem reduction and adaptive multi-start: a new technique for superior iterative partitioning. IEEE Trans CAD 16:709–717
21. Held M, Karp R (1970) The traveling-salesman problem and minimum spanning trees. Oper Res 18:1138–1162
22. Hernando L, Mendiburu A, Lozano JA (2013) An evaluation of methods for estimating the number of local optima in combinatorial optimization problems. Evol Comput 21(4):625–658
23. Hickernell F, Yuan Y (1997) A simple multistart algorithm for global optimization. OR Trans 1:1–11
24. Hu X, Shonkwiler R, Spruill M (1994) Random restarts in global optimization. Ga Inst Technol 1:1–10

25. Kan AR, Timmer G (1987) Stochastic global optimization methods (Part II): multi level methods. Math Program 39:57–78
26. Kan AR, Timmer G (1998) Global optimization. In: Kan R, Todds (eds) Handbooks in operations research and management science. North Holland, Amsterdam, pp 631–662
27. Kaucic M (2013) A multi-start opposition-based particle swarm optimization algorithm with adaptive velocity for bound constrained global optimization. J Glob Optim 55:165–188
28. Lan G, DePuy G (2006) On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem. Comput Ind Eng 51:362–374
29. Martí R, Reinelt G, Duarte A (2012) A benchmark library and a comparison of heuristic methods for the linear ordering problem. Comput Optim Appl 51(3):1297–1317
30. Martí R, Resende M, Ribeiro C (2013) Multi-start methods for combinatorial optimization. Eur J Oper Res 226(1):1–8
31. Mayne DQ, Meewella C (1988) A non-clustering multistart algorithm for global optimization. In: Bensoussan A, Lions J-L (eds) Analysis and optimization of systems. Lecture notes in control and information sciences. Springer, Berlin/New York, pp 111–117
32. Mezmaz M, Melab N, Talbi E (2006) Using the multi-start and island models for parallel multi-objective optimization on the computational grid. In: Second IEEE international conference on e-science and grid computing, Amsterdam
33. Moreno J, Mladenovic N, Moreno-Vega J (1995) An statistical analysis of strategies for multistart heuristic searches for p-facility location-allocation problems. In: Eighth meeting of the EWG on locational analysis Lambrecht
34. Muth JF, Thompson GL (1963) Industrial scheduling. Prentice-Hall, Englewood Cliffs
35. Patterson R, Pirkul H, Rolland E (1999) Adaptive reasoning technique for the capacitated minimum spanning tree problem. J Heuristics 5:159–180
36. Reeves C, Aupetit-Bélaidouni M (2004) Estimating the number of solutions for SAT problems. In: Yao X, Burke E, Lozano J, Smith J, Merelo-Guervós J, Bullinaria J, Rowe J, Tino P, Kabán A, Schwefel HP (eds) Parallel problem solving from nature – PPSN VIII. Lecture notes in computer science, vol 3242. Springer, Berlin/Heidelberg, pp 101–110
37. Resende M, Ribeiro C (2010) Greedy randomized adaptive search procedures: advances and applications. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics, 2nd edn. Springer, New York, pp 293–319
38. Solis F, Wets R (1981) Minimization by random search techniques. Math Oper Res 6:19–30
39. Taillard E, Badeau P, Gendreau M, Guertin F, Potvin J (1997) A tabu search heuristic of the vehicle routing problem with time windows. Transp Sci 31:170–186
40. Tu W, Mayne R (2002) An approach to multi-start clustering for global optimization with non-linear constraints. Int J Numer Methods Eng 53:2253–2269
41. Ugray Z, Lasdon L, Plummer J, Glover F, Kelly J, Martí R (2007) Scatter search and local NLP solvers: a multistart framework for global optimization. INFORMS J Comput 19(3):328–340
42. Ugray Z, Lasdon L, Plummer J, Bussieck M (2009) Dynamic filters and randomized drivers for the multi-start global optimization algorithm MSNLP. Optim Methods Softw 24:4–5
43. Villegas J, Prins C, Prodhon C, Medaglia A, Velasco N (2010) GRASP/VND and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots. Eng Appl Artif Intell 23:780–794

Multi-objective Optimization

7

Carlos A. Coello Coello

Contents

Introduction . . . 178
Basic Concepts . . . 179
Pareto Optimality . . . 179
Multi-objective Evolutionary Algorithms . . . 180
Multi-objective Evolutionary Algorithms . . . 182
Recent Algorithmic Trends . . . 185
Use of Other Metaheuristics . . . 186
Artificial Immune Systems (AIS) . . . 186
Particle Swarm Optimization (PSO) . . . 187
Ant Colony Optimization (ACO) . . . 187
Other Metaheuristics . . . 188
Some Applications . . . 188
Some Current Challenges . . . 189
Conclusions . . . 191
Cross-References . . . 192
References . . . 192

Abstract

This chapter provides a short overview of multi-objective optimization using metaheuristics. The chapter includes a description of some of the main metaheuristics that have been used for multi-objective optimization. Although special emphasis is made on evolutionary algorithms, other metaheuristics, such as particle swarm optimization, artificial immune systems, and ant colony optimization, are also briefly discussed. Other topics such as applications and

C. A. Coello Coello (✉)
Departmento de Computación, CINVESTAV-IPN, Mexico City, México
e-mail: ccoello@cs.cinvestav.mx

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_17

recent algorithmic trends are also included. Finally, some of the main research trends that are worth exploring in this area are briefly discussed.

Keywords
Multi-objective optimization · metaheuristics · evolutionary algorithms · optimization

Introduction

Metaheuristics have been widely used for solving different types of optimization problems (see, e.g., [88, 109, 198]).

One particular class of optimization problems involves having two or more (often conflicting) objectives which we aim to optimize at the same time. In fact, such problems, which are called "multi-objective" are quite common in real-world applications, and their solution has triggered an important amount of work within Operations Research [135].

During the last 40 years, a large number of mathematical programming techniques have been developed to solve certain specific classes of multi-objective optimization problems. However, such techniques have a relatively limited applicability (e.g., some of them require the first or even the second derivative of the objective functions and the constraints; others can only deal with convex Pareto fronts, etc.). Such limitations have motivated the development of alternative optimization methods, from which metaheuristics have become a very popular alternative [43].

From the many metaheuristics currently available, evolutionary algorithms have been, without doubt, the most popular choice for dealing with any sort of optimization problem [21, 85], and multi-objective optimization is, by no means, an exception. Thus, in this chapter, we will focus our discussion mainly on the use of evolutionary algorithms for solving multi-objective optimization problems.

The use of evolutionary algorithms for solving multi-objective optimization problems was originally hinted in 1967 [173], but the first actual implementation of what is now called a "multi-objective evolutionary algorithm (MOEA)" was not produced until the mid-1980s [178, 179]. However, this area, which is now called "evolutionary multi-objective optimization," or EMO, has experienced a very important growth, mainly in the last 20 years [41, 42, 52, 201]. The author maintains the EMOO repository, which, as of December 8, 2017, contains over 11,000 bibliographic entries, as well as public-domain implementation of some of the most popular MOEAs. The EMOO repository is located at https://emoo.cs.cinvestav.mx/.

The remainder of this chapter is organized as follows: In section "Basic Concepts", we provide some basic concepts related to multi-objective optimization, which are required to make this chapter self-contained. The use of evolutionary algorithms in multi-objective optimization is motivated in section "Multi-objective Evolutionary Algorithms". A short discussion on other bio-inspired metaheuristics

that have also been used for multi-objective optimization is provided in section "Use of Other Metaheuristics". Some of the main research topics which are currently attracting a lot of attention in the EMO field are briefly discussed in section "Multi-objective Evolutionary Algorithms". A set of sample applications of MOEAs is provided in section "Some Applications". Some of the main topics of research in the EMO field that currently attract a lot of attention are briefly discussed in section "Some Current Challenges". Such topics include the use of other metaheuristics. Finally, some conclusions are provided in section "Conclusions".

Basic Concepts

In this chapter, we focus on the solution of multi-objective optimization problems (MOPs) of the form:

$$\text{minimize } \left[f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})\right] \tag{1}$$

subject to the m inequality constraints

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, \ldots, m \tag{2}$$

and the p equality constraints

$$h_i(\vec{x}) = 0 \quad i = 1, 2, \ldots, p \tag{3}$$

where k is the number of objective functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$. We call $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ the vector of decision variables. We wish to determine from among the set $\mathcal{F}$ of all vectors which satisfy (2) and (3) the particular set of values $x_1^*, x_2^*, \ldots, x_n^*$ which yield the optimum values of all the objective functions.

Pareto Optimality

It is rarely the case that there is a single point that simultaneously optimizes all the objective functions. In fact, this situation only arises when there is no conflict among the objectives, which would make unnecessary the development of special solution methods, since this single solution could be reached after the sequential optimization of all the objectives, considered separately. Therefore, we normally look for "trade-offs," rather than single solutions when dealing with multi-objective optimization problems. The notion of "optimality" normally adopted in this case is the one originally proposed by Francis Ysidro Edgeworth [63] and later generalized by the French economist Vilfredo Pareto [152]. Although some authors call this notion *Edgeworth-Pareto optimality*, we will use the most commonly adopted term: *Pareto optimality*.

We say that a vector of decision variables $\vec{x}^* \in \mathcal{F}$ (i.e., a feasible solution) is *Pareto optimal* if there does not exist another $\vec{x} \in \mathcal{F}$ such that $f_i(\vec{x}) \leq f_i(\vec{x}^*)$ for all $i = 1, \ldots, k$ and $f_j(\vec{x}) < f_j(\vec{x}^*)$ for at least one j (assuming that all the objectives are being minimized).

In words, this definition says that $\vec{x}^*$ is a Pareto optimal solution if there exists no feasible vector of decision variables $\vec{x} \in \mathcal{F}$ which would decrease some criterion without causing a simultaneous increase in at least one other criterion. Assuming the inherent conflict normally present among (at least some) objectives, the use of this concept normally produces several solutions. Such solutions constitute the so-called *Pareto optimal set*. The vectors $\vec{x}^*$ corresponding to the solutions included in the Pareto optimal set are called *nondominated*. The image of the Pareto optimal set under the objective functions (i.e., the objective function values corresponding to the decision variables contained in the Pareto optimal set) is called *Pareto front*.

Multi-objective Evolutionary Algorithms

The core ideas related to the development of search techniques that simulate the mechanism of natural selection (Darwin's survival of the fittest principle) can be traced back to the early 1930s [75]. However, the three main techniques based on this notion were developed during the 1960s: genetic algorithms [101], evolution strategies [184], and evolutionary programming [74]. These approaches, which are now generically denominated "evolutionary algorithms," have been found to be very effective for solving single-objective optimization problems [76, 89, 185].

The basic operation of an evolutionary algorithm (EA) is described next. First, a set of potential solutions (called "population") to the problem being solved is randomly generated. Each solution in the population (called "individual") encodes all the decision variables of the problem (i.e., each individual contains all the decision variables of the problem to be solved). The user needs to define a measure of performance for each of the solutions. Such a measure of performance is called "fitness function" and will allow us to know how good is a solution with respect to the others. Such a fitness function is normally a variation of the objective function of the problem that we wish to solve (e.g., the objective function that we aim to optimize). Then, a selection mechanism must be applied in order to decide which individuals will "mate." This selection process is generally stochastic and is normally based on the fitness contribution of each individual (i.e., the fittest individuals have a higher probability of being selected). Upon mating, a set of "offspring" (or children) are generated. Such offspring are "mutated" (this operator produces a small random change, with a low probability, on the contents of an individual) and constitute the new population to be evaluated at the next iteration (each iteration is called a "generation"). This process is repeated until reaching a stopping condition (normally, a maximum number of generations defined by the user) [64].

The main motivation for using EAs for solving multi-objective optimization problems relies on their population-based nature, which allows them to generate (if properly manipulated) several elements of the Pareto optimal set, in a single run. In contrast, mathematical programming techniques normally generate a single element of the Pareto optimal set per run. Additionally, the so-called multi-objective evolutionary algorithms (MOEAs) are less susceptible to the shape and continuity of the Pareto front and require less specific domain information to operate [41].

MOEAs extend a traditional (single-objective) EA in two main aspects:

- **The selection mechanism:** In this case, the aim is to select nondominated solutions and to consider all the nondominated solutions in a population to be equally good (unless there is some specific preference from the user, all the elements of the Pareto optimal set are equally good).
- **A diversity maintenance mechanism:** Because of stochastic noise, EAs tend to converge to a single solution if run for a sufficiently long time [89]. In order to avoid this, it is necessary to block the selection mechanism in a MOEA, favoring the diversity of solutions, so that several elements of the Pareto optimal set can be generated in a single run.

Regarding selection, early MOEAs relied on the use of aggregating functions (mainly linear) [93] and relatively simple population-based approaches [179]. However, such approaches have evident drawbacks (i.e., the use of linear aggregating functions does not allow the generation of non-convex portions of the Pareto front regardless of the weights combination that is adopted [49]). Toward the mid-1990s, MOEAs started to adopt variations of the so-called *Pareto ranking* selection mechanism. This approach was originally proposed by David E. Goldberg in his seminal book on genetic algorithms [89], and it consists of sorting the population of an EA based on Pareto optimality, such that all nondominated individuals are assigned the same rank (or importance). The aim is that all nondominated individuals get the same probability of being selected and that such probability is higher than the one corresponding to individuals which are dominated. Although conceptually simple, this sort of selection mechanism allows for a wide variety of possible implementations [41, 52].

Regarding diversity maintenance, a wide variety of methods have been proposed in the specialized literature to maintain diversity in a MOEA. Such approaches include fitness sharing and niching [53, 91], clustering [203, 230], geographically based schemes [120], the use of entropy [46, 117], and parallel coordinates [100], among others. In all cases, the core idea behind diversity maintenance mechanisms is to penalize solutions that are too close from each other in some space (i.e., decision variable or objective function space or even both). Most MOEAs penalize solutions that are too close from each other in objective function space, because it is normally aimed to have solutions well-distributed along the Pareto front.

Additionally, some researchers have proposed the use of mating restriction schemes (which imposes rules on the individuals that can be recombined) [188, 230]. Furthermore, the use of relaxed forms of Pareto dominance has also become

relatively popular in recent years, mainly as an archiving technique which encourages diversity, while allowing the archive to regulate convergence (the most popular of such mechanisms is, with no doubt, ϵ-dominance [124], which has been adopted in some approaches such as ϵ-MOEA [57]).

A third component of modern MOEAs is elitism, which normally consists of using an external archive (also called "secondary population") that may (or may not) interact in different ways with the main (or "primary") population of the MOEA during selection. The main purpose of this archive is to store *all* the nondominated solutions generated throughout the search process, while removing those that become dominated later in the search (called local nondominated solutions). The approximation of the Pareto optimal set produced by a MOEA is thus the final contents of this archive. It is important to emphasize that the use of elitism is not only advisable (the lack of elitism could make us lose nondominated solutions generated during the search), but it is also required because of theoretical reasons (elitism is required in order to guarantee convergence of a MOEA to the Pareto optimal set as proved in [174]).

It is worth noticing that, in practice, external archives are normally bounded to a certain maximum number of solutions. This was originally done in some MOEAs that used the external archive during the selection stage (see, e.g., [230]). In such a case, allowing the size of the archive to grow too much dilutes the selection pressure, which has a negative effect on the performance of the MOEA. However, most modern MOEAs bound the size of the external archive, even if the archive is not used during the selection process, mainly because of practical reasons (this makes it easier to compare results with respect to other MOEAs).

An important remark is that the use of a plus (+) selection is another possible elitist mechanism. Under this sort of selection scheme, the population of parents competes with the population of offspring (both populations are of the same size), and then we keep only the best half. This sort of selection scheme has been relatively popular in single-objective optimization and has been also adopted in some modern MOEAs (see, e.g., [56]), but it is less popular than the use of external archives.

Multi-objective Evolutionary Algorithms

In spite of the very large number of publications related to MOEAs that can be found in the literature, there is only a handful of algorithms that are actually used by a significant number of researchers and/or practitioners around the world.

1. **Strength Pareto Evolutionary Algorithm 2** (SPEA2): This is an updated version of the Strength Pareto Evolutionary Algorithm (SPEA) proposed in the late 1990s [230] whose main features are the following. It adopts an external archive (called the external nondominated set), which stores the nondominated solutions previously generated and participates in the selection process (together with the main population). For each individual in this archive, a *strength* value is computed. This strength value is proportional to the number of solutions that a

certain individual dominates. In SPEA, the fitness of each member of the current population is computed according to the strengths of all external nondominated solutions that dominate it. As the size of the external nondominated set grows too much, this significantly reduces the selection pressure and slows down the search. In order to avoid this, SPEA adopts a clustering technique that prunes the contents of the external nondominated set so that its size remains below a certain (pre-defined) threshold. SPEA standardized the use of external archives as the elitist mechanism of a MOEA, although this sort of mechanism had been used before by other researchers (see, e.g., [107]). SPEA2 has three main differences with respect to the original SPEA [231]: (1) it incorporates a fine-grained fitness assignment strategy which takes into account, for each individual, both the number of individuals that dominate it and the number of individuals by which it is dominated, (2) it uses a nearest neighbor density estimation technique which guides the search more efficiently (i.e., a more efficient clustering algorithm is adopted), and (3) it uses an enhanced archive truncation method that guarantees the preservation of boundary solutions (this fixes a bug from the original SPEA).

2. **Pareto Archived Evolution Strategy** (PAES): This is perhaps the most simple MOEA than can be possibly designed. It was proposed by Knowles and Corne [119], and it consists of a (1+1) evolution strategy (i.e., a single parent that generates a single offspring through the application of mutation to the parent) in combination with a historical archive that stores the nondominated solutions previously found. This archive is used as a reference set against which each mutated individual is being compared. Such (external) archive adopts a crowding procedure that divides objective function space in a recursive manner. Then, each solution is placed in a certain grid location based on the values of its objectives (which are used as its "coordinates" or "geographical location"). A map of such a grid is maintained, indicating the number of solutions that reside in each grid location. When a new nondominated solution is ready to be stored in the archive, but there is no room for it (the size of the external archive is bounded), a check is made on the grid location to which the solution would belong. If this grid location is less densely populated than the most densely populated grid location, then a solution (randomly chosen) from this heavily populated grid location is deleted to allow the storage of the newcomer. This aims to redistribute solutions, favoring the less densely populated regions of the Pareto front. Since the procedure is adaptive, no extra parameters are required (except for the number of divisions of the objective space).

3. **Nondominated Sorting Genetic Algorithm II** (NSGA-II): This is a heavily revised version of the Nondominated Sorting Genetic Algorithm (NSGA), which was originally proposed in the mid-1990s [193] as a straightforward implementation of the Pareto ranking algorithm described by Goldberg in his book [89]. NSGA was, however, slow and produced poorer results than other non-elitist MOEAs available in the mid-1990s, such as MOGA [77] and NPGA [103]. NSGA-II adopts a more efficient ranking procedure than its predecessor.

Additionally, it estimates the density of solutions surrounding a particular solution in the population by computing the average distance of two points on either side of this point along each of the objectives of the problem. This value is the so-called crowding distance. During selection, NSGA-II uses a crowded comparison operator which takes into consideration both the nondomination rank of an individual in the population and its crowding distance (i.e., nondominated solutions are preferred over dominated solutions, but between two solutions with the same nondomination rank, the one that resides in the less crowded region is preferred). NSGA-II does not use an external archive as most of the modern MOEAs in current use. Instead, the elitist mechanism of NSGA-II consists of combining the best parents with the best offspring obtained (i.e., this is a $(\mu + \lambda)$-selection used in evolution strategies). Due to its clever mechanisms, NSGA-II is much more efficient (computationally speaking) than its predecessor, and its performance is so good that it has become very popular, triggering a significant number of applications and becoming some sort of landmark against which new MOEAs have been compared during more than 10 years, in order to merit publication. More recently, the Nondominated Sorting Genetic Algorithm III (NSGA-III) [54] was introduced. NSGA-III still adopts the same NSGA-II framework (i.e., it still performs a classification of the population in nondominated levels). However, its mechanism to maintain diversity is based on the use of a number of well-spread reference points which are adaptively updated. The NSGA-III does not require any additional parameters (other than those associated to the genetic algorithm that is used as its search engine), and it is designed to deal with many-objective optimization problems (i.e., problems having 4 or more objectives).

4. **Pareto Envelope-based Selection Algorithm** (PESA): This algorithm was proposed by Corne et al. [44] and uses a small internal population and a larger external (or secondary) population. PESA adopts the same adaptive grid from PAES to maintain diversity. However, its selection mechanism is based on the crowding measure. This same crowding measure is used to decide what solutions to introduce into the external population (i.e., the archive of nondominated vectors found along the evolutionary process). Therefore, in PESA, the external memory plays a crucial role in the algorithm since it determines not only the diversity scheme but also the selection performed by the method. There is also a revised version of this algorithm, called PESA-II [45], which is identical to PESA, except for the fact that a region-based selection is used in this case. In region-based selection, the unit of selection is a hyperbox rather than an individual. The procedure consists of selecting (using any of the traditional selection techniques [90]) a hyperbox and then randomly selecting an individual within such hyperbox. The main motivation of this approach is to reduce the computational costs associated with traditional MOEAs (i.e., those based on Pareto ranking).
5. **Multi-objective Evolutionary Algorithm based on Decomposition** (MOEA/D): This approach was proposed by Zhang and Li [221]. The main idea of this

algorithm is to decompose a multi-objective optimization problem into several scalar optimization subproblems which are simultaneously optimized. The decomposition process requires the use of weights, but the authors provide a method to generate them. During the optimization of each subproblem, only information from the neighboring subproblems is used, which allows this algorithm to be effective and efficient. MOEA/D is generally considered one of the most powerful MOEAs currently available, as has been evidenced by several comparative studies.

Recent Algorithmic Trends

Many other MOEAs have been proposed in the specialized literature (see, e.g., [39, 57, 200, 202, 219]), but they will not be discussed here due to obvious space limitations. A more interesting issue, however, is to try to predict which sort of MOEA will become predominant in the next few years.

Efficiency is, for example, a concern nowadays, and several approaches have been developed in order to improve the efficiency of MOEAs (see, e.g., [113]). Also, the use of fitness approximation, fitness inheritance, surrogates, and other similar techniques has become more common in recent years, which is a clear indication of the more frequent use of MOEAs for the solution of computationally expensive problems (see, e.g., [118, 157, 166, 169, 177, 217, 222]).

However, the most promising research line within algorithmic design seems to be the use of a performance measure in the selection mechanism of a MOEA. This research trend formally started with the Indicator-Based Evolutionary Algorithm (IBEA) [229], although this idea had been already formulated and used (in different ways) by other authors (see, e.g., [106, 120]). Nevertheless, the most representative MOEA within this family is perhaps the S metric selection Evolutionary Multi-Objective Algorithm (SMS-EMOA) [19, 67]. SMS-EMOA was originally proposed by Emmerich et al. [67] and is based on NSGA-II. SMS-EMOA creates an initial population, and then, it generates only one solution by iteration using the operators (crossover and mutation) of the NSGA-II. After that, it applies Pareto ranking. When the last front has more than one solution, SMS-EMOA uses the hypervolume contribution to decide which solution will be removed. The **Hypervolume** (also known as the S metric or the Lebesgue Measure) of a set of solutions measures the size of the portion of objective space that is dominated by those solutions collectively. This is the only unary performance indicator which is known to be Pareto compliant [232]. Beume et al. [19] proposed not to use the contribution to the hypervolume indicator when in the Pareto ranking we obtain more than one front. In that case, they proposed to use the number of solutions which dominate to one solution (the solution that is dominated by more solutions is removed). The authors argued that the motivation for using the hypervolume indicator is to improve the distribution in the nondominated front and then it is not necessary in fronts different to the nondominated front.

The hypervolume indicator has attracted a lot of attention from researchers due to its interesting theoretical properties. For example, it has been proven that the maximization of the hypervolume is equivalent to finding the Pareto optimal set [73]. Empirical studies have shown that (for a certain number of points previously determined) the maximization of the hypervolume does indeed produce subsets of the Pareto front which are well-distributed [67, 120].

However, there are also practical reasons for being interested in indicator-based selection. The main one is that MOEAs such as SMS-EMOA seem to continue working as usual as we increase the number of objectives, as opposed to Pareto-based selection mechanisms which are known to degrade quickly in problems having more than three objectives (this research area is known as *many-objective optimization*). Although the reasons for the poor scalability of Pareto-based MOEAs requires further study (see, e.g., [182]), the need for scalable selection mechanisms has triggered an important amount of research around indicator-based MOEAs. The main drawback of adopting the hypervolume in the selection mechanism of a MOEA is its extremely high computational cost. One possible alternative for dealing with this high computational cost is to estimate the hypervolume contribution. However, MOEAs designed around this idea (e.g., HyPE [7]) seem to have a poor performance with respect to those that adopt exact hypervolume contributions. An alternative is to rely on other performance indicators. In this regard, several researchers have proposed the design of selection mechanisms based on performance measures such as Δ_p [168, 183] and $R2$ [24, 94, 99, 154]. The use of the maximin [10, 132] is another intriguing alternative, as this expression seems to be equivalent to the use of the ϵ-indicator [233].

Use of Other Metaheuristics

A wide variety of other bio-inspired metaheuristics have become popular in the last few years for solving optimization problems [43]. Multi-objective extensions of many of these metaheuristics already exist [41], but few efforts have been made to actually exploit the main specific features of each of them. There are also few efforts in trying to understand the types of problems in which each of these metaheuristics can be more suitable.

Next, we briefly review three popular bio-inspired metaheuristics that are good candidates for being used as multi-objective optimizers, but many other choices also exist (see, e.g., [41]).

Artificial Immune Systems (AIS)

Our natural immune system has provided a fascinating metaphor for developing a new bio-inspired metaheuristic. Indeed, from a computational point of view, our immune system can be considered as a highly parallel intelligent system that is able to learn and retrieve previously acquired knowledge (i.e., it has "memory"),

when solving highly complex recognition and classification tasks. This motivated the development of the so-called artificial immune systems (AISs) during the early 1990s [50, 51].

AISs were identified in the early 1990s as a useful mechanism to maintain diversity in the context of multimodal optimization [78, 191]. Smith et al. [192] showed that fitness sharing can emerge when their emulation of the immune system is used. Furthermore, the approach that they proposed turns out to be more efficient (computationally speaking) than traditional fitness sharing [53], and it does not require additional information regarding the number of niches to be formed.

Over the years, a wide variety of multi-objective extensions of AISs have been proposed (see, e.g., [27, 32, 38, 79, 130, 155]). However, most of the algorithmic trends in MOEAs have had a delayed arrival in multi-objective AISs. Also, the high potential of multi-objective AISs for pattern recognition and classification tasks has been scarcely exploited.

For more information on multi-objective AISs, the interested reader is referred to [28, 80].

Particle Swarm Optimization (PSO)

This metaheuristic is inspired on the movements of a flock of birds seeking food, and it was originally proposed in the mid-1990s [116]. In the particle swarm optimization algorithm, the behavior of each particle (i.e., individual) is affected by either the best local (within a certain neighborhood) or the best global (i.e., with respect to the entire swarm or population) individual. PSO allows particles to benefit from their past experiences (a mechanism that does not exist in traditional evolutionary algorithms) and uses neighborhood structures that can regulate the behavior of the algorithm.

The similarity of PSO with EAs has made possible the quick development of an important number of multi-objective variants of this metaheuristic (see, e.g., [40, 139, 142, 153, 165, 216]). Unlike AISs, most algorithmic trends adopted with MOEAs have quickly been incorporated into multi-objective particle swarm optimizers (MOPSOs). Nevertheless, few PSO models have been used for multi-objective optimization, and the study of the specific features that could make MOPSOs advantageous over other metaheuristics in some specific classes of problems is still a pending task.

For more information on MOPSOs, the interested reader is referred to [11, 37, 41, 61, 167].

Ant Colony Optimization (ACO)

This metaheuristic was inspired on the behavior observed in colonies of real ants seeking for food. Ants deposit a chemical substance on the ground, called pheromone [60], which influences the behavior of the ants: they tend to take those

paths in which there is a larger amount of pheromone. Therefore, pheromone trails can be seen as an indirect communication mechanism used by the ants (which can be seen as agents that interact to solve complex tasks). This interesting behavior of ants gave rise to a metaheuristic called *ant system*, which was originally applied to the traveling salesperson problem. Nowadays, the several variations of this algorithm that have been developed over the years are collectively denominated by *ant colony optimization* (ACO), and they have been applied to a wide variety of domains, including continuous optimization problems.

Several multi-objective versions of ACO are currently available (see, e.g., [2, 4, 70, 111, 137, 140, 170]). However, the algorithmic trends developed in MOEAs have had a slow delay in being incorporated into multi-objective ACO algorithms. Additionally, the use of alternative ACO models for multi-objective optimization has been relatively scarce.

For more information on multi-objective ACO, the interested reader is referred to [4, 41, 82].

Other Metaheuristics

Many other metaheuristics have been extended to deal with multi-objective optimization problems, including simulated annealing [48, 65, 189, 190, 194], differential evolution [134, 138, 197, 199, 204, 207, 208], tabu search [30, 86, 95, 149], scatter search [12, 15, 16, 158, 160, 211], and artificial bee colony [33, 58, 127, 159], among many others. However, their discussion was omitted due to obvious space constraints.

Some Applications

Multi-objective metaheuristics have been extensively applied to a wide variety of domains. Next, we will provide a short list of sample applications classified in three large groups: (1) engineering, (2) industrial, and (3) scientific. Specific areas within each of these large groups are also identified.

By far, engineering applications are the most popular in the current literature on multi-objective metaheuristics. This is not surprising if we consider that engineering disciplines normally have problems with better understood mathematical models. A representative sample of engineering applications is the following:

- Electrical engineering [161, 176]
- Hydraulic engineering [186, 196]
- Structural engineering [218, 224]
- Aeronautical engineering [6, 172]
- Robotics [115, 220]
- Control [123, 131]
- Telecommunications [105, 141]

- Civil engineering [145, 214]
- Transport engineering [8, 223]

Industrial applications are the second most popular in the literature on multi-objective metaheuristics. A representative sample of industrial applications of multi-objective metaheuristics is the following:

- Design and manufacture [9, 17, 228]
- Scheduling [26, 29]
- Management [34, 69]

Finally, there are several publications devoted to scientific applications. For obvious reasons, computer science applications are the most popular in the literature on multi-objective metaheuristics. A representative sample of scientific applications is the following:

- Chemistry [13, 71, 126, 227]
- Physics [14, 171]
- Medicine [97, 122]
- Computer science [146, 175]

This sample of applications should give at least a rough idea of the increasing interest of researchers for adopting multi-objective metaheuristics in practically all kinds of disciplines.

Some Current Challenges

The existence of challenging, but solvable problems, is a key issue to preserve the interest in a research discipline. Although multi-objective optimization using metaheuristics is a discipline in which a very important amount of research has been conducted, mainly within the last 15 years, several interesting problems still remain open. Additionally, the research conducted so far has also led to new and intriguing topics. The following is a small sample of open problems that currently attract a significant amount of research within this area:

- **Scalability**: Although multi-objective metaheuristics have been commonly used for a wide variety of applications, they have certain limitations. For example, as indicated before, selection mechanisms based on Pareto optimality are known to degrade quickly as we increase the number of objectives. The reason is that, as we increase the number of objectives, unless there is a significant increase in the population size, all the individuals will quickly become nondominated, which will cause stagnation (i.e., no individual will be better than the others, which makes the selection mechanism totally useless). In fact, there is experimental evidence that indicates that, when dealing with ten objectives, random sampling

performs better than Pareto-based selection [121]. There are alternative ranking techniques that can be used to deal with these problems having four or more objectives [83, 129], but it is also possible to use relaxed forms of Pareto dominance [59, 72], dimensionality reduction techniques [23, 128], or indicator-based selection mechanisms [19, 229]. It is worth indicating that scalability in decision variable space (i.e., the capability of multi-objective metaheuristics for dealing with large-scale problems) has been scarcely studied in the specialized literature (see, e.g., [5, 62]).

- **Incorporation of user's preferences**: It is normally the case that the user does not need the entire Pareto front of a problem but only a certain portion of it. For example, solutions lying at the extreme parts of the Pareto front are normally unnecessary since they represent the best value for one objective, but the worst for the others. Thus, if the user has at least a rough idea of the sort of trade-offs that he/she aims to find, it is desirable to be able to explore in more detail only the nondominated solutions within the neighborhood of such trade-offs. This is possible, if we use, for example, biased versions of Pareto ranking [47] or some multi-criteria decision- making technique, from the many developed in Operations Research [22, 35, 150]. In spite of the importance of preference incorporation in real-world applications, the use of these schemes in multi-objective metaheuristics is still relatively scarce [31, 68, 81, 110].

- **Dealing with expensive problems**: There is an important number of real-world multi-objective problems for which a single evaluation of an objective function may take several minutes, hours, or even days [6]. Evidently, such problems require special techniques to be solvable using MOEAs [177]. The main approaches that have been developed in this area can be roughly divided into three main groups:

  1. **Use of parallelism:** This is the most obvious approach given the current access to low-cost parallel architectures (e.g., GPUs [18, 187]). It is worth noting, however, that in spite of the existence of interesting proposals (see, e.g., [3, 136]), the basic research in this area has remained scarce, since most publications involving parallel MOEAs focus on specific applications or on parallel extensions of specific MOEAs, but rarely involve basic research.
  2. **Surrogates:** In this case, knowledge of past evaluations of a MOEA is used to build an empirical model that approximates the fitness functions to be optimized. This approximation can then be used to predict promising new solutions at a smaller evaluation cost than that of the original problem [114, 118]. Current functional approximation models include polynomials (response surface methodologies [162]), neural networks (e.g., multilayer perceptrons (MLPs) [102, 108, 156]), radial-basis function (RBF) networks [147, 205, 213], support vector machines (SVMs) [1, 20], Gaussian processes [25, 206], and Kriging [66, 163] models. The use of statistical models developed in the mathematical programming literature in combination with

MOEAs is also an interesting alternative (see, e.g., the multi-objective P-algorithm [151, 226]).

3. **Fitness approximation:** The idea of these approaches is to predict the fitness value of an individual without performing its actual evaluation. From the several techniques available, fitness inheritance has been the most popular in multi-objective optimization [164], but the research in this area has remained scarce.

- **Theoretical foundations**: Although an important effort has been made in recent years to provide a solid theoretical foundation to this field, a lot of work is still required, and the work done in the Operations Research community can provide some useful hints (see, e.g., [151, 225]). The most relevant theoretical work in this area includes topics such as convergence [174, 209], archiving [180, 181], algorithm complexity [143, 144], and run-time analysis [87, 125].
- **Constraint-handling**: The use of multi-objective optimization concepts to handle constraints in single-objective evolutionary algorithms has been a relatively popular research area (see, e.g., [84, 98, 133, 195, 210, 215]). In contrast, however, the design of constraint-handling mechanisms for MOEAs has not been that popular (see, e.g., [92, 96, 104, 112, 148, 212]). This is remarkable, given the importance of constraints in real-world applications, but this may be due to the fact that some modern MOEAs (e.g., NSGA-II) already incorporated a constraint-handling mechanism. Additionally, the use of benchmarks containing constrained test problems (e.g., [55]) has not been very popular in the specialized literature. Most of the current research in this area has focused on extending the Pareto optimality relation in order to incorporate constraints (e.g., giving preference to feasibility over dominance, such that an infeasible solution is discarded even if it is nondominated). Also, the use of penalty functions that "punish" a solution for not being feasible are easy to incorporate into a MOEA [36]. However, topics such as the design of constraint-handling mechanisms for dealing with equality constraints, the design of scalable test functions that incorporate constraints of different types (linear, nonlinear, equality, inequality), and the study of mechanisms that allow an efficient exploration of constrained search spaces in MOPs remain practically unexplored.

Conclusions

In this chapter, we have provided some basic concepts related to multi-objective optimization using metaheuristics. This overview has included basic concepts related to multi-objective optimization in general, as well as some algorithmic details, with a particular emphasis on multi-objective evolutionary algorithms.

Some of the recent algorithmic trends have also been discussed, and some sample applications have been addressed. In general, breadth has been favored over depth, but a significant number of bibliographic references are provided for those interested in gaining an in-depth knowledge of any of the topics discussed herein.

The information provided in this chapter aims to serve as a general overview of the field and to motivate the interest of the reader for pursuing research in this area. As could be seen, several research opportunities are still available for newcomers.

Cross-References

- ▸ Ant Colony Optimization: A Component-Wise Overview
- ▸ Evolutionary Algorithms
- ▸ Genetic Algorithms
- ▸ Particle Swarm Methods
- ▸ Theoretical Analysis of Stochastic Search Algorithms

Acknowledgments The author acknowledges support from CONACYT project no. 221551.

References

1. Abboud K, Schoenauer M (2002) Surrogate deterministic mutation. In: Collet P, Fonlupt C, Hao J-K, Lutton E, Schoenauer M (eds) Artificial evolution, 5th international conference, evolution artificielle, EA 2001. Lecture notes in computer science, vol 2310. Springer, Le Creusot, pp 103–115
2. Akay B (2013) Synchronous and asynchronous Pareto-based multi-objective Artificial Bee Colony algorithms. J Glob Optim 57(2):415–445
3. Alba E, Luque G, Nesmachnow S (2013) Parallel metaheuristics: recent advances and new trends. Int Trans Oper Res 20(1):1–48
4. Angus D, Woodward C (2009) Multiple objective ant colony optimisation. Swarm Intell 3(1):69–85
5. Antonio LM, Coello Coello CA (2013) Use of cooperative coevolution for solving large scale multiobjective optimization problems. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 2758–2765. ISBN:978-1-4799-0454-9
6. Arias-Montaño A, Coello Coello CA, Mezura-Montes E (2012) Multi-objective evolutionary algorithms in aeronautical and aerospace engineering. IEEE Trans Evol Comput 16(5): 662–694
7. Bader J, Zitzler E (2011) HypE: an algorithm for fast hypervolume-based many-objective optimization. Evol Comput 19(1):45–76. Spring
8. Bai Q, Labi S, Sinha KC (2012) Trade-off analysis for multiobjective optimization in transportation asset management by generating Pareto frontiers using extreme points non-dominated sorting genetic algorithm II. J Trans Eng-ASCE 138(6):798–808
9. Balesdent M, Berend N, Depince P, Chriette A (2012) A survey of multidisciplinary design optimization methods in launch vehicle design. Struct Multidiscip Optim 45(5):619–642
10. Balling R, Wilson S (2001) The maximin fitness function for multi-objective evolutionary computation: application to city planning. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO'2001), San Francisco. Morgan Kaufmann Publishers, pp 1079–1084
11. Banks A, Vincent J, Anyakoha C (2008) A review of particle swarm optimization. II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Nat Comput 7(1):109–124. Unconventional Computation 2006, Selected Papers

12. Baños R, Gil C, Reca J, Martínez J (2009) Implementation of scatter search for multi-objective optimization: a comparative study. Comput Optim Appl 42(3):421–441
13. Baronas R, Žilinskas A, Litvinas L (2016) Optimal design of amperometric biosensors applying multi-objective optimization and decision visualization. Electrochim Acta 211: 586–594
14. Bartolini R, Apollonio M, Martin IPS (2012) Multi-objective genetic algorithm optimization of the beam dynamics in linac drivers for free electron lasers. Phys Rev Spec Top Accel Beams 15(3). Article number:030701
15. Beausoleil RP (2006) "MOSS" multiobjective scatter search applied to non-linear multiple criteria optimization. Eur J Oper Res 169(2):426–449
16. Beausoleil RP (2008) "MOSS-II" Tabu/Scatter search for nonlinear multiobjective optimization. In: Siarry P, Michalewicz Z (eds) Advances in metaheuristic methods for hard optimization. Springer, Berlin, pp 39–67. ISBN:978-3-540-72959-4
17. Benyoucef L, Xie X (2011) Supply chain design using simulation-based NSGA-II approach. In: Wang L, Ng AHC, Deb K (eds) Multi-objective evolutionary optimisation for product design and manufacturing. Springer, London, pp 455–491. ISBN:978-0-85729-617-7. Chapter 17
18. Bernardes de Oliveira F, Davendra D, Gadelha Guimar aes F (2013) Multi-objective differential evolution on the GPU with C-CUDA. In: Snášel V, Abraham A, Corchado ES (eds) Soft computing models in industrial and environmental applications, 7th international conference (SOCO'12). Advances in intelligent systems and computing, vol 188. Springer, Ostrava, pp 123–132
19. Beume N, Naujoks B, Emmerich M (2007) SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur J Oper Res 181(3):1653–1669
20. Bhattacharya M, Lu G (2003) A dynamic approximate fitness based hybrid ea for optimization problems. In: Proceedings of IEEE congress on evolutionary computation, pp 1879–1886
21. Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer Academic Publishers, Boston. ISBN:0-7923-7631-5
22. Branke J (2008) Consideration of partial user preferences in evolutionary multiobjective optimization. In: Branke J, Deb K, Miettinen K, Slowinski R (eds) Multiobjective optimization. Interactive and evolutionary approaches. Lecture notes in computer science, vol 5252. Springer, Berlin, pp 157–178
23. Brockhoff D, Friedrich T, Hebbinghaus N, Klein C, Neumann F, Zitzler E (2007) Do additional objectives make a problem harder? In: Thierens D (ed) 2007 genetic and evolutionary computation conference (GECCO'2007), vol 1. ACM Press, London, pp 765–772
24. Brockhoff D, Wagner T, Trautmann H (2012) On the properties of the $R2$ indicator. In: 2012 genetic and evolutionary computation conference (GECCO'2012). ACM Press, Philadelphia, pp 465–472. ISBN:978-1-4503-1177-9
25. Bueche D, Schraudolph NN, Koumoutsakos P (2005) Accelerating evolutionary algorithms with gaussian process fitness function models. IEEE Trans Syst Man Cybern Part C 35(2):183–194
26. Burke EK, Li J, Qu R (2012) A Pareto-based search methodology for multi-objective nurse scheduling. Ann Oper Res 196(1):91–109
27. Campelo F, Guimar aes FG, Saldanha RR, Igarashi H, Noguchi S, Lowther DA, Ramirez JA (2004) A novel multiobjective immune algorithm using nondominated sorting. In: 11th international IGTE symposium on numerical field calculation in electrical engineering, Seggauberg
28. Campelo F, Guimar aes FG, Igarashi H (2007) Overview of artificial immune systems for multi-objective optimization. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Evolutionary multi-criterion optimization, 4th international conference (EMO 2007), Matshushima. Lecture notes in computer science, vol 4403. Springer, pp 937–951
29. Campos SC, Arroyo JEC (2014) NSGA-II with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. In: 2014 genetic and evolutionary computation conference (GECCO 2014), Vancouver. ACM Press, pp 429–436. ISBN:978-1-4503-2662-9

30. Carcangiu S, Fanni A, Montisci A (2008) Multiobjective Tabu search algorithms for optimal design of electromagnetic devices. IEEE Trans Magn 44(6):970–973
31. Carrese R, Winarto H, Li X, Sobester A, Ebenezer S (2012) A comprehensive preference-based optimization framework with application to high-lift aerodynamic design. Eng Optim 44(10):1209–1227
32. Chang Y-C (2012) Multi-objective optimal SVC installation for power system loading margin improvement. IEEE Trans Power Syst 27(2):984–992
33. Chaves-Gonzalez JM, Vega-Rodriguez MA, Granado-Criado JM (2013) A multiobjective swarm intelligence approach based on artificial bee colony for reliable DNA sequence design. Eng Appl Artif Intel 26(9):2045–2057
34. Chikumbo O, Goodman E, Deb K (2012) Approximating a multi-dimensional Pareto front for a land use management problem: a modified MOEA with an epigenetic silencing metaphor. In: 2012 IEEE congress on evolutionary computation (CEC'2012), Brisbane. IEEE Press, pp 480–488
35. Coello Coello CA (2000) Constraint-handling using an evolutionary multiobjective optimization technique. Civ Eng Environ Syst 17:319–346
36. Coello Coello CA (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191(11–12):1245–1287
37. Coello Coello CA (2011) An introduction to multi-objective particle swarm optimizers. In: Gaspar-Cunha A, Takahashi R, Schaefer G, Costa L (eds) Soft computing in industrial applications. Advances in intelligent and soft computing series, vol 96. Springer, Berlin, pp 3–12. ISBN:978-3-642-20504-0
38. Coello Coello CA, Cruz Cortés N (2005) Solving multiobjective optimization problems using an artificial immune system. Genet Program Evolvable Mach 6(2):163–190
39. Coello Coello CA, Toscano Pulido G (2001) Multiobjective optimization using a micro-genetic algorithm. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO'2001), San Francisco. Morgan Kaufmann Publishers, pp 274–282
40. Coello Coello CA, Toscano Pulido G, Salazar Lechuga M (2004) Handling multiple objectives with particle swarm optimization. IEEE Trans Evol Comput 8(3):256–279
41. Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) Evolutionary algorithms for solving multi-objective problems, 2nd edn. Springer, New York. ISBN:978-0-387-33254-3
42. Collette Y, Siarry P (2003) Multiobjective optimization. Principles and case studies. Springer Berlin, Germany. ISBN:3-540-40182-2
43. Corne D, Glover F, Dorigo M (eds) (1999) New ideas in optimization. McGraw-Hill, Berkshire. ISBN:007-709506-5
44. Corne DW, Knowles JD, Oates MJ (2000) The Pareto envelope-based selection algorithm for multiobjective optimization. In: Schoenauer M, Deb K, Rudolph G, Yao X, Lutton E, Merelo JJ, Schwefel H-P (eds) Proceedings of the parallel problem solving from nature VI conference, Paris. Lecture notes in computer science, vol 1917. Springer, pp 839–848
45. Corne DW, Jerram NR, Knowles JD, Oates MJ (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO'2001), San Francisco. Morgan Kaufmann Publishers, pp 283–290
46. Cui X, Li M, Fang T (2001) Study of population diversity of multiobjective evolutionary algorithm based on immune and entropy principles. In: Proceedings of the congress on evolutionary computation 2001 (CEC'2001), Piscataway, vol 2. IEEE Service Center, pp 1316–1321
47. Cvetković D, Parmee IC (2002) Preferences and their application in evolutionary multiobjective optimisation. IEEE Trans Evol Comput 6(1):42–57

48. Czyzak P, Jaszkiewicz A (1998) Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. J Multi-Criteria Decis Anal 7:34–47
49. Das I, Dennis J (1997) A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. Struct Optim 14(1):63–69
50. Dasgupta D (ed) (1999) Artificial immune systems and their applications. Springer, Berlin
51. de Castro LN, Timmis J (2002) An introduction to artificial immune systems: a new computational intelligence paradigm. Springer, London. ISBN:1-85233-594-7
52. Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, Chichester. ISBN:0-471-87339-X
53. Deb K, Goldberg DE (1989) An investigation of niche and species formation in genetic function optimization. In: Schaffer JD (ed) Proceedings of the third international conference on genetic algorithms, San Mateo. George Mason University, Morgan Kaufmann Publishers, pp 42–50
54. Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans Evol Comput 18(4):577–601
55. Deb K, Pratap A, Meyarivan T (2001) Constrained test problems for multi-objective evolutionary optimization. In: Zitzler E, Deb K, Thiele L, Coello Coello CA, Corne D (eds) First international conference on evolutionary multi-criterion optimization. Lecture notes in computer science, vol 1993. Springer, pp 284–298
56. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA–II. IEEE Trans Evol Comput 6(2):182–197
57. Deb K, Mohan M, Mishra S (2005) Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions. Evol Comput 13(4):501–525. Winter
58. Dhouib S, Dhouib S, Chabchoub H (2013) Artificial bee colony metaheuristic to find Pareto optimal solutions set for engineering design problems. In: 2013 5th international conference on modeling, simulation and applied optimization (ICMSAO), Hammamet. IEEE Press. ISBN:978-1-4673-5812-5
59. di Pierro F, Khu S-T, Savić DA (2007) An investigation on preference order ranking scheme for multiobjective evolutionary optimization. IEEE Trans Evol Comput 11(1): 17–45
60. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge. ISBN:0-262-04219-3
61. Durillo JJ, García-Nieto J, Nebro AJ, Coello Coello CA, Luna F, Alba E (2009) Multi-objective particle swarm optimizers: an experimental comparison. In: Ehrgott M, Fonseca CM, Gandibleux X, Hao J-K, Sevaux M (eds) Evolutionary multi-criterion optimization. 5th international conference (EMO 2009). Lecture notes in computer science, vol 5467. Springer, Nantes, pp 495–509
62. Durillo JJ, Nebro AJ, Coello Coello CA, Garcia-Nieto J, Luna F, Alba E (2010) A study of multiobjective metaheuristics when solving parameter scalable problems. IEEE Trans Evol Comput 14(4):618–635
63. Edgeworth FY (1881) Mathematical psychics. P. Keagan, London
64. Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Berlin. ISBN:3-540-40184-9
65. Ekbal A, Saha S (2013) Combining feature selection and classifier ensemble using a multiobjective simulated annealing approach: application to named entity recognition. Soft Comput 17(1):1–16
66. Emmerich M, Giotis A, Özdemir M, Bäck T, Giannakoglou K (2002) Metamodel-assisted evolution strategies. In: Merelo Guervós JJ, Adamidis P, Beyer H-G, Fernández-Villaca nas J-L, Schwefel H-P (eds) Parallel problem solving from nature—PPSN VII, Granada. Lecture notes in computer science, vol 2439. Springer, pp 371–380
67. Emmerich M, Beume N, Naujoks B (2005) An EMO algorithm using the hypervolume measure as selection criterion. In: Coello Coello CA, Hernández Aguirre A, Zitzler E

(eds) Evolutionary multi-criterion optimization. Third international conference (EMO 2005), Guanajuato. Lecture notes in computer science, vol 3410. Springer, pp 62–76
68. Eppe S, López-Ibá nez M, Stützle T, De Smet Y (2011) An experimental study of preference model integration into multi-objective optimization heuristics. In: 2011 IEEE congress on evolutionary computation (CEC'2011), New Orleans. IEEE Service Center, pp 2751–2758
69. Esparcia-Alcazar AI, Martínez-García A, García-Sánchez P, Merelo JJ, Mora AM (2013) Towards a multiobjective evolutionary approach to inventory and routing management in a retail chain. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 3166–3173. ISBN:978-1-4799-0454-9
70. Falcon-Cardona JG, Coello Coello CA (2017) A new indicator-based many-objective ant colony optimizer for continuous search spaces. Swarm Intell 11(1):71–100
71. Fang G, Xue M, Su M, Hu D, Li Y, Xiong B, Ma L, Meng T, Chen Y, Li J, Li J, Shen J (2012) CCLab-a multi-objective genetic algorithm based combinatorial library desing software and an application for histone deacetylase inhibitor desing. Bioorg Med Chem Lett 22(14): 4540–4545
72. Farina M, Amato P (2004) A fuzzy definition of "optimality" for many-criteria optimization problems. IEEE Trans Syst Man and Cybern Part A Syst Hum 34(3):315–326
73. Fleischer M (2003) The measure of Pareto optima. Applications to multi-objective metaheuristics. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary multi-criterion optimization. Second international conference (EMO 2003), Faro. Lecture notes in computer science, vol 2632. Springer, pp 519–533
74. Fogel LJ (1966) Artificial intelligence through simulated evolution. John Wiley, New York
75. Fogel DB (1995) Evolutionary computation. Toward a new philosophy of machine intelligence. The Institute of Electrical and Electronic Engineers, New York
76. Fogel LJ (1999) Artificial intelligence through simulated evolution. Forty years of evolutionary programming. Wiley, New York
77. Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms, San Mateo. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers, pp 416–423
78. Forrest S, Perelson AS (1991) Genetic algorithms and the immune system. In: Schwefel H-P, Männer R (eds) Parallel problem solving from nature. Lecture notes in computer science. Springer, Berlin, pp 320–325
79. Freschi F, Repetto M (2006) VIS: an artificial immune network for multi-objective optimization. Eng Optim 38(8):975–996
80. Freschi F, Coello Coello CA, Repetto M (2009) Multiobjective optimization and artificial immune systems: a review. In: Mo H (ed) Handbook of research on artificial immune systems and natural computing: applying complex adaptive technologies. Medical Information Science Reference, Hershey/New York, pp 1–21. ISBN:978-1-60566-310-4
81. Friedrich T, Kroeger T, Neumann F (2011) Weighted preferences in evolutionary multiobjective optimization. In: Wang D, Reynolds M (eds) AI 2011: advances in artificial intelligence, 24th Australasian joint conference, Perth. Lecture notes in computer science, vol 7106. Springer, pp 291–300
82. García-Martínez C, Cordón O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur J Oper Res 180(1):116–148
83. Garza Fabre M, Toscano Pulido G, Coello Coello CA (2009) Ranking methods for many-objective problems. In: Aguirre AH, Borja RM, García CAR (eds) MICAI 2009: advances in artificial intelligence. 8th Mexican international conference on artificial intelligence, Guanajuato. Lecture notes in artificial intelligence, vol 5845. Springer, pp 633–645
84. Garza-Fabre M, Rodriguez-Tello E, Toscano-Pulido G (2015) Constraint-handling through multi-objective optimization: the hydrophobic-polar model for protein structure prediction. Comput Oper Res 53:128–153
85. Gen M, Cheng R (2000) Genetic algorithms and engineering optimization. Wiley series in engineering design and automation. Wiley, New York

86. Ghisu T, Parks GT, Jaeggi DM, Jarrett JP, Clarkson PJ (2010) The benefits of adaptive parametrization in multi-objective Tabu search optimization. Eng Optim 42(10):959–981
87. Giel O (2003) Expected runtimes of a simple multi-objective evolutionary algorithm. In: Proceedings of the 2003 congress on evolutionary computation (CEC'2003), vol 3, Canberra. IEEE Press, pp 1918–1925
88. Glover F, Kochenberger GA (eds) (2003) Handbook of metaheuristics. Kluwer Academic Publishers, Boston. ISBN:1-4020-7263-5
89. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Publishing Company, Reading
90. Goldberg DE, Deb K (1991) A comparison of selection schemes used in genetic algorithms. In: Rawlins GJE (ed) Foundations of genetic algorithms. Morgan Kaufmann, San Mateo, pp 69–93
91. Goldberg DE, Richardson J (1987) Genetic algorithm with sharing for multimodal function optimization. In: Grefenstette JJ (ed) Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms, Hillsdale. Lawrence Erlbaum, pp 41–49
92. Gupta H, Deb K (2005) Handling constraints in robust multi-objective optimization. In: 2005 IEEE congress on evolutionary computation (CEC'2005), vol 1, Edinburgh. IEEE Service Center, pp 25–32
93. Hajela P, Lin CY (1992) Genetic search strategies in multicriterion optimal design. Struct Optim 4:99–107
94. Hansen MP (1998) Metaheuristics for multiple objective combinatorial optimization. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark
95. Hansen MP (2000) Tabu search for multiobjective combinatorial optimization: TAMOCO. Control Cybern 29(3):799–818
96. Harada K, Sakuma J, Ono I, Kobayashi S (2007) Constraint-handling method for multi-objective function optimization: Pareto descent repair operator. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Evolutionary multi-criterion optimization, 4th international conference (EMO 2007), Matshushima. Lecture notes in computer science, vol 4403. Springer, pp 156–170
97. Heris SMK, Khaloozadeh H (2011) Open- and closed-loop multiobjective optimal strategies for HIV therapy using NSGA-II. IEEE Trans Biomed Eng 58(6):1678–1685
98. Hernández Aguirre A, Botello Rionda S, Lizárraga Lizárraga G, Coello Coello C (2004) IS-PAES: multiobjective optimization with efficient constraint handling. In: Burczyński T, Osyczka A (eds) IUTAM symposium on evolutionary methods in mechanics. Kluwer Academic Publishers, Dordrecht/Boston/London, pp 111–120. ISBN:1-4020-2266-2
99. Hernández Gómez R, Coello Coello CA (2013) MOMBI: a new metaheuristic for many-objective optimization based on the $R2$ indicator. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 2488–2495. ISBN:978-1-4799-0454-9
100. Hernández Gómez R, Coello Coello CA, Alba Torres E (2016) A multi-objective evolutionary algorithm based on parallel coordinates. In: 2016 genetic and evolutionary computation conference (GECCO'2016), Denver. ACM Press, pp 565–572. ISBN:978-1-4503-4206-3
101. Holland JH (1962) Concerning efficient adaptive systems. In: Yovits MC, Jacobi GT, Goldstein GD (eds) Self-organizing systems—1962. Spartan Books, Washington, DC, pp 215–230
102. Hong Y-S, Lee H, Tahk M-J (2003) Acceleration of the convergence speed of evolutionary algorithms using multi-layer neural networks. Eng Optim 35(1):91–102
103. Horn J, Nafpliotis N, Goldberg DE (1994) A niched Pareto genetic algorithm for multiobjective optimization. In: Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence, Piscataway, vol 1. IEEE Service Center, pp 82–87
104. Hsieh M-N, Chiang T-C, Fu L-C (2011) A hybrid constraint handling mechanism with differential evolution for constrained multiobjective optimization. In: 2011 IEEE congress on evolutionary computation (CEC'2011), New Orleans. IEEE Service Center, pp 1785–1792
105. Huang B, Buckley B, Kechadi TM (2010) Multi-objective feature selection by using NSGA-II for customer churn prediction in telecommunications. Expert Syst Appl 37(5):3638–3646

106. Huband S, Hingston P, White L, Barone L (2003) An evolution strategy with probabilistic mutation for multi-objective optimisation. In: Proceedings of the 2003 congress on evolutionary computation (CEC'2003), Canberra, vol 3. IEEE Press, pp 2284–2291
107. Husbands P (1994) Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In: Fogarty TC (ed) Evolutionary computing. AIS workshop. Selected papers. Lecture notes in computer science, vol 865. Springer, pp 150–165
108. Hüscken M, Jin Y, Sendhoff B (2005) Structure optimization of neural networks for aerodynamic optimization. Soft Comput 9(1):21–28
109. Ibaraki T, Nonobe K, Yagiura M (eds) (2005) Metaheuristics. Progress as real problem solvers. Springer, New York. ISBN:978-0-387-25382-4
110. Iordache R, Iordache S, Moldoveanu F (2014) A framework for the study of preference incorporation in multiobjective evolutionary algorithms. In: 2014 genetic and evolutionary computation conference (GECCO 2014), Vancouver. ACM Press, pp 621–628. ISBN:978-1-4503-2662-9
111. Iredi S, Merkle D, Middendorf M (2001) Bi-criterion optimization with multi colony ant algorithms. In: Zitzler E, Deb K, Thiele L, Coello Coello CA, Corne D (eds) First international conference on evolutionary multi-criterion optimization. Lecture notes in computer science, vol 1993. Springer, pp 359–372
112. Jain H, Deb K (2014) An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach. IEEE Trans Evol Comput 18(4):602–622
113. Jensen MT (2003) Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms. IEEE Trans Evol Comput 7(5):503–515
114. Jin Y, Sendhoff B, Körner E (2005) Evolutionary multi-objective optimization for simultaneous generation of signal-type and symbol-type representations. In: Coello Coello CA, Hernández Aguirre A, Zitzler E (eds) Evolutionary multi-criterion optimization. Third international conference, EMO 2005, Guanajuato. Lecture notes in computer science, vol 3410. Springer, pp 752–766
115. Kelaiaia R, Zaatri A, Company O (2012) Multiobjective optimization of 6-dof UPS parallel manipulators. Adv Robot 26(16):1885–1913
116. Kennedy J, Eberhart RC (2001) Swarm intelligence. Morgan Kaufmann Publishers, San Francisco
117. Kita H, Yabumoto Y, Mori N, Nishikawa Y (1996) Multi-objective optimization by means of the thermodynamical genetic algorithm. In: Voigt H-M, Ebeling W, Rechenberg I, Schwefel H-P (eds) Parallel problem solving from nature—PPSN IV. Lecture notes in computer science, Berlin. Springer, pp 504–512
118. Knowles J (2006) ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Trans Evol Comput 10(1):50–66
119. Knowles JD, Corne DW (2000) Approximating the nondominated front using the pareto archived evolution strategy. Evol Comput 8(2):149–172
120. Knowles J, Corne D (2003) Properties of an adaptive archiving algorithm for storing nondominated vectors. IEEE Trans Evol Comput 7(2):100–116
121. Knowles J, Corne D (2007) Quantifying the effects of objective space dimension in evolutionary multiobjective optimization. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Evolutionary multi-criterion optimization, 4th international conference (EMO 2007), Matshushima. Lecture notes in computer science, vol 4403. Springer, pp 757–771
122. Lahsasna A, Ainon RN, Zainuddin R, Bulgiba A (2012) Design of a fuzzy-based decision support system for coronary heart disease diagnosis. J Med Syst 36(5):3293–3306
123. Larzabal E, Cubillos JA, Larrea M, Irigoyen E, Valera JJ (2012) Soft computing testing in real industrial platforms for process intelligent control. In: Snášel V, Abraham A, Corchado ES (eds) Soft computing models in industrial and environmental applications, 7th international conference (SOCO'12). Advances in intelligent systems and computing, vol 188. Springer, Ostrava, pp 221–230
124. Laumanns M, Thiele L, Deb K, Zitzler E (2002) Combining convergence and diversity in evolutionary multi-objective optimization. Evol Comput 10(3):263–282. Fall

125. Laumanns M, Thiele L, Zitzler E (2004) Running time analysis of multiobjective evolutionary algorithms on Pseudo-Boolean functions. IEEE Trans Evol Comput 8(2):170–182
126. Levene C, Correa E, Blanch EW, Goodacre R (2012) Enhancing surface enhanced raman scattering (SERS) detection of propranolol with multiobjective evolutionary optimization. Anal Chem 84(18):7899–7905
127. Li J-Q, Pan Q-K, Gao K-Z (2011) Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. Int J Adv Manuf Tech 55(9–12):1159–1169
128. López Jaimes A, Coello Coello CA, Chakraborty D (2008) Objective reduction using a feature selection technique. In: 2008 genetic and evolutionary computation conference (GECCO'2008), Atlanta. ACM Press, pp 674–680. ISBN:978-1-60558-131-6
129. López Jaimes A, Santana Quintero LV, Coello Coello CA (2009) Ranking methods in many-objective evolutionary algorithms. In: Chiong R (ed) Nature-inspired algorithms for optimisation. Springer, Berlin, pp 413–434. ISBN:978-3-642-00266-3
130. Luh G-C, Chueh C-H, Liu W-W (2003) MOIA: multi-objective immune algorithm. Eng Optim 35(2):143–164
131. Mahmoodabadi MJ, Arabani Mostaghim S, Bagheri A, Nariman-zadeh N (2013) Pareto optimal design of the decoupled sliding mode controller for an inverted pendulum system and its stability simulation via Java programming. Math Comput Model 57(5–6):1070–1082
132. Menchaca-Mendez A, Coello Coello CA (2013) Selection operators based on maximin fitness function for multi-objective evolutionary algorithms. In: Purshouse RC, Fleming PJ, Fonseca CM, Greco S, Shaw J (eds) Evolutionary multi-criterion optimization, 7th international conference (EMO 2013). Lecture notes in computer science, vol 7811, Sheffield. Springer, pp 215–229
133. Mezura-Montes E, Coello Coello CA (2008) Constrained optimization via multiobjective evolutionary algorithms. In: Knowles J, Corne D, Deb K (eds) Multi-objective problem solving from nature: from concepts to applications. Springer, Berlin, pp 53–75. ISBN:978-3-540-72963-1
134. Mezura-Montes E, Reyes-Sierra M, Coello Coello CA (2008) Multi-objective optimization using differential evolution: a survey of the state-of-the-art. In: Chakraborty UK (ed) Advances in differential evolution. Springer, Berlin, pp 173–196. ISBN:978-3-540-68827-3
135. Miettinen KM (1999) Nonlinear multiobjective optimization. Kluwer Academic Publishers, Boston
136. Mishra BSP, Dehuri S, Mall R, Ghosh A (2011) Parallel single and multiple objectives genetic algorithms: a survey. Int J Appl Evol Comput 2(2):21–57
137. Moncayo-Martinez LA, Zhang DZ (2011) Multi-objective ant colony optimisation: a meta-heuristic approach to supply chain design. Int J Prod Econ 131(1):407–420
138. Montaño AA, Coello Coello CA, Mezura-Montes E (2010) MODE-LD+SS: a novel differential evolution algorithm incorporating local dominance and scalar selection mechanisms for multi-objective optimization. In: 2010 IEEE congress on evolutionary computation (CEC'2010), Barcelona. IEEE Press, pp 3284–3291
139. Moore J, Chapman R, Dozier G (2000) Multiobjective particle swarm optimization. In: Turner AJ (ed) Proceedings of the 38th annual southeast regional conference, Clemson. ACM Press, pp 56–57
140. Mora AM, Garcia-Sanchez P, Merelo JJ, Castillo PA (2013) Pareto-based multi-colony multi-objective ant colony optimization algorithms: an island model proposal. Soft Comput 17(7):1175–1207
141. Narayanan L, Subramanian B, Arokiaswami A, Iruthayarajan MW (2012) Optimal placement of mobile antenna in an urban area using evolutionary multiobjective optimization. Microw Opt Technol Lett 54(3):737–743
142. Nebro AJ, Durillo JJ, Garcia-Nieto J, Coello Coello CA, Luna F, Alba E (2009) SMPSO: a new PSO-based metaheuristic for multi-objective optimization. In: 2009 IEEE symposium on computational intelligence in multi-criteria decision-making (MCDM'2009), Nashville. IEEE Press, pp 66–73. ISBN:978-1-4244-2764-2

143. Neumann F (2007) Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. Eur J Oper Res 181(3):1620–1629
144. Neumann F (2012) Computational complexity analysis of multi-objective genetic programming. In: 2012 genetic and evolutionary computation conference (GECCO'2012), Philadelphia. ACM Press, pp 799–806. ISBN:978-1-4503-1177-9
145. Ning X, Lam KC (2013) Cost-safety trade-off in unequal-area construction site layout planning. Autom Constr 32:96–103
146. Olmo JL, Romero JR, Ventura S (2012) Classification rule mining using ant programming guided by grammar with multiple Pareto fronts. Soft Comput 16(12):2143–2163
147. Ong YS, Nair PB, Keane AJ, Wong KW (2004) Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In: Jin Y (ed) Knowledge incorporation in evolutionary computation. Studies in fuzziness and soft computing. Springer, Berlin, Germany, pp 307–332
148. Oyama A, Shimoyama K, Fujii K (2007) New constraint-handling method for multi-objective and multi-constraint evolutionary optimization. Trans Jpn Soc Aeronaut Space Sci 50(167):56–62
149. Pacheco J, Marti R (2006) Tabu search for a multi-objective routing problem. J Oper Res Soc 57(1):29–37
150. Pardalos PM, Siskos Y, Zopounidis C (eds) (1995) Advances in multiciteria analysis. Springer-Science+Business Media, B.V. ISBN:978-1-4419-4748-2
151. Pardalos PM, Žilinskas A, Žilinskas J (2017) Non-convex multi-objective optimization. Springer, Cham. ISBN:978-3-319-61005-4
152. Pareto V (1896) Cours D'Economie Politique, vol I and II. F. Rouge, Lausanne
153. Parsopoulos KE, Taoulis DK, Pavlidis NG, Plagianakos VP, Vrahatis MN (2004) Vector evaluated differential evolution for multiobjective optimization. In: 2004 congress on evolutionary computation (CEC'2004), Portland, vol 1. IEEE Service Center, pp 204–211
154. Phan DH, Suzuki J (2013) R2-IBEA: R2 indicator based evolutionary algorithm for multi-objective optimization. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 1836–1845. ISBN:978-1-4799-0454-9
155. Pierrard T, Coello Coello CA (2012) A multi-objective artificial immune system based on hypervolume. In: Coelo Coello CA, Greensmith J, Krasnogor N, Liò P, Nicosia G, Pavone M (eds) Artificial immune systems, 11th international conference (ICARIS 2012). Lecture notes in computer science, vol 7597. Springer, Taormina, pp 14–27. ISBN:978-3-642-33756-7
156. Pierret S (1999) Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. ASME J Turbomach 121(3):326–332
157. Pilato C, Loiacono D, Tumeo A, Ferrandi F, Lanzi PL, Sciuto D (2010) Speeding-up expensive evaluations in high-level synthesis using solution modeling and fitness inheritance. In: Tenne Y, Goh C-K (eds) Computational intelligence in expensive optimization problems. Springer, Berlin, pp 701–723. ISBN:978-3-642-10700-9
158. Rahimi-Vahed AR, Javadi B, Rabbani M, Tavakkoli-Moghaddam R (2008) A multi-objective scatter search for a bi-criteria no-wait flow shop scheduling problem. Eng Optim 40(4): 331–346
159. Rakshit P, Konar A, Nagar AK (2014) Artificial bee colony induced multi-objective optimization in presence of noise. In: 2014 IEEE congress on evolutionary computation (CEC'2014), Beijing. IEEE Press, pp 3176–3183. ISBN:978-1-4799-1488-3
160. Rao ARM, Lakshmi K (2008) Multi-objective scatter search algorithm for combinatorial optimisation. In: Thulasiram R (ed) ADCOM: 2008 16th international conference on advanced computing and communications, Chennai. IEEE Press, pp 303–308. ISBN:978-1-4244-2962-2
161. Rao BS, Vaisakh K (2013) Multi-objective adaptive clonal selection algorithm for solving environmental/economic dispatch and OPF problems with load uncertainty. Int J Electr Power Energy Syst 53:390–408
162. Rasheed K, Ni X, Vattam S (2005) Comparison of methods for developing dynamic reduced models for design optimization. Soft Comput 9(1):29–37

163. Ratle A (1998) Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In: Eiben AE, Bäck T, Schoenauer M, Schwefel H-P (eds) Parallel problem solving from nature—PPSN V, 5th international conference, Amsterdam. Lecture notes in computer science, vol 1498. Springer, pp 87–96
164. Reyes Sierra M, Coello Coello CA (2005) Fitness inheritance in multi-objective particle swarm optimization. In: 2005 IEEE swarm intelligence symposium (SIS'05), Pasadena. IEEE Press, pp 116–123
165. Reyes Sierra M, Coello Coello CA (2005) Improving PSO-based multi-objective optimization using crowding, mutation and ϵ-dominance. In: Coello Coello CA, Hernández Aguirre A, Zitzler E (eds) Evolutionary multi-criterion optimization. Third international conference (EMO 2005), Guanajuato. Lecture notes in computer science, vol 3410. Springer, pp 505–519
166. Reyes Sierra M, Coello Coello CA (2005) A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization. In: 2005 IEEE congress on evolutionary computation (CEC'2005), Edinburgh, vol 1. IEEE Service Center, pp 65–72
167. Reyes-Sierra M, Coello Coello CA (2006) Multi-objective particle swarm optimizers: a survey of the state-of-the-art. Int J Comput Intell Res 2(3):287–308
168. Rodríguez Villalobos CA, Coello Coello CA (2012) A new multi-objective evolutionary algorithm based on a performance assessment indicator. In: 2012 genetic and evolutionary computation conference (GECCO'2012), Philadelphia. ACM Press, pp 505–512. ISBN:978-1-4503-1177-9
169. Rohling G (2008) Methods for decreasing the number of objective evaluations for independent computationally expensive objective problems. In: 2008 congress on evolutionary computation (CEC'2008), Hong Kong. IEEE Service Center, pp 3304–3309
170. Romero CEM, Manzanares EM (1999) MOAQ an ant-Q algorithm for multiple objective optimization problems. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Genetic and evolutionary computing conference (GECCO'99), San Francisco, vol 1. Morgan Kaufmann, pp 894–901
171. Romero-Garcia V, Sanchez-Perez JV, Garcia-Raffi LM (2012) Molding the acoustic attenuation in quasi-ordered structures: experimental realization. Appl Phys Express 5(8). Article number:087301
172. Ronco CCD, Ponza R, Benini E (2014) Aerodynamic shape optimization in aeronautics: a fast and effective multi-objective approach. Arch Comput Methods Eng 21(3):189–271
173. Rosenberg R (1967) Simulation of genetic populations with biochemical properties. PhD thesis, Department of Communication Sciences, University of Michigan, Ann Arbor
174. Rudolph G, Agapie A (2000) Convergence properties of some multi-objective evolutionary algorithms. In: Proceedings of the 2000 conference on evolutionary computation, Piscataway, vol 2. IEEE Press, pp 1010–1016
175. Saha I, Maulik U, Bandyopadhyay S, Plewczynski D (2011) Unsupervised and supervised learning approaches together for microarray analysis. Fundamenta Informaticae 106(1): 45–73
176. Sahoo NC, Ganguly S, Das D (2012) Fuzzy-Pareto-dominance driven possibilistic model based planning of electrical distribution systems using multi-objective particle swarm optimization. Expert Syst Appl 39(1):881–893
177. Santana-Quintero LV, Arias Montaño A, Coello Coello CA (2010) A review of techniques for handling expensive functions in evolutionary multi-objective optimization. In: Tenne Y, Goh C-K (eds) Computational intelligence in expensive optimization problems. Springer, Berlin, pp 29–59. ISBN:978-3-642-10700-9
178. Schaffer JD (1984) Multiple objective optimization with vector evaluated genetic algorithms. PhD thesis, Vanderbilt University, Nashville
179. Schaffer JD (1985) Multiple objective optimization with vector evaluated genetic algorithms. In: Genetic algorithms and their applications: proceedings of the first international conference on genetic algorithms. Lawrence Erlbaum, pp 93–100
180. Schuetze O, Laumanns M, Tantar E, Coello Coello CA, Talbi E (2007) Convergence of stochastic search algorithms to gap-free Pareto front approximations. In: Thierens D (ed)

2007 genetic and evolutionary computation conference (GECCO'2007), London, vol 1. ACM Press, pp 892–899
181. Schuetze O, Laumanns M, Tantar E, Coello Coello CA, Talbi E (2010) Computing gap free Pareto front approximations with stochastic search algorithms. Evol Comput 18(1):65–96. Spring
182. Schütze O, Lara A, Coello Coello CA (2011) On the influence of the number of objectives on the hardness of a multiobjective optimization problem. IEEE Trans Evol Comput 15(4):444–455
183. Schütze O, Esquivel X, Lara A, Coello Coello CA (2012) Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. IEEE Trans Evol Comput 16(4):504–522
184. Schwefel H-P (1965) Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Dipl.-Ing. thesis (in German)
185. Schwefel H-P (1981) Numerical optimization of computer models. Wiley, Chichester
186. Sharifi S, Massoudieh A (2012) A novel hybrid mechanistic-data-driven model identification framework using NSGA-II. J Hydroinf 14(3):697–715
187. Sharma D, Collet P (2013) Implementation techniques for massively parallel multi-objective optimization. In: Tsutsui S, Collet P (eds) Massively parallel evolutionary computation on GPGPUs. Springer, pp 267–286. ISBN:978-3-642-37958-1
188. Shaw KJ, Fleming PJ (1996) Initial study of practical multi-objective genetic algorithms for scheduling the production of chilled ready meals. In: Proceedings of mendel'96, the 2nd international mendel conference on genetic algorithms, Brno
189. Singh HK, Isaacs A, Ray T, Smith W (2008) A simulated annealing algorithm for constrained multi-objective optimization. In: 2008 congress on evolutionary computation (CEC'2008), Hong Kong. IEEE Service Center, pp 1655–1662
190. Smith KI (2006) A study of simulated annealing techniques for multi-objective optimisation. PhD thesis, University of Exeter
191. Smith RE, Forrest S, Perelson AS (1992) Searching for diverse, cooperative populations with genetic algorithms. Technical report TCGA No. 92002, University of Alabama, Tuscaloosa
192. Smith RE, Forrest S, Perelson AS (1993) Population diversity in an immune system model: implications for genetic search. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann Publishers, San Mateo, pp 153–165
193. Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248. Fall
194. Suman B, Kumar P (2006) A survey of simulated annealing as a tool for single and multiobjective optimization. J Oper Res Soc 57(10):1143–1160
195. Surry PD, Radcliffe NJ (1997) The COMOGA method: constrained optimisation by multiobjective genetic algorithms. Control Cybern 26(3):391–412
196. Sweetapple C, Fu G, Butler D (2014) Multi-objective optimisation of wastewater treatment plant control to reduce greenhouse gas emissions. Water Res 55:52–62
197. Tagawa K, Shimizu H, Nakamura H (2011) Indicator-based differential evolution using exclusive hypervolume approximation and parallelization for multi-core processors. In: 2011 genetic and evolutionary computation conference (GECCO'2011), Dublin. ACM Press, pp 657–664
198. Talbi E-G (ed) (2009) Metaheuristics. From design to implementation. Wiley, New Jersey. ISBN:978-0-470-27858-1
199. Talukder AKMKA, Kirley M, Buyya R (2009) Multiobjective differential evolution for scheduling workflow applications on global Grids. Concurrency Comput-Pract Exp 21(13):1742–1756
200. Tan KC, Lee TH, Khor EF (2001) Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. IEEE Trans Evol Comput 5(6):565–588
201. Tan KC, Khor EF, Lee TH (2005) Multiobjective evolutionary algorithms and applications. Springer, London. ISBN:1-85233-836-9

202. Toscano Pulido G, Coello Coello CA (2003) The micro genetic algorithm 2: towards online adaptation in evolutionary multiobjective optimization. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary multi-criterion optimization. Second international conference (EMO 2003), Faro. Lecture notes in computer science, vol 2632. Springer, pp 252–266
203. Toscano Pulido G, Coello Coello CA (2004) using clustering techniques to improve the performance of a particle swarm optimizer. In: Deb K et al (ed) Genetic and evolutionary computation–GECCO 2004. Proceedings of the genetic and evolutionary computation conference. Part I, Seattle, Washington. Lecture notes in computer science, vol 3102. Springer, pp 225–237
204. Tušar T, Filipič B (2007) Differential evolution versus genetic algorithms in multiobjective optimization. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Evolutionary multi-criterion optimization, 4th international conference (EMO 2007), Matshushima. Lecture notes in computer science, vol 4403. Springer, pp 257–271
205. Ulmer H, Streicher F, Zell A (2003) Model-assisted steady-state evolution strategies. In: Cantú-Paz E et al (ed) Genetic and evolutionary computation—GECCO 2003. Proceedings, Part I. Lecture notes in computer science, vol 2723. Springer, pp 610–621
206. Ulmer H, Streichert F, Zell A (2003) Evolution startegies assisted by Gaussian processes with improved pre-selection criterion. In: Proceedings of the 2003 IEEE congress on evolutionary computation (CEC'2003), Canberra, vol 1. IEEE Press, pp 692–699
207. Vargas DEC, Lemonge ACC, Barbosa HJC, Bernardino HS (2013) Differential evolution with the adaptive penalty method for constrained multiobjective optimization. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 1342–1349. ISBN:978-1-4799-0454-9
208. Venske SM, Goncalves RA, Delgado MR (2014) ADEMO/D: multiobjective optimization by an adaptive differential evolution algorithm. Neurocomputing 127:65–77
209. Villalobos-Arias M, Coello Coello CA, Hernández-Lerma O (2006) Asymptotic convergence of metaheuristics for multiobjective optimization problems. Soft Comput 10(11):1001–1005
210. Wang J, Terpenny JP (2005) Interactive preference incorporation in evolutionary engineering design. In: Jin Y (ed) Knowledge incorporation in evolutionary computation. Springer, Berlin/Heidelberg, pp 525–543. ISBN:3-540-22902-7
211. Wang X, Tang J, Yung K (2009) Optimization of the multi-objective dynamic cell formation problem using a scatter search approach. Int J Adv Manuf Technol 44(3–4):318–329
212. Woldesenbet YG, Tessema BG, Yen GG (2007) Constraint handling in multi-objective evolutionary optimization. In: 2007 IEEE congress on evolutionary computation (CEC'2007), Singapore. IEEE Press, pp 3077–3084
213. Won KS, Ray T (2004) Performance of kriging and cokriging based surrogate models within the unified framework for surrogate assisted optimization. In: 2004 congress on evolutionary computation (CEC'2004), Portland, vol 2. IEEE Service Center, pp 1577–1585
214. Xu J, Li Z (2012) Multi-objective dynamic costruction site layout plannig in fuzzy random environment. Autom Constr 27:155–169
215. Yong W, Zixing C (2005) A constrained optimization evolutionary algorithm based on multiobjective optimization techniques. In: 2005 IEEE congress on evolutionary computation (CEC'2005), Edinburgh, vol 2. IEEE Service Center, pp 1081–1087
216. Zapotecas Martínez S, Coello Coello CA (2011) A multi-objective particle swarm optimizer based on decomposition. In: 2011 genetic and evolutionary computation conference (GECCO'2011), Dublin. ACM Press, pp 69–76
217. Zapotecas Martínez S, Coello Coello CA (2013) Combining surrogate models and local search for dealing with expensive multi-objective optimization problems. In: 2013 IEEE congress on evolutionary computation (CEC'2013), Cancún. IEEE Press, pp 2572–2579. ISBN:978-1-4799-0454-9
218. Zavala GR, Nebro AJ, Luna F, Coello Coello CA (2014) A survey of multi-objective metaheuristics applied to structural optimization. Struct Multidiscip Optim 49(4):537–558

219. Zeng SY, Kang LS, Ding LX (2004) An orthogonal multi-objective evolutionary algorithm for multi-objective optimization problems with constraints. Evol Comput 12(1):77–98. Spring
220. Zhang D, Gao Z (2012) Forward kinematics, performance analysis, and multi-objective optimization of a bio-inspired parallel manipulator. Robot Comput Intregr Manuf 28(4): 484–492
221. Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
222. Zhang Q, Liu W, Tsang E, Virginas B (2010) Expensive multiobjective optimization by MOEA/D with Gaussian process model. IEEE Trans Evol Comput 14(3):456–474
223. Zheng Y-J, Chen S-Y (2013) Cooperative particle swarm optimization for multiobjective transportation planning. Appl Intell 39(1):202–216
224. Zhu J, Cai X, Pan P, Gu R (2014) Multi-objective structural optimization design of horizontal-axis wind turbine blades using the non-dominated sorting genetic algorithm II and finite element method. Energies 7(2):988–1002
225. Žilinskas A (2013) On the worst-case optimal multi-objective global optimization. Opt Lett 7:1921–1928
226. Žilinskas A (2014) A statistical model-based algorithm for 'black-box' multi-objective optimisation. Int J Syst Sci 45(1):82–93
227. Žilinskas A, Fraga ES, Mackuté A (2006) Data analysis and visualisation for robust multi-criteria process optimisation. Comput Chem Eng 30:1061–1071
228. Žilinskas J, Goldengorin B, Pardalos PM (2015) Pareto-optimal front of cell formation problem in group technology. J Glob Optim 61:91–108
229. Zitzler E, Künzli S (2004) Indicator-based selection in multiobjective search. In: Yao X et al (ed) Parallel problem solving from nature – PPSN VIII, Birmingham. Lecture notes in computer science, vol 3242. Springer, pp 832–842
230. Zitzler E, Deb K, Thiele L (1999) Comparison of multiobjective evolutionary algorithms on test functions of different difficulty. In: Wu AS (ed) Proceedings of the 1999 genetic and evolutionary computation conference. Workshop program, Orlando, pp 121–122
231. Zitzler E, Laumanns M, Thiele L (2002) SPEA2: improving the strength Pareto evolutionary algorithm. In: Giannakoglou K, Tsahalis D, Periaux J, Papailou P, Fogarty T (eds) EUROGEN 2001. Evolutionary methods for design, optimization and control with applications to industrial problems, Athens, pp 95–100
232. Zitzler E, Thiele L, Laumanns M, Fonseca CM, da Fonseca VG (2003) Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans Evol Comput 7(2):117–132
233. Zitzler E, Laumanns M, Bleuler S (2004) A tutorial on evolutionary multiobjective optimization. In: Gandibleux X, Sevaux M, Sörensen K, T'kindt V (eds) Metaheuristics for multiobjective optimisation, Berlin. Lecture notes in economics and mathematical systems, vol 535. Springer, pp 3–37

Restart Strategies

8

Oleg V. Shylo and Oleg A. Prokopyev

Contents

Introduction 206
Basic Restart Strategies 208
Restart Distribution and Optimal Restart Strategies 210
Single Algorithm Portfolios of Restart Algorithms 212
Mixed Algorithm Portfolios of Restart Algorithms 217
Conclusions 218
Cross-References 219
References 219

Abstract

This chapter is focused on restart strategies in optimization, which often provide a substantial algorithmic acceleration for randomized optimization procedures. Theoretical models that describe optimal restart strategies are presented alongside with their relations to parallel computing implementations.

Keywords

Algorithm · Algorithm portoflio · Parallel optimization · Restart · Superlinear speedup

O. V. Shylo (✉)
University of Tennessee, Knoxville, TN, USA
e-mail: oshylo@utk.edu

O. A. Prokopyev
University of Pittsburgh, Pittsburgh, PA, USA
e-mail: droleg@pitt.edu

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_15

Introduction

When evaluating performance of an optimization method, the most commonly reported statistics are running time (or execution time) and solution quality. Often these statistics are sensitive to initial conditions and algorithm parameters, especially in randomized algorithms. To capture this variability when analyzing computational performance, it is common to report a sequence of run-times and the corresponding objective function values (or their averages and some measures of dispersion) for different problem instances, initial conditions, and parameter settings. In general, it is natural to consider a probability distribution of time that an algorithm requires to obtain a solution of a given quality (e.g., an optimal solution or a solution whose value is within factor α from optimal). The variability in run-times can often be attributed to randomized algorithmic steps, such as randomized branching rules and column selection in branch-and-price methods, arbitrarily resolved ties, stochastic initial solutions, and random local search moves in metaheuristics.

In some applications, the distribution of the algorithm's run-time has a large spread, and there is a relatively high probability of having a run-time that is far from the average run-time value [6]. Such peculiar distributions can be exploited to accelerate the search via multi-start strategy: each run has a limited duration, and the algorithm is repeatedly restarted with different initial conditions or random seeds. Importantly, such acceleration opportunity is not just a mathematical curiosity, but it is commonly found in many real-life applications providing significant acceleration for the state-of-the-art optimization algorithms; see, e.g., [6, 12, 16, 18].

If a probability distribution of an algorithm's run-time admits such acceleration, it is referred to as a *restart distribution*. In [18], the authors provide the formal definition of a restart distribution. The authors also establish the theoretical explanation of the efficient restart strategies for enumeration algorithms. A search tree model is described in [2] to provide a formal proof of heavy-tailed behavior in imbalanced search trees.

Incorporating variability into deterministic methods may lead to faster algorithms. Randomizing certain decision in the deterministic algorithms may also induce restart distribution of run-times. This type of acceleration has been reported for various applications of randomized backtracking search. In [6], random steps are embedded in a complete search algorithm for the constraint satisfaction problem and the propositional satisfiability problem, and it is shown that the proposed randomization accelerates deterministic methods.

Finally, multi-start is a popular framework in many metaheuristics (see, e.g., [3, 13, 14, 16]). However, the impact of restart strategies on the computational performance is often underestimated. One of the most successful metaheuristics – tabu search – and its run-time distributions in the context of restart strategies are discussed in [19].

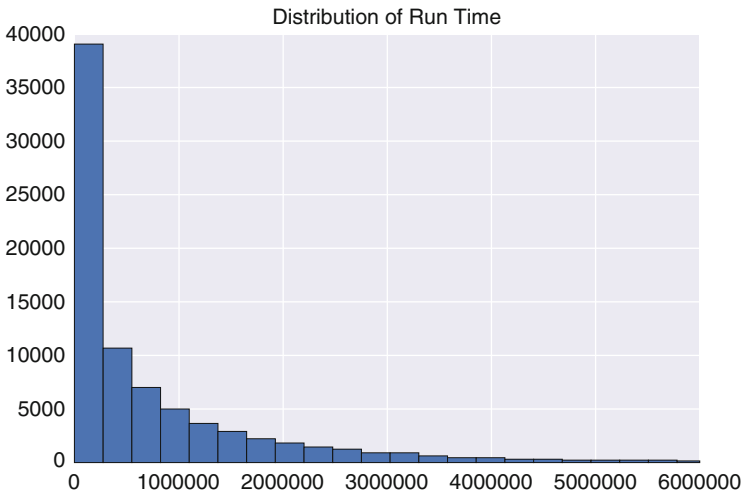


Fig. 1 The distribution of run-time of a tabu search algorithm for an instance of the maximum cut problem based on 80,000 runs

The following example of a restart distribution is based on a randomly generated instance of the directed *maximum cut problem* [20] with 400 vertices, where every possible edge between vertices is included in the graph with probability $\frac{1}{2}$ (both directions are considered separately). This random instance is repeatedly solved by a simple version of the *tabu search* method [4] starting from different initial solutions, terminating when the algorithm finds a solution whose objective is better than a predefined threshold value. The implementation of the tabu search in this example has a fixed tabu tenure of ten iterations, and it does not include any advanced features, except for the cycle detection [4].

The distribution of the run-times is presented in Fig. 1. Markedly, it can be observed that the probability of large computational times is rather high or, in other words, the run-time has a heavy-tailed distribution (the tails are not exponentially bounded). In order to accelerate the search, one can select a restart parameter value and repeatedly restart the algorithm after the number of iterations exceeds this predefined parameter. Figure 2 provides the average run times for the tabu algorithm as a function of the number of iterations between restarts. It is clear from the plot that certain values of the restart parameter guarantee superior algorithmic performance, when compared to the original algorithm that does not employ any restart strategy. In this context, an interesting question can be stated: What is the best restart value for algorithm acceleration? The following discussion provides an overview of the research that addresses this important question.

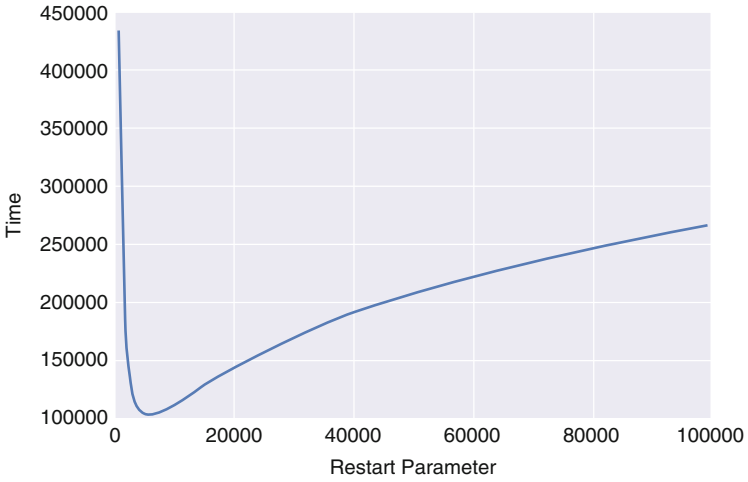


Fig. 2 The average run-time of a tabu search algorithm for an instance of the maximum cut problem as a function of the restart parameter (the same instance as in Fig. 1)

Basic Restart Strategies

Unlike Monte Carlo algorithms, which are guaranteed to provide approximate answers or solutions within a fixed time, the Las Vegas-type algorithms are defined as algorithms that always find "correct solutions," but the required computational time is uncertain. To define the correctness of a solution, one can demand a proof of optimality or comparison of the solution objective function to a predefined threshold, in which case the "correct solution" is any solution with the objective function better than the threshold.

In general, the run-time of an algorithm can be measured either in iterations or in computational time. A *restart strategy* associated with a given algorithm is typically defined (see, e.g., [10, 22]) as a sequence, $S = \{t_1, t_2, \ldots\}$, that represents allocated run-times between restarts. According to the strategy S, the first run continues for t_1 time units (iterations) or until the correct answer is found. The second run lasts t_2 time units or until the correct answer is found and so on. The runs are independent from each other, as no information is passed between different computational runs. As briefly discussed in section "Introduction", different restart strategies might lead to different average run-times.

In [10], the authors consider integer run-times that represent the total number of iterations until a desired outcome is achieved. They consider a wide domain of restart strategies including strategies with random restart intervals and those that enable suspension of algorithm execution in favor of another run with an option to continue its execution later. It is shown that there always exists a *uniform restart strategy* (i.e., $t_1 = t_2 = \ldots = t^*$) that is optimal. It is easy to see

that the same result holds if a run-time is determined using continuous time units instead of iterations – for example, consider rounding continuous times to the nearest integral values (number of seconds) and redefining the run-time distribution accordingly.

To find an optimal restart period t^*, a good estimate of the underlying run-time distribution is required, which might be problematic in practice. If this information is not available, the *universal restart strategy* introduced in [10] can provide a decent performance guarantee. In this strategy, all run length are powers of 2 starting with a run of length 2^0 (this sequence may be rescaled). Considering the duration of the current run, if the number of previous runs with the same length is even, then the next run duration is double of the current run duration. Otherwise, the next run has the length equal to one. These rules define the following numerical sequence:

$$S^{\text{univ}} = \{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots\}$$

When comparing S^{univ} to the optimal restart strategy, its performance is within a logarithmic factor from optimality without any requirements on prior knowledge of the run-time distribution [10]. Formally, if T^{univ} denotes the expected run-time of the algorithm that uses universal restart strategy and T^{opt} denotes the expected run time under the optimal restart strategy, then

$$T^{\text{univ}} \leq 192 \cdot T^{\text{opt}}(\log_2(T^{\text{opt}}) + 5).$$

Unfortunately, the difference between run-times of the universal and optimal strategies may be large in practice, which limits the applicability of this important theoretical result.

Restart strategies can be naturally extended to parallel optimization [9], where copies of the same algorithm are executed in parallel. Note that due to randomness, the search trajectories are in general different. As in the serial setting, a restart strategy defines maximum run-times for each copy of the algorithm, i.e.,

$$S = \begin{cases} t_1^1, t_2^1, t_3^1, \ldots \\ t_1^2, t_2^2, t_3^2, \ldots \\ \cdots \\ t_1^n, t_2^n, t_3^n, \ldots \end{cases} \tag{1}$$

In a uniform parallel restart strategy, each copy of the algorithm has the same restart schedule (i.e., $t_j^i = t^*$). This strategy is no longer optimal as it was in the serial case, but its performance is provably within a constant factor from optimality. An example of nonuniform optimal strategy can be found in [9].

Restart Distribution and Optimal Restart Strategies

Next, we overview some formal characterizations of run-time distribution that can be exploited to gain computational acceleration. Our discussion in this and the next sections is mostly based on the results from [22].

Consider a randomized algorithm A for solving a problem instance P. The execution time of A when solving P is a random variable, ξ. Based on the algorithm A, one can define its uniform restart version, A_R, which follows the same algorithmic steps as A, but is restarted after R iterations or time units if the correct solution is not found. Similarly to A, algorithm A_R terminates as soon as the correct solution is obtained. The positive parameter R is referred to as a *restart period* (or a *restart parameter*) of A_R.

The number of restarts before the first successful run of A_R is a geometric random variable, and the expected run-time, $T(R)$, of A_R can be expressed as

$$T(R) = R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] \tag{2}$$

The first part of (2) is an expected number of restarts multiplied by the duration of each run, while the second is an average duration of the run that produces the final solution.

Definition 1. A probability distribution of a random variable ξ, $Pr\{\xi \leq x\}$, is called a *restart distribution* if there exists $R > 0$, such that

$$R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] < E[\xi] \tag{3}$$

From (3), if a run-time of an algorithm follows a restart distribution, then the algorithm has a restart version that outperforms the algorithm without restarts in terms of its average run-time. Clearly, the properties of the run-time distribution depend on the optimization problem and the algorithm itself. If, for example, an optimization algorithm uses an optimal restart policy, then its run-time will no longer follow a restart distribution.

A log-normal distribution is an example of a restart distribution. For example, consider three log-normal random variables with the same location parameter, $\mu = 1$. The scale parameters, σ, are 2, 2.5, and 1.5. The expected values of these random variables are 20.08, 61.86, and 8.37, respectively. If these variables represented run-times of some algorithms, we would be able to achieve acceleration by restarting after R units of run-time. Figure 3 illustrates this opportunity by showing the expected run-time given by (2) as a function of restart period R. It shows that restarting an algorithm allows us to achieve significant acceleration for these run-time distributions. Furthermore, we can see that the larger variability in run-times provides a better opportunity for acceleration by the restart mechanism.

The optimality conditions can be derived for continuous and differentiable (first and second derivative) cumulative distribution functions (CDF) of run-times

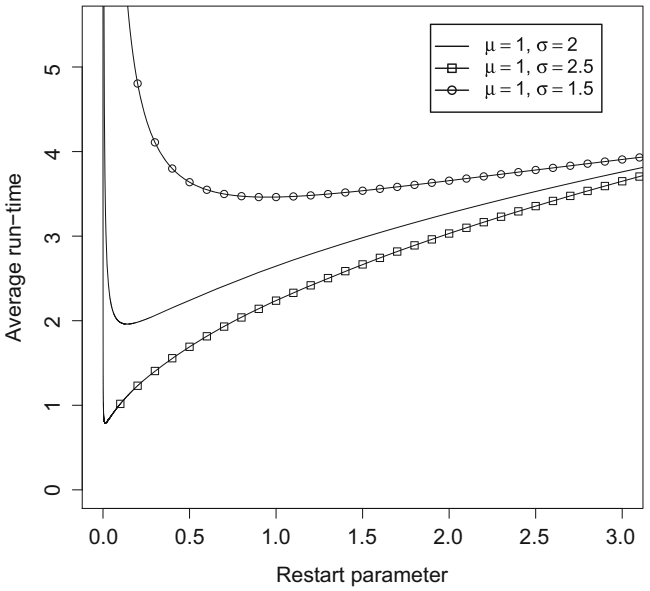


Fig. 3 The average run-time as a function of the restart parameter, R, for log-normally distributed run-times

(see [22] for more details). If this is not the case, then a continuous and differentiable approximation of original CDF can be used instead.

Proposition 1. *Let R^* be an optimal restart period for A_R. Then the expected run-time of A_{R^*} is*

$$T(R^*) = \frac{1 - Pr\{\xi \leq R^*\}}{\frac{dPr\{\xi \leq R\}}{dR}\Big|_{R=R^*}} \tag{4}$$

Proof.

$$\begin{aligned}
\frac{dT(R)}{dR} &= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + R \cdot \left(\frac{-\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} - \frac{1 - Pr\{\xi \leq R\}}{(Pr\{\xi \leq R\})^2} \cdot \frac{dPr\{\xi \leq R\}}{dR} \right) \\
&\quad + R \cdot \frac{dPr\{\xi \leq R\}}{dR} \frac{1}{Pr\{\xi \leq R\}} + \frac{-\frac{dPr\{\xi \leq R\}}{dR}}{(Pr\{\xi \leq R\})^2} \int_0^R x f(x) dx \\
&= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} - \frac{\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} \left(R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] \right) \\
&= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} - \frac{\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} \cdot T(R)
\end{aligned}$$

□

The reciprocal of the expression in the right-hand side of (4) is known as a *hazard rate function*, which is an important concept in reliability engineering [17]. The hazard rate function, or failure rate, can be used to describe general properties of restart distributions.

Proposition 2. *A hazard rate function of a restart distribution is nonincreasing on some interval containing an optimal restart period.*

This property allows us to rule out the distribution that has an increasing hazard rate functions [7]. The previous proposition shows, for example, that the following distributions are not restart distributions:

- Weibull distributions with the shape parameter $k > 1$,
- Gamma distribution with the shape parameter $k > 1$,
- Uniform distribution,
- Normal distribution.

Single Algorithm Portfolios of Restart Algorithms

Algorithmic acceleration can be achieved by combining algorithms in portfolios that are executed concurrently in a distributed manner. For example, a *single algorithm portfolio* consists of different copies of the same algorithm that are deployed on different processors. This simple parallelism can be extremely efficient in practical applications that involve randomized algorithms. There are numerous reports of superlinear speedup when using this type of parallelization (e.g., see [15]). By *superlinear speedup*, we imply that the algorithm utilizes up to n processors and on average is more than n time faster than the algorithm utilizing one processor.

The superlinear speedup can be related to the concept of restart distributions. Consider a single restart algorithm portfolio: a parallel algorithm A_R^n that consists of n independent copies of A_R running in parallel (no communication), where, as previously, A_R is the restart version A with parameter R. The algorithm A_R^n halts whenever one of n copies finds a correct solution. Let random variable $\xi_{\min}^n$ denote the run-time of A_R^n, while $T_n(R)$ denotes the expected run-time of A_R^n.

The expected run-time of the single algorithm portfolio with a uniform restart strategy is given by

$$T_n(R) = R \cdot \frac{(1 - Pr\{\xi \leq R\})^n}{1 - (1 - Pr\{\xi \leq R\})^n} + E[\xi_{\min}^n | \xi_{\min}^n \leq R] \tag{5}$$

An optimality condition similar to the serial case can be derived for parallel restart algorithms (see [22]).

Proposition 3. *Let R_n^* be the optimal restart period for A_R^n; then the expected running time of A_R^n is*

$$T_n(R_n^*) = \frac{(1 - Pr\{\xi \leq R_n^*\})}{n \cdot \frac{dPr\{\xi \leq R\}}{dR}\Big|_{R=R_n^*}} \tag{6}$$

As mentioned earlier, the log-normal distribution is a restart distribution. We use this distribution to illustrate some of the ideas, since we can easily calculate the exact values for the conditional expectations in (5) without resorting to statistical sampling. For example, we can use the log-normal run-time to provide an illustration of the superlinear speedup.

Let ξ be a random variable with a log-normal distribution with parameters $\sigma = 2$ and $\mu = 1$. We define a *parallel speedup* as a ratio between run-time of a serial algorithm and run-time of its parallel version. Firstly, we consider the speedup for the algorithm that does not apply any restart strategy. The data presented in Fig. 4 clearly indicates the superlinear average speedup relative to the run-time of the serial algorithm.

The following proposition from [22] states that if the superlinear parallel speedup is achieved by a single algorithm portfolio, then the underlying distribution of the run-time is a restart distribution.

Proposition 4. *If R^* and R_n^* are optimal restart periods for A_R and A_R^n, respectively, then $\frac{T(R^*)}{T_n(R_n^*)} \leq n$.*

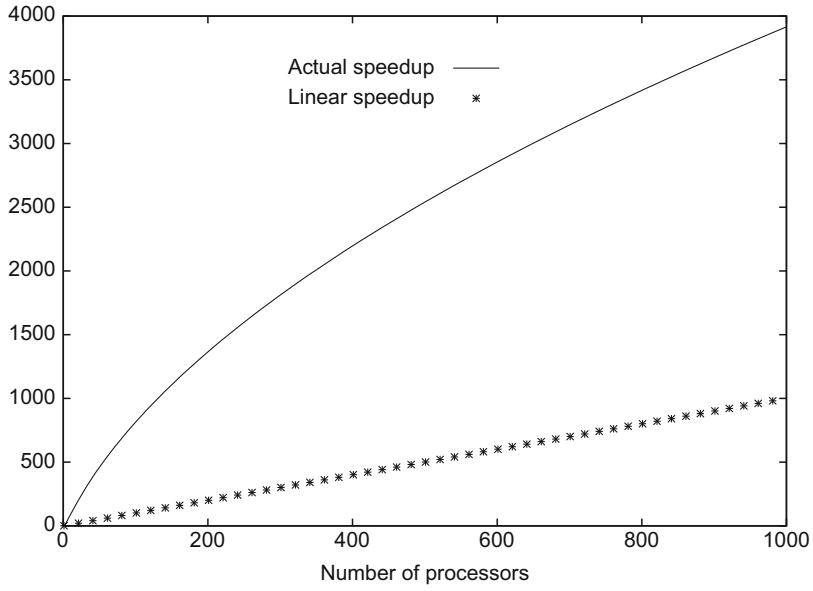


Fig. 4 Speedup is obtained by comparing the **serial no-restart** version with the **parallel no-restart** version [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

Simply speaking, Proposition 4 indicates that the superlinear parallel speedup can be attributed to the inefficiencies of the serial algorithm, and these inefficiencies can be alleviated by adopting an appropriate restart strategy.

Consider now the situation when both serial and parallel algorithms implement optimal restart strategies. It is important to note that the optimal values in parallel and serial cases are not necessarily the same. The following proposition relates the number of processors to the value of the optimal restart parameter.

Proposition 5. *If the hazard rate function of the run-time distribution is unimodal and* $\frac{T(R^*)}{T_n(R_n^*)} < n$, *then* $R^* < R_n^*$.

Intuitively, it might seem that the best serial algorithm is the best candidate for parallelizing. However, Proposition 5 shows that the optimal restart parameter for the serial algorithm is not necessarily optimal when considering the performance of the corresponding single algorithm portfolio. In other words, good average performance in serial setting can often provide suboptimal parallel performance.

Suppose that the run-time of a serial algorithm follows the same log-normal distribution as in the previous example. The optimality conditions (4) and (6) for the optimal restart parameters provide the corresponding optima. Figure 5 illustrates the computational speedup achieved by the parallel algorithms with optimal restart parameters and varying number of parallel processors compared to the serial algorithm (for the log-normal run-times). The restart parameters of the serial and parallel algorithms are different: in fact, it turns out that the optimal restart parameter is monotonically increasing as a function of the number of processors.

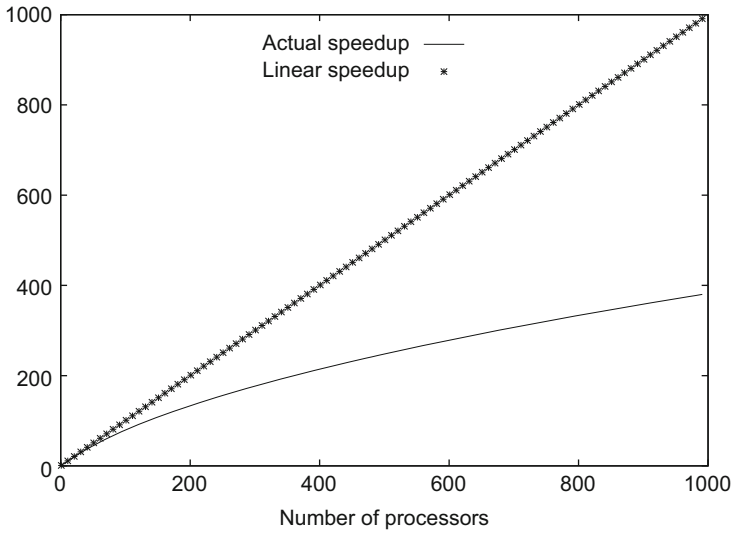


Fig. 5 Speedup is obtained by comparing the **optimal serial restart** version with the **optimal parallel restart** version [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

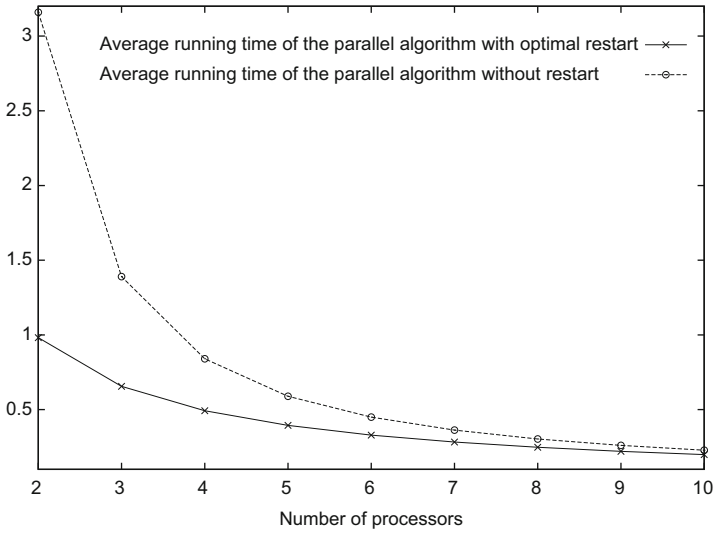


Fig. 6 Comparison of the parallel algorithms with and without implementation of the optimal restart strategy [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

The speedup is almost linear for the number of processors below 100. For large numbers of processors, the speedup quickly becomes sublinear, and the single algorithm portfolio parallelization becomes less effective.

Another interesting question that is important for practical applications is: what is the value of knowing the optimal restart parameter? Figure 6 shows the value of implementing optimal restart strategy with respect to the number of processors. As the number of processors increases, the benefit of knowing the exact value of optimal restart parameter is steadily decreasing. In some sense, the single algorithm portfolio framework takes advantage of restart-distribution properties without restarting the algorithm. The performance can be improved by applying an effective restart strategy explicitly; however, the value of such improvement quickly diminishes as the number of processes increases.

An optimal value of restart parameter is typically unknown in a realistic problem setting. Furthermore, as mentioned previously, the value of the optimal restart parameter may decrease as the number of processors grows. These observations suggest that the main effort should be concentrated on developing and identifying algorithms with run-times that follow restart distributions instead of focusing on the methods that find optimal restart values. The potential for acceleration of run-times that follow restart distributions can be automatically exploited via portfolio parallelization. Furthermore, any errors in estimating the exact value of the optimal restart sequence can degrade the performance with respect to adopting no-restart strategy.

We illustrate this idea by looking at different tenure parameters of tabu search. Again we consider a random instance of the maximum directed cut problem with 400 vertices. The computational experiment is based on a simple tabu search with

Table 1 Average number of iterations for different number of processors

Number of processors	Average number of iterations	
	Tabu tenure = 10	Tabu tenure = 30
1	192,216	26,073
2	19,950	13,264
4	1049	6996
8	434	3664
16	433	2003
32	325	1124

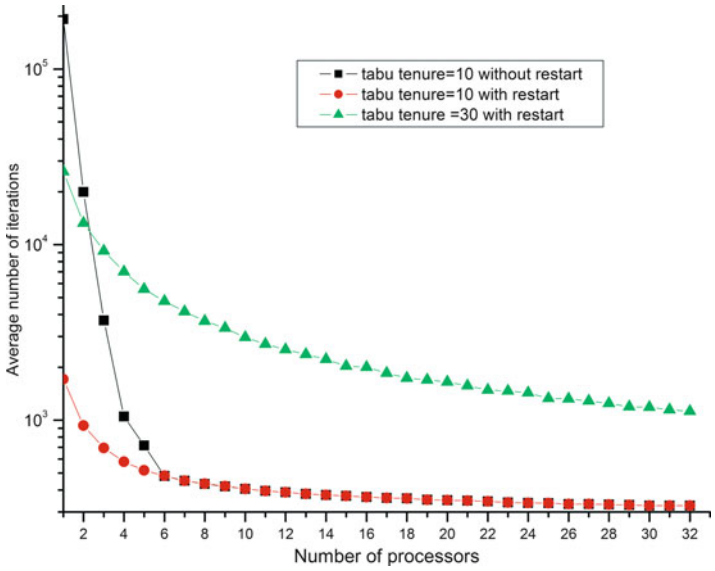


Fig. 7 Computational results for MAXDICUT problem using tabu search in a single algorithm portfolio [19]

different values of tabu tenure that was mentioned in earlier examples. Table 1 shows the results for two tabu algorithms with different tenure parameters. The algorithm with tenure parameter 30 is significantly faster than the algorithm with tenure parameter 10 when deployed on a single processor. However, the situation changes when parallel implementation is considered: the parallel algorithm with tenure 10 is much faster than its serial version. This example (see [19]) shows that if one uses the average serial run-time as the main criterion for choosing an algorithm for parallel implementation, the better parameter choices (or even algorithmic approaches) can be dropped in favor of suboptimal choices.

Figure 7 highlights the value of knowing the optimal restart parameter. The tabu algorithm with tenure 10 without restarts quickly converges to the performance of the same tabu algorithm that uses an optimal restart strategy.

Mixed Algorithm Portfolios of Restart Algorithms

Instead of focusing on a single algorithm portfolio, one may consider a combination of different algorithms. The diversity of mixed algorithms can improve the overall performance with respect to a single algorithm portfolio. Suppose that we have a set of available randomized algorithms and we want to select a subset of them to include into a mixed algorithm portfolio of a given size. If the available algorithms are simply different copies of the same algorithm, we call such selection a single algorithm portfolio; otherwise, we will refer to it as a *mixed algorithm portfolio*.

There are a number of examples in the literature of mixed algorithm portfolios that outperform single algorithm portfolios. In particular, the algorithm portfolio approach for constraint satisfaction and mixed integer programming is presented in [5]. The authors show that the mixed algorithm portfolio can outperform a single algorithm portfolio and discuss intuition behind such situations. An efficient algorithm portfolio approach using backtracking search for the graph-coloring problems is considered in [8]. Extensive computational experiments with restart strategies and algorithm portfolios for benchmark instances of network design problem are also investigated in [1].

The mathematical model of mixed portfolios of restart algorithms can be outlined as follows. Consider a set of m algorithms $A_1, \ldots, A_m$ with restart parameters $R_1, \ldots, R_m$ and random run-times $\xi_1, \ldots, \xi_m$, respectively. Additionally, there are N parallel processors that are available, and we need to select N algorithms to deploy on each processor. Each processor should be used by a single algorithm, and the same algorithm can be deployed on multiple processors. We assume that the run-time of each algorithm is an integer multiple of its restart parameter. In other words, even if an algorithm finds a solution in the beginning of the run, an actual run-time will be rounded up to the next restart period.

Using this setup, there are m single algorithm portfolios that can be formed. Let T_s denote the average run-time of the best single algorithm portfolio, which can be easily defined using the properties of the geometric distribution:

$$T_s(A_1, \ldots, A_m, N) = \min \left\{ R_1 \frac{1}{1 - p_1^N}, \ldots, R_m \frac{1}{1 - p_m^N} \right\}.$$

Let $T_m(A_1, n_1, \ldots, A_m, n_m)$ denote the expected run-time of the mixed algorithm portfolio, which consists of n_1 copies of A_1, n_2 copies of A_2, and so on $\left(\sum_{i=1}^{m} n_i = N\right)$. The mixed algorithm portfolio terminates as soon as one of the algorithms finds the target solution (e.g., a solution with objective below a certain threshold).

To identify the computational benefit that can be achieved by mixing randomized algorithms with different properties, we define the *speedup ratio* S as

$$S = \frac{T_s(A_1, \ldots, A_m, n_1 + \ldots + n_m)}{T_m(A_1, n_1, \ldots, A_m, n_m)}. \tag{7}$$

Unlike the serial case, in [9] the authors demonstrate that the best uniform restart strategy repeated on every processor is not necessarily optimal; however, its performance is within a constant factor of the optimal strategy. Furthermore, in [21] and the subsequent work in [11], it is shown that the speedup ratio S of any mixed algorithm portfolio satisfies

$$S \leq \frac{1}{1 - e^{-1}} \approx 1.58. \tag{8}$$

Therefore, if one has a full knowledge of run-time distributions, the best mixed algorithm portfolio is less than two times faster than the best single algorithm portfolio. However, a mixed algorithm portfolio can be viewed as a strategy that can reduce risks associated with a nonoptimal algorithm selection for single algorithm portfolios. Recall that by the definition in (7), the value of S compares the performance of a mixed algorithm portfolio against the best possible algorithm portfolio.

Moreover, it is interesting to note that according to the theoretical approach described above (see derivation details of (8) in [11, 21]), the best performance of a mixed algorithm portfolio is achieved when it consists of N algorithms that can also be used as *candidates to form the best single algorithm portfolio*. Thus, the expected performance of these algorithms in the case of a single algorithm portfolio is the same. However, for the mixed algorithm portfolio, one should select $N - 1$ algorithms with a relatively short restart parameter and exactly one algorithm with a long restart parameter. As all of these algorithms have the same performance in the single portfolio setting, it also implies that each of the former $N - 1$ algorithms is relatively unreliable within its short restart period (thus, these algorithms have to be restarted a large number of times), while the remaining N-th algorithm, despite its large restart parameter, is very reliable (i.e., the corresponding p is close to one). Clearly, the existence of such algorithms is not necessarily guaranteed. Nevertheless, the above result provides an intuitive characterization of effective mixed algorithm portfolios. Namely, if there exists a trade-off that involves the restart parameter value and the algorithm reliability, then it can be exploited within a mixed algorithm portfolio.

Conclusions

In the discussion above, we only consider a setting with a single problem instance. However, the same framework can be easily extended to the setting with multiple problem instances. For example, optimization models of daily locomotive scheduling remain constant, i.e., one needs to find optimal routes using existing railroad network for a given demand. The particular demand patterns can vary substantially from day to day producing different problem instances. We can form a training set by selecting a set of problem instances from historical demands and use them for construction of optimal restart strategies and/or algorithm portfolios. This is also

a typical approach for computational experiments. The researchers often test their techniques on a small set of instances, tune the algorithms, and then conduct the final experiment on a larger set of problems.

Cross-References

- ▶ Matheuristics
- ▶ Multi-start Methods
- ▶ Tabu Search

References

1. Chabrier A, Danna E, Pape CL, Perron L (2004) Solving a network design problem. Ann Oper Res 130:217–239
2. Chen H, Gomes CP, Selman B (2001) Formal models of heavy-tailed behavior in combinatorial search. In: CP '01: proceedings of the 7th international conference on principles and practice of constraint programming. Springer, London, pp 408–421
3. D'apuzzo MM, Migdalas A, Pardalos PM, Toraldo G (2006) Parallel computing in global optimization. In: Kontoghiorghes E (ed) Handbook of parallel computing and statistics. Chapman & Hall/CRC, Boca Raton, pp 225–258
4. Glover F, Laguna M (1997) Tabu Search. Kluwer Academic, Norwell
5. Gomes CP, Selman B (2001) Algorithm portfolios. Artif Intell 126(1–2):43–62
6. Gomes CP, Selman B, Kautz H (1998) Boosting combinatorial search through randomization. In: Proceedings of the 15th national conference on artificial intelligence. AAAI Press, Madison, pp 431–437
7. Gupta AK, Zeng WB, Wu Y, Gupta AK, Zeng WB, Wu Y (2010) Parametric families of lifetime distributions. In: Gupta AK, Zeng W-B, Wu Y (eds) Probability and statistical models. Birkhäuser, Boston, pp 71–86
8. Huberman BA, Lukose RM, Hogg T (1997) An economics approach to hard computational problems. Science 275(5296):51–54
9. Luby M, Ertel W (1994) Optimal parallelization of Las Vegas algorithms. In: Proceedings of the 11th annual symposium on theoretical aspects of computer science, STACS '94. Springer, London, pp 463–474
10. Luby M, Sinclair A, Zuckerman D (1993) Optimal speedup of Las Vegas algorithms. Inf Process Lett 47:173–180
11. Mostovyi O, Prokopyev OA, Shylo OV (2013) On maximum speedup ratio of restart algorithm portfolios. INFORMS J Comput 25(2):222–229. https://doi.org/10.1287/ijoc.1120.0497
12. Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. J Sched 8(2):145–159. https://doi.org/10.1007/s10951-005-6364-5
13. Nowicki E, Smutnicki C (2005) Some new ideas in TS for job shop scheduling. In: Operations research/computer science interfaces series, vol 30, Part II. Springer, pp 165–190
14. Palubeckis G, Krivickiene V (2004) Application of multistart tabu search to the max-cut problem. Inf Technol Control 31(2):29–35
15. Pardalos PM, Rodgers GP (1992) A branch and bound algorithm for the maximum clique problem. Comput Oper Res 19:363–375
16. Resende MG, Ribeiro CC (2011) Restart strategies for grasp with path-relinking heuristics. Optim Lett 5(3):467–478
17. Ross SM (1996) Stochastic processes. Wiley, New York

18. Sergienko IV, Shilo VP, Roshchin VA (2000) Restart technology for solving discrete optimization problems. Cybern Syst Anal 36(5):659–666
19. Sergienko IV, Shilo VP, Roshchin VA (2004) Optimization parallelizing for discrete programming problems. Cybern Syst Anal 40(2):184–189
20. Shylo V, Shylo OV (2011) Path relinking scheme for the max-cut problem within global equilibrium search. IJSIR 2(2):42–51
21. Shylo O, Prokopyev O, Rajgopal J (2011) On algorithm portfolios and restart strategies. Oper Res Lett 39(1):49–52
22. Shylo OV, Middelkoop T, Pardalos PM (2011) Restart strategies in optimization: parallel and serial cases. Parallel Comput 37:60–68

Part II
Local Search

Constraint-Based Local Search

9

Laurent Michel and Pascal Van Hentenryck

Contents

Introduction . . . 224
Foundations . . . 225
Getting Started . . . 226
The Problem . . . 226
The Model . . . 227
Foundations . . . 228
Models . . . 229
Programs . . . 235
Case Studies . . . 240
Progressive Party . . . 241
Car Sequencing . . . 245
Scene Allocation . . . 248
Implementation . . . 249
Invariants . . . 250
Differentiation . . . 254
Empirical Results . . . 257
Progressive Party . . . 257
Car Sequencing . . . 257
Scene Allocation . . . 258
Conclusion . . . 258
References . . . 259

L. Michel (✉)
University of Connecticut, Storrs, CT, USA
e-mail: ldm@engr.uconn.edu

P. Van Hentenryck
University of Michigan, Ann Arbor, MI, USA
e-mail: pvanhent@umich.edu

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_7

Abstract

Constraint-Based Local Search emerged in the last decade as a framework for declaratively expressing hard combinatorial optimization problems and solve them with local search techniques. It delivers tools to practitioners that enables them to quickly experiment with multiple models, heuristics, and meta-heuristics, focusing on their application rather than the delicate minutiae of producing a competitive implementation. At its heart, the declarative models are reminiscent of the modeling facilities familiar to constraint programming, while the underlying computational model heavily depends on incrementality. The net result is a platform capable of delivering competitive local search solutions at a fraction of the efforts needed with a conventional approach delivering model-and-run to local search users.

Keywords

Constraint · Local search · Neighborhood · Synthetic search satisfaction · Optimization · Incremental model · Declarative

Introduction

Complete techniques such as Integer Programming and Constraint Programming typically offer optimality guarantees on the results they deliver but do not always scale to large instances. This explains the appeal of local search methods that deliver a different trade-off in the algorithmic design space, favoring scalability at the expense of guarantees. Local search algorithms apply to diverse application domains such as routing, scheduling, resource allocation, or rostering to name just a few. In most cases, local search techniques scale to truly large problem instances that are often out of scope for complete techniques and are capable of producing streams of solutions of improving quality.

Yet, while modeling and high-level tools are ubiquitous for Mixed Integer Programming and Constraint Programming, they have been relatively unexplored for local search until recently. This is primarily due to the fact that the separation of models and algorithms is not as simple in local search compared to MIP and CP which are declarative in nature. Indeed, the vast majority of papers discussing local search describe their solution in *algorithmic* terms rather than *declarative* terms like models, decision variables, and constraints. The 1990s witnessed a shift in interest and the emergence of multiple tools expressly dedicated to local search techniques. GSAT [13, 14] and WalkSAT [15] offered the initial impetus behind formulating problems with a simple language (in clausal form) and using generic local search algorithms operating on that encoding. Integer Optimization by Local Search [25] generalized this line of work to the richer language of integer programming.

Localizer [8,9] took a different approach, providing a first step to build a general-purpose modeling language for local search. It introduced the concept of *invariants* to express arbitrary one-way constraints that are automatically and incrementally updated under variable assignments. These invariants can then encode, in a declara-

tive fashion, the incremental data structures typically needed in the implementation of meta-heuristics such as Tabu Search [5], Min-Conflict Search [10], Simulated Annealing [6], Scatter Search [7], or GRASP [4] to name just a few.

Constraint-Based Local Search is the culmination of this line of work. It complements the constraint programming efforts typically focused on complete search techniques and delivers a "model and search" framework in which one uses decision variables and constraints to model a problem and relies on search procedures to explore the underlying search space. COMET is an optimization platform that embodies Constraint-Based Local Search and is a direct descendant of LOCALIZER. It is an object-oriented programming language with explicit support for modeling problems declaratively and solving them with local search techniques. The contributions underlying COMET span from the incremental computation model, the modeling abstractions, and the control mechanisms to specify and execute local search heuristics and meta-heuristics easily, efficiently, and compactly.

Foundations

Constraint-Based Local Search aims at implementing the vision captured by the equation

$$\text{LocalSearch} = \text{Model} + \text{Search}$$

that expresses the belief that a local search algorithm is best viewed as the composition of a *declarative* model *with a search* component. This separation of concerns is central: it postulates the importance of expressing the structures of the problem being solved declaratively and compositionally and providing a search component which *exploits* those structures and guide the search toward high-quality local optima.

The Constraint-Based Local Search architecture delivers several key benefits:

Rich Language Constraints are declarative and capture the problem substructures. They range from simple arithmetic constraints, the indexing of arrays of variables with variables, meta-constraints (constraints on the truth value of other constraints) and logical constraints, to combinatorial constraints such as cardinality, sequence, or alldifferent constraints to name just a few. Constraints (and combinators) for local search were introduced in [24].

Rich Search Programming meta-heuristics is supported by a wealth of language combinators and control abstractions to automate the most tedious and error-prone aspects of an actual implementation. The abstractions foster the decoupling of neighborhood, heuristics, and meta-heuristics specifications while leveraging the incrementality exposed by the constraints present in the declarative model. Control abstractions were introduced in [20].

Separation The untangling of model and search promotes the independent design and evolution of these two components. With Constraint-Based Local Search, it is possible to explore alternative models independently of refinements to the search procedures.

Versatility The ability to specify meta-heuristics orthogonally to the model also enables a collection of *generic* search routines which are highly reusable and promote the experimental process to design an effective CBLS program. This versatility further promotes the reuse of highly generic "canned" search procedures.

Extensibility New constraints and objective functions can be added to the system library and used in conjunction with native constraints. Perhaps even more interestingly, the new constraints can be implemented directly in the host language (i.e., COMET), and the bulk of the implementation is often cast in terms of invariants. New heuristics and meta-heuristics can also be easily added to the system as the implementation of *any* heuristic or meta-heuristic relies on a few key concepts such as closures, continuations, selectors, and neighbors.

Performance Finally, the architecture heavily relies on incrementally maintaining the computational state over time. This capability is compositional and a direct by-product of the declarative model. The approach enables the platform to deliver Constraint-Based Local Search *programs* that are a fraction of the size and complexity of handcrafted code, yet deliver performance comparable, and sometime exceeding, manually crafted implementations.

The rest of this chapter starts with an illustration of Constraint-Based Local Search through the modeling and resolution of the classic $n-$queens problem in section "Getting Started." Section "Foundations" discusses the theoretical underpinnings of Constraint-Based Local Search starting with models and concluding with programs. Section "Case Studies" focuses on how to model applications with a rich language that goes beyond Boolean formulas or linear equations. Section "Implementation" explores the implementation issues, starting with the support for incremental computation through invariants and proceeding with a discussion of differentiation. Section "Empirical Results" gives a brief survey of the type of performance that can be expected from Constraint-Based Local Search systems, and section "Conclusion" concludes the chapter.

Getting Started

To introduce Constraint-Based Local Search, it is valuable to start with the modeling and resolution of a simple well-known problem. The examples presented in this chapter uses the COMET platform.

The Problem

The $n-$queens problem is to place n queens on a chess board of size $n \times n$ so that no two queens can attack each other. While the problem is polynomial, its simplicity is appealing to illustrate Constraint-Based Local Search. The COMET program is shown in Fig. 1 and features two clear components that are discussed next.

```
1 import lssolver;
2 int n = 8;
3 range Size = 1..n;
4 Solver<LS> m();
5 ConstraintSystem<LS> S(m);
6 var{int} queen[Size](m,Size);
7 S.add(alldifferent(queen));
8 S.add(alldifferent(all(i in Size) (queen[i] + i)));
9 S.add(alldifferent(all(i in Size) (queen[i] - i)));
10
11 while(S.violations() > 0)
12   selectMin(i in Size,v in Size: queen[i]!=v)(S.getAssignDelta(queen[i],v))
13     queen[i] := v;
```

Fig. 1 A Constraint-Based Local Search model for the queens problem

The Model

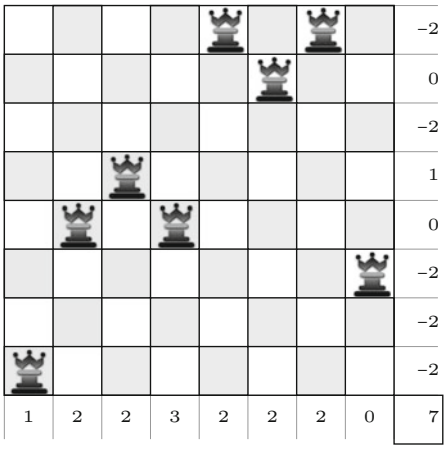
The model definition spans lines 4–9, is declarative, and exclusively focuses on defining an array of decision variables queen in line 6 and three combinatorial constraints on lines 7–9. Variable queen_i models the row on which the queen in column i is to be placed. Each decision variable has a domain $Size$ representing the set of permissible values for that variable. Each combinatorial `alldifferent` constraint takes as input an array of expressions and requires that the values of all entries in the array be pairwise distinct. For instance, line 7 states that no two queens can be assigned to the same row. The constraints on lines 8 and 9 play a similar role but for the upward and downward diagonals. Note how an operator `all` is used on line 8 to create an array of expressions

$$(\text{queen}_1 + 1, \text{queen}_2 + 2, \text{queen}_3 + 3, \cdots, \text{queen}_7 + 7, \text{queen}_8 + 8)$$

which is the input of the combinatorial constraint.

The Search

A classic Constraint-Based Local Search starts from a tentative assignment of values to the decision variables and iteratively transforms it, moving from tentative assignments to tentative assignments using a *local* move operator. The local move used in this specific case is the assignment of a single variable to a new value. Namely, with n decision variables, each with a domain of size n, one could consider up to $\Theta(n^2)$ such moves. To illustrate the search process, consider the left board in Fig. 2a which depicts a tentative assignment. Here, a move consists of relocating a queen to a new row and the algorithm chooses the best such move. The right side, Fig. 2b, shows the reduction in violations for each possible move. Therefore, a viable move is to relocate queen_4 to row 3, reducing the total number of conflicts (violations) from 7 to 5.



(a) Starting point.

0	−1	0	−2	0	0	0	2
1	−1	−1	0	0	0	0	2
−1	−1	−1	−2	0	−1	−1	0
1	0	0	1	0	0	−1	2
1	0	0	0	−1	−1	0	1
1	0	1	−2	0	−1	−1	0
0	−1	−2	−2	−2	−1	−1	0
0	−1	−1	−2	0	0	0	1

(b) Delta Matrix.

Fig. 2 The first step of the CBLS algorithm for the queens problem. (**a**) Starting point. (**b**) Delta matrix

The search outlined above is a classic greedy search. The implementation spanning lines 11–14 obtains the set of constraints present in the model (line 11) and starts from a randomly initialized tentative assignment σ_0. It proceeds through a sequence of moves aimed at satisfying the three softened constraints present in S. The underlying neighborhood function N defined as

$$N(\sigma) = \{\sigma' \mid \exists j \in 1 \dots 8 \; \forall i \in 1 \dots 8, i \neq j : \sigma'(\text{queen}_i) = \sigma(\text{queen}_i) \wedge \\ \sigma'(\text{queen}_j) \in D(\text{queen}_j) \setminus \{\sigma(\text{queen}_j)\}\}$$

is implemented with the `select` statement on line 12 that considers the reassignment of a single decision variable at a time. Overall, the search produces a sequence of tentative assignments

$$\sigma_0, \sigma_1 \in N(\sigma_0), \sigma_2 \in N(\sigma_1), \cdots, \sigma_k \in N(\sigma_{k-1})$$

delivering a solution σ_k that satisfies all the softened constraints (violations of S are 0).

Foundations

This section reviews the foundations of Constraint-Based Local Search, covering both models and programs. It introduces the main concepts behind the declarative and operational models.

Models

From an end-user perspective, constraints and objective functions are at the heart of Constraint-Based Local Search. They provide the declarative bricks needed to construct models describing requirements and properties of the solution being sought. They also support the underlying computational model. This section first reviews the key concepts of constraints and objective functions before finishing with a presentation of the underlying semantics.

Basics

Constraint-Based Local Search is the process of looking for an assignment of values to decision variables that meets specific requirements. Decision variables are central to the modeling process as they characterize solutions. This chapter focuses on integer variables only for simplicity. Extensions to more complex variables (i.e., set, paths, and trees) have been proposed in [12, 23]. The chapter also assumes that Constraint-Based Local Search models are defined over a set X of decision variables.

Definition 1 (Assignment). An assignment σ is a mapping $X \rightarrow D(X)$ from variables to values in their domain. The value assigned to x in σ is denoted by $\sigma(x)$. The set of all possible assignments is denoted by Σ.

For convenience, the expression $\sigma[x/v]$ denotes a new assignment σ' which is similar to σ except that variable x is assigned to v, i.e.,

$$\forall y \in X \setminus \{x\} \; : \; \sigma'(y) = \sigma(y) \wedge \sigma'(x) = v.$$

Constraints are used to impose requirements on the decision variables. Constraints are naturally declarative, can be expressed in a variety of ways, and always capture a relation over a subset of variables from X.

Definition 2 (Constraint). A constraint $c(x_1, \cdots, x_n)$ is a $n-$ary relation over variables $x_1 \cdots x_n \in X$. The set vars(c) is the set of n variables appearing in c, i.e., vars$(c) = \{x_1, \cdots, x_n\}$.

Constraints can be expressed through algebraic expressions, logical statements, or combinatorial structures.

Example 1. In the queens example, the requirement that any two variables cannot lay on the same row, i.e., `alldifferent(x)`, is semantically equivalent to the conjunction of constraints

$$\bigwedge_{i \in 1...8, j \in i+1...8} x_i \neq x_j.$$

Operationally, however, the `alldifferent` maintains its state and violations more efficiently than the naive reformulation above.

Example 2. A constraint system S is a set of constraints and its truth value is equivalent to the truth value of $\wedge_{c \in S} c$.

Evaluations and Violations

The driving force behind Constraint-Based Local Search rests on the ability to assess how badly constraints are violated. This is captured by the concept of violation degrees (or violations for short).

Definition 3 (Violation Degree). The violation degree of $c(x_1, \cdots, x_n)$ is a function $v_c : \Sigma \rightarrow \mathbb{R}^+$ such that, for any assignment σ such that $\sigma(x_1) \in D(x_1), \cdots, \sigma(x_n) \in D(x_n)$, it holds that

$$c(\sigma(x_1), \cdots, \sigma(x_n)) \equiv v_c(\sigma) = 0.$$

The violation degree definition depends critically on the structure conveyed by constraint c. The definition of violation is constraint-dependent but is derived systematically for algebraic and logical expressions. For completeness, consider the specification of expression evaluation.

Definition 4 (Expression Evaluation). Let $e \in \mathcal{E}$ be an arithmetic expression and $\sigma \in \Sigma$ be an assignment. The evaluation of e with respect to (wrt) σ is specified by the function $\mathbb{E}(\sigma, e) : \mathcal{E} \times \Sigma \rightarrow \mathbb{Z}$ which is defined inductively on the structure of e in Fig. 3.

Algebraic *constraints* are based on relations and logical combinators. To reason about the violations of relational and logical constraints, it is useful to derive an *expression* modeling the violations of constraint c from the structure of c.

Definition 5 (Constraint Conversion). Let c be a logical or relational constraint. The constraint conversion of c is an expression $\mathbb{V}(c)$ which can be evaluated to determine the violations of c with respect to some assignment. It is specified by the function $\mathbb{V}(c) : \mathcal{E} \rightarrow \mathcal{E}$ which is defined by induction on the structure of c as specified in Fig. 4.

Fig. 3 The evaluation of an expression with respect to an assignment

$$\begin{aligned}
\mathbb{E}(\sigma, v) &= v \Leftrightarrow v \in \mathbb{R} \\
\mathbb{E}(\sigma, x) &= \sigma(x)(x \in X) \\
\mathbb{E}(\sigma, e_1 + e_2) &= \mathbb{E}(\sigma, e_1) + \mathbb{E}(\sigma, e_2) \\
\mathbb{E}(\sigma, e_1 - e_2) &= \mathbb{E}(\sigma, e_1) - \mathbb{E}(\sigma, e_2) \\
\mathbb{E}(\sigma, e_1 \times e_2) &= \mathbb{E}(\sigma, e_1) \times \mathbb{E}(\sigma, e_2) \\
\mathbb{E}(\sigma, \textsc{Abs}(e)) &= \textsc{Abs}(\mathbb{E}(\sigma, e)) \\
\mathbb{E}(\sigma, \min(e_1, e_2)) &= \min(\mathbb{E}(\sigma, e_1), \mathbb{E}(\sigma, e_2))
\end{aligned}$$

Fig. 4 The constraint conversion function

$$
\begin{aligned}
\mathbb{V}(e_1 = e_2) &= \textsc{Abs}(e_1 - e_2) \\
\mathbb{V}(e_1 \leqslant e_2) &= \max(e_1 - e_2, 0) \\
\mathbb{V}(e_1 \geqslant e_2) &= \max(e_2 - e_1, 0) \\
\mathbb{V}(e_1 \neq e_2) &= 1 - \min(1, \textsc{Abs}(e_1 - e_2)) \\
\mathbb{V}(r_1 \wedge r_2) &= \mathbb{V}(r_1) + \mathbb{V}(r_2) \\
\mathbb{V}(r_1 \vee r_2) &= \min(\mathbb{V}(r_1), \mathbb{V}(r_2)) \\
\mathbb{V}(\neg r) &= 1 - \min(1, \mathbb{V}(r))
\end{aligned}
$$

The conversion for the conjunction of two constraints is none other than the sum of the converted relation violations.

Example 3. The conversion of $x = 5 \wedge y \neq 10$ where $x, y \in X$ is derived as follows:

$$\mathbb{V}(x{=}5 \wedge y \neq 10) {=} \mathbb{V}(x = 5) {+} \mathbb{V}(y \neq 10) = \textsc{Abs}(x{-}5){+}1{-}\min(1, \textsc{Abs}(y{-}10)).$$

It is now possible to compose both functions to obtain the actual violations of an algebraic constraint c with respect to an assignment σ.

Example 4. The violation of an arithmetic constraint $c \equiv l \geq r$ is given by

$$\mathbb{E}(\sigma, \mathbb{V}(l \geq r))$$

The violation degree function for c is then given by

$$v_c(\sigma) = \mathbb{E}(\sigma, \textsc{Abs}(r - l)) = \max(\mathbb{E}(\sigma, r) - \mathbb{E}(\sigma, l), 0).$$

Example 5. The violation of a conjunction $c = c_1 \wedge c_2$ is defined as

$$v_c(\sigma) = \mathbb{E}(\sigma, \mathbb{V}(c_1 \wedge c_2)).$$

While the above mechanics are appropriate to obtain the violations of algebraic constraints, combinatorial constraints define their own notion of violations that capture the combinatorial substructure at hand. Consider the `alldifferent` constraint again.

Example 6. The violation function of $c = \texttt{alldifferent}(x_1, \cdots, x_n)$ simply counts the number of values used more than once by variables $x_1, \ldots, x_n$. More precisely, given an assignment σ and $\text{vars}(c) = \{x_1, \cdots, x_n\}$,

$$v_c(\sigma) = \sum_{i \in S} \max(0, |\{x_j \in \text{vars}(c) | \sigma(x_j) = i\}| - 1)$$

where $S = \bigcup_{i \in 1 \ldots n} D(x_i)$. Consider the `alldifferent` constraint on the rows for the board in Fig. 2a. The assignment is $\sigma = [1, 4, 5, 4, 8, 7, 8, 3]$: It has 2 variables using value 4 and 2 variables using value 8, giving a violation of 2.

Differentiation

The infrastructure covered so far enables the incremental assessment of the satisfaction, and the number of violations, of a constraint c. However, it does not specify how to assess the impact of a local move on the satisfiability or violations of constraints. *Gradients* are the cornerstone of this process.

Definition 6 (Gradient). Given an assignment σ and an arithmetic expression e, $\uparrow_x (\sigma, e)$ denotes the maximum increase for the evaluation of e over all possible values in $D(x)$ wrt σ, whereas $\downarrow_x (\sigma, e)$ denotes the largest decrease for the evaluation of e over all possible values in $D(x)$ wrt σ, i.e.,

$$\uparrow_x (\sigma, e) = \max_{v \in D(x)} \mathbb{E}(\sigma[x/v], e) - \mathbb{E}(\sigma, e)$$

$$\downarrow_x (\sigma, e) = \mathbb{E}(\sigma, e) - \min_{v \in D(x)} \mathbb{E}(\sigma[x/v], e).$$

Note that gradients are nonnegative in this specification. An efficient implementation can be derived inductively on the structure of expression e. Figure 5 gives an abridged version of such derivation. The definition for $\uparrow_x (\sigma, x)$ is an interesting

$$\uparrow_x (\sigma, e) = \max_{v \in D(x)} \mathbb{E}(\sigma[x/v], e) - \mathbb{E}(\sigma, e)$$

$$\downarrow_x (\sigma, e) = \mathbb{E}(\sigma, e) - \min_{v \in D(x)} \mathbb{E}(\sigma[x/v], e).$$

$$\begin{aligned}
\uparrow_x (\sigma, v) &= 0 \Leftrightarrow v \in \mathbb{R} \\
\uparrow_x (\sigma, y) &= 0 \Leftrightarrow y \in X \backslash \{x\} \\
\uparrow_x (\sigma, x) &= \max_{v \in D(x)} v - \sigma(x) \\
\uparrow_x (\sigma, e_1 + e_2) &= \uparrow_x (\sigma, e_1) + \uparrow_x (\sigma, e_2) \\
\uparrow_x (\sigma, e_1 - e_2) &= \uparrow_x (\sigma, e_1) + \downarrow_x (\sigma, e_2) \\
\uparrow_x (\sigma, \textsc{Abs}(e)) &= \max(\textsc{Abs}(\mathbb{E}(\sigma, e) + \uparrow_x (\sigma, e)), \textsc{Abs}(\mathbb{E}(\sigma, e) - \downarrow_x (\sigma, e))) - \mathbb{E}(\sigma, \textsc{Abs}(e)) \\
\uparrow_x (\sigma, \min(e_1, e_2)) &= \min(\mathbb{E}(\sigma, e_1) + \uparrow_x (\sigma, e_1), \mathbb{E}(\sigma, e_2) + \uparrow_x (\sigma, e_2)) - \mathbb{E}(\sigma, \min(e_1, e_2))
\end{aligned}$$

$$\begin{aligned}
\downarrow_x (\sigma, v) &= 0 \Leftrightarrow v \in \mathbb{R} \\
\downarrow_x (\sigma, y) &= 0 \Leftrightarrow y \in X \backslash \{x\} \\
\downarrow_x (\sigma, x) &= \sigma(x) - \min_{v \in D(x)} v \\
\downarrow_x (\sigma, e_1 + e_2) &= \downarrow_x (\sigma, e_1) + \downarrow_x (\sigma, e_2) \\
\downarrow_x (\sigma, e_1 - e_2) &= \downarrow_x (\sigma, e_1) + \uparrow_x (\sigma, e_2) \\
\downarrow_x (\sigma, \textsc{Abs}(e)) &= \texttt{if } \mathbb{E}(\sigma, e) \geqslant 0 \\
&\quad \texttt{then } \min(\mathbb{E}(\sigma, e), \downarrow_x (\sigma, e)) \\
&\quad \texttt{else } \min(-\mathbb{E}(\sigma, e), \uparrow_x (\sigma, e)) \\
\downarrow_x (\sigma, \min(e_1, e_2)) &= \mathbb{E}(\sigma, \min(e_1, e_2)) - \min(\mathbb{E}(\sigma, e_1) + \downarrow_x (\sigma, e_1), \mathbb{E}(\sigma, e_2) + \downarrow_x (\sigma, e_2))
\end{aligned}$$

Fig. 5 Expression gradients

```
1 interface Constraint<LS> {
2   bool holds();
3   var{int} violations();
4   var{int} violations(var{int} x);
5   int getAssignDelta(var{int} x,int v);
6 }
```

Fig. 6 The constraint interface

base case. Indeed, $\uparrow_x (\sigma, x) = \max_{v \in D(x)} v - \sigma(x)$ which has the effect of picking up the largest increase as the distance between the value currently assigned to x in σ and the largest value of the domain. Similarly, note how $\uparrow_x (\sigma, e_1 - e_2)$ combines the largest increase in e_1 with the largest decrease in e_2.

The next concept, variable violation, is interesting: It captures how many violations can be attributed to a specific variable for a given assignment. Variable violations are specified in terms of gradients.

Definition 7 (Variable Violations). Given a constraint c, the variable violations of c wrt $x \in \text{vars}(c)$ and assignment σ are specified by $\downarrow_x (\sigma, \mathbb{V}(c))$.

The concepts of violation degrees and variable violations are generic and hence they enable the specification of a common API for all constraints. This API makes it possible to implement the slogan

$$\text{LocalSearch} = \text{Model} + \text{Search}$$

mentioned in the introduction. In particular, the API of a constraint is specified in Fig. 6. For instance, the method call `violations()` simply returns the evaluation of $v_c(\sigma)$, while the method call `violations(x)` return $\downarrow_x (\sigma, \mathbb{V}(c))$, for the current variable assignment σ. Finally, the call to method `getAssignDelta(x,v)` returns the variation in violation degree when using the assignment $\sigma[x/v]$ instead of σ, i.e., it returns

$$v_c(\sigma) - v_c(\sigma[x/v]).$$

Combinatorial constraints like `alldifferent` also conform to this interface, and the implementation of the last two methods takes advantage of the constraint semantics to implement the specification incrementally.

Objective Functions

Objective functions play an equally critical role within Constraint-Based Local Search. Objectives provide the necessary mechanics to express and exploit objective functions. Interestingly, once gradients are available, objectives do not add much complexity. Objective functions must conform to the interface depicted in Fig. 7.

```
1 interface Objective<LS> {
2   var{int} evaluation();
3   var{int} increase(var{int} x);
4   var{int} decrease(var{int} x);
5   int getAssignDelta(var{int} x,int v);
6 }
```

Fig. 7 The objective interface

Given the current variable assignment σ, the call `evaluation()` returns $\mathbb{E}(\sigma, e)$, while the call `increase(x)` returns $\uparrow_x (\sigma, e)$ and the call `decrease(x)` returns $\downarrow_x (\sigma, e)$.

Models and Constraint Hardness

From a purely pragmatic and operational standpoint, it is often convenient to partition the actual constraint set C in two

$$C = S \cup R \quad (S \cap R = \emptyset)$$

where R represents a set of *required* but easy to solve constraints and S represents a set of *softened* constraints that are typically much harder to satisfy. The intent is to handle both type of constraints differently. Intuitively, required constraints are always satisfied during the search, while softened constraints may be violated. In the $n-$queens examples introduced earlier, $R = \emptyset$ and all the constraints are softened in S. In general, however, R may contain some constraints that are not worth relaxing in S. The membership in S or R is clearly a design decision for the modeler to consider.

Definition 8 (Constraint Model). A Constraint-Based Local Search model M for a constraint optimization problem is a quintuplet $M = \langle X, D, F, R, S \rangle$ where

- Every $x \in X$ is a decision variable taking its value from $D(x)$,
- F is, without loss of generality, a minimization function,
- R is a set of required constraints (easy to satisfy), and
- S is a set of soft constraints (difficult to satisfy).

Declaratively, the semantics of a Constraint-Based Local Search model is given by the optimization problem

$$\min_{\sigma \in \Sigma} \mathbb{E}(\sigma, F) + \sum_{c \in S} v_c(\sigma)$$

subject to

$$v_c(\sigma) = 0 \quad : \forall c \in R$$

Namely, the objective is to minimize the sum of the violations over the softened constraints and the original objective function subject to the required constraints. In the case of constraint satisfaction, $\mathbb{E}(\sigma, F) = 0$ for every σ as the objective is absent. Note that constraints can be weighted, which makes it possible to balance the two terms of the objective. It is simple to write a combinator to weight a constraint in a generic way [23].

Definition 9 (Feasible Solution). A feasible solution σ to a Constraint-Based Local Search model $M = \langle X, D, F, R, S \rangle$ satisfies

$$\sum_{c \in S} \nu_c(\sigma) = 0.$$

$\mathcal{F}(M)$ denotes the set of feasible solutions to M.

Note that, by definition, every assignment σ also satisfy all the required constraints ($\nu_c(\sigma) = 0 \ : \ \forall c \in R$).

Definition 10 (Optimal Solution). An optimal solution σ^* to a Constraint-Based Local Search model $M = \langle X, D, F, R, S \rangle$ is a feasible solution $\sigma^* \in \mathcal{F}(M)$ such that

$$\forall \sigma \in \mathcal{F}(M) : \mathbb{E}(\sigma^*, F) \leq \mathbb{E}(\sigma, F).$$

Definition 11 (Search Procedure). A search procedure produces a sequence of assignments $\sigma_0, \cdots, \sigma_k$ where $\forall i \in 0..k \ : \ \nu_c(\sigma_i) = 0$ ($c \in R$) and returns σ satisfying

$$\min_{\sigma \in \{\sigma_0, \cdots, \sigma_k\}} \mathbb{E}(\sigma, F) + \sum_{c \in S} \nu_c(\sigma).$$

A Constraint-Based Local Search procedure succeeds if $\sigma \in \mathcal{F}(M)$.

Programs

The model specifies the properties satisfied by assignments appearing in a trace $s_0, s_1, \cdots, s_k$; It does not dictate how the assignments in the trace are generated. This is the prerogative of concrete search procedures which are often viewed as the composition of a *neighborhood* function, a *legality* restriction function, and a candidate *selection* function.

Definition 12 (Neighborhood). The neighborhood of an assignment σ_k, denoted $N(\sigma_k)$, is the set of assignments reachable from σ_k via a local move.

Local moves can be *microscopic* changes to the candidate solution such as the reassignment of a single variable or the swap of the values associated with two distinct variables. In specific domains, e.g., in scheduling, the move can be *macroscopic* and involve changing several variables to capture moves in more complex neighborhood, e.g., moving a task in a job sequence.

Example 7. In the 8−queens example outlined in section "Getting Started", the neighborhood is based on the reassignment of a single queen to a new row. The neighborhood consist of $\Theta(n^2)$ assignments. It could be further restricted to $\Theta(n)$ assignments by only considering the queen appearing in the largest number of conflicts and its possible reassignments. Given a model $M = \langle X, D, 0, \emptyset, S\rangle$, where S is the set consisting of three softened `alldifferent` constraints, the quadratic neighborhood function is

$$N(\sigma_k) = \{\sigma_k[x/v] \mid x \in X \wedge v \in D(x) \setminus \{\sigma_k(x)\}\},$$

while the linear neighborhood function is

$$N(\sigma_k) = \left\{ \sigma_k[x/v] \mid x \in \underset{y \in X}{\text{arg-max}} \downarrow_y (\sigma_k, \mathbb{V}(\underset{c \in S}{\wedge} c)) \wedge v \in D(x) \setminus \{\sigma_k(x)\} \right\}.$$

Definition 13 (Legal Neighbors). The legal neighborhood of an assignment σ_k is a restriction of $N(\sigma_k)$, namely, $L(N(\sigma_k), \sigma_k) \subseteq N(\sigma_k)$, and the legal subset is required to at least satisfy all the required constraints R.

Note that the legal subset of $N(\sigma_k)$ may even be more restrictive and reject neighbors that are feasible with respect to R but fail to exhibit other characteristics. The *full* characterization of the function L is part of the definition of a meta-heuristic. For instance, the tabu meta-heuristic excludes neighbors that were seen in the recent past.

Definition 14 (Neighbor Selection). The selection function S is responsible for choosing the next assignment among legal neighbors. Namely,

$$\sigma_{k+1} = S(L(N(\sigma_k), \sigma_k), \sigma_k) \in L(N(\sigma_k), \sigma_k).$$

For instance, a greedy selection chooses the best neighbor, i.e.,

$$\sigma_{k+1} \in \underset{\sigma \in N(\sigma_k)}{\text{arg-min}}\, \mathbb{E}(\sigma, \mathbb{V}(S))$$

A heuristics or meta-heuristics specifies the three functions N, L, and S which parameterize the computation model.

Control Primitives

To support these abstractions, Constraint-Based Local Search programs rely on a handful of control primitives designed to automate tedious and error-prone implementation details.

While the specification of a program relies on three distinct functions N, L, and S, any implementation concerned with performance produces code that fuses the three functions often degrading the readability and maintainability in the process. Key considerations such as randomization and tiebreaking add another layer of complexity to the code. The implementation of meta-heuristics induces its share of complexity by refining move legality and selection further.

COMET attempts to strike a delicate balance between efficiency, code readability, and ease of maintenance by decoupling these aspects as much as possible. The language introduces *selectors*, *neighbors*, and *randomized choosers* as control abstractions.

Neighborhood Selectors

The concept of *neighborhood selector* is a cornerstone for the specification of complex local searches. A simplified version of its interface is presented below:

```
1 interface Neighborhood {
2   void insert(int q,Closure c);
3   boolean hasMove();
4   Closure getMove();
5 }
```

The key idea is that a neighbor is defined as a pair $\langle q, c\rangle$ consisting of a quality measure q and a closure c. The intuition is that, in general, the closure c defines the move which, when executed, will produce an objective value of q for the resulting assignment. Method `insert` adds a neighbor c of quality q (line 2), method `hasMove` checks whether the neighborhood is nonempty, and method `getMove` returns the selected move.

Concrete implementations of this interface commit to a specific selection policy. For instance, the concrete selector `MinNeighborSelector` implements the neighborhood interface and retains only a neighbor minimizing the quality measure. Namely, if the set of inserted neighbors is $N = \{\langle q_1, c_1\rangle, \cdots, \langle q_n, c_n\rangle\}$, the selector retains *one* neighbor

$$\langle q, c\rangle \in \underset{\langle q_j, c_j\rangle \in N}{\text{arg-min}}\, q_j$$

and produces it as its selection when `getMove` is called. Alternative selectors include `KMinNeighborSelector` that returns one of the top$-k$ best neighbors (k being a parameter of the selector).

```
1 MinNeighborSelector N();
2 while(S.violations()!=0) {
3    forall(i in Size,v in queen[i].getDomain() : queen[i] != v)
4       neighbor(S.getAssignDelta(queen[i],v),N) queen[i] := v;
5    if (N.hasMove()) call(N.getMove());
6 }
```

Fig. 8 An alternative greedy search for $n-$queens

The `neighbor` Control Primitive

Neighborhood selectors work in conjunction with the `neighbor` control primitive. The primitive has the following syntax:

`neighbor`$(\langle expr \rangle, \langle N \rangle)$ $\langle Body \rangle$

where $\langle expr \rangle$ refers to an arbitrary arithmetic expression, $\langle N \rangle$ is an expression referring to a selector, and $\langle Body \rangle$ is a statement (or block of statements). From a semantics standpoint, the control primitive creates a closure c of the body code responsible for producing the assignment σ_{k+1} from the current assignment σ_k. It then associates the closure with a quality measure q and submits the pair $\langle q, c \rangle$ to the selector denoted by N. What makes `neighbor` particularly attractive is the syntactic closeness between the code specifying the quality of the move and the code carrying out the move even if, in practice, there is a strong temporal disconnection between the time when the closure is recorded with the selector and the time it gets executed.

The search procedure in Fig. 8 illustrates an alternative implementation of the greedy search presented in Fig. 1. The loop on lines 3–4 scans all variables and values accumulating the reassignments in the selector N and associating with each one a quality measure based on the delta. The 1-instruction block `queen[i] := v;` on line 4 is automatically wrapped in a closure that is entrusted by `neighbor` to the selector N. The beauty in the construction lies in the ability to easily accumulate in the same selector the union of multiple neighborhoods, all defined with different move operators. This capability will be illustrated in the Progressive Party Problem use case in section "Progressive Party."

Randomized Choosers

To support developers, COMET provides control abstractions to make greedy, semi-greedy, and randomized choices. The abstractions encapsulate the necessary state to deliver independent pseudorandom streams in each such instruction appearing in the program. For instance, the $\Theta(n)$-sized neighborhood that considers reassigning the queen with the most violations to a new row, is easily modeled with instructions in lines 3–7 below.

```
1 MinNeighborSelector N();
2 while(S.violations()!=0) {
3     selectMax[2](i in Size)(S.violations(queen[i])) {
4         forall(v in queen[i].getDomain() : queen[i] != v)
5             neighbor(S.getAssignDelta(queen[i],v),N)
6                 queen[i] := v;
7     }
8     if (N.hasMove()) call(N.getMove());
9 }
```

The bracketed 2 on line 3 simply requests the selector to retain any value i among the best two (according to the violation measure).

Discussion

The Constraint-Based Local Search model appearing in Fig. 1 highlights the key features of the framework. It features a complete separation between a declarative model and a procedural search component. The declarative model relies on combinatorial constraints specifying the properties of solutions, while the search is exclusively devoted to the selection of a heuristic and meta-heuristic. In this model, the search remains problem specific. Yet, both the model and the search can evolve independently. One can add constraints without changing the search or choose a different search strategy without modifying the model. All these characteristics, when blended with the performance of an incremental computation, deliver an appealing architecture for producing local search solutions.

The program in Fig. 1 can still be improved. In particular, it is tempting to provide syntactic sugar to automatically handle the boiler plate code and support the notion of *constraint annotations* to partition the constraint set into soft and required constraints $S \cup R$. The resulting program is shown in Fig. 9. The model m is now attaching a *soft* annotation to each constraint to dictate its addition to S, while line 10 is used to extract the set of soft constraints from m. Perhaps even more interestingly, the adoption of an explicit first-class model concept enables the authoring of completely generic search procedures. Indeed, the code in lines 10–13 can be packaged as the reusable min-conflict procedure depicted in Fig. 10.

Naturally, lines 10–13 from Fig. 9 disappear from the model in favor of a single line calling the generic `minConflictSearch` on model m program, i.e.,

```
1 minConflictSearch(m);
```

This leaves us with a 10 lines Constraint-Based Local Search implementation.

```
MinNeighborSelector N();
while(S.violations()!=0) {
   selectMax[2](i in Size)(S.violations(queen[i])) {
      forall(v in queen[i].getDomain() : queen[i] != v)
         neighbor(S.getAssignDelta(queen[i],v),N)
            queen[i] := v;
   }
   if (N.hasMove()) call(N.getMove());
}
```

```
import lssolver;
int n = 8;
range Size = 1..n;
model m {
   var{int} queen[Size](Size);
   soft: alldifferent(queen);
   soft: alldifferent(all(i in Size) (queen[i] + i));
   soft: alldifferent(all(i in Size) (queen[i] - i));
}
ConstraintSystem<LS> S = m.getSoftConstraintSystem();
while(S.violations() > 0)
  selectMin(i in Size,v in Size : queen[i]!=v)(S.getAssignDelta(queen[i],v))
    queen[i] := v;
```

Fig. 9 A revised Constraint-Based Local Search model for n-queens

```
void function minConflictSearch(Model<LS> m) {
   ConstraintSystem<LS> S = m.getSoftConstraintSystem();
   var{int}[] X = S.getIntVariables();
   while(S.violations() > 0)
     selectMin(i in X.getRange(),
               v in X[i].getDomain() : X[i]!=v)(S.getAssignDelta(X[i],v))
       X[i] := v;
}
```

Fig. 10 A reusable min-conflict procedure

Case Studies

To explore Constraint-Based Local Search, it is desirable to consider a few applications that are elegantly and effectively solved with COMET. The next three subsections consider the progressive party problem [16], car sequencing [3], and scene allocation [18] as each application illustrates a different aspect.

Progressive Party

The progressive party problem is a standard benchmark in combinatorial optimization and it illustrates two important features. It shows a rich model with many combinatorial constraints as well as constraints on the truth value of relations. It also shows how soft constraints may be instrumental in obtaining a neighborhood. The goal in this problem is to assign guest parties to boats (the hosts) over multiple time periods. Each guest can visit the same boat only once and can meet every other guest at most once over the course of the party. Moreover, for each time period, the guest assignment must satisfy the capacity constraints of the boats.

Figure 11 depicts the declarative part of the model. The decision variable `boat[g,p]` specifies the boat visited by guest `g` in period `p`. Lines 8–9 specify the alldifferent constraints for each guest, lines 10–11 specify the capacity constraints, and lines 12–13 state that two guests meet at most once during the evening. The `soft(2)` annotations added on lines 9 and 11 are not only specifying that the constraint must be softened, but they also associate a fixed static weight of 2 with each constraint. The weights can be easily incorporated in the inductive definition of $\mathbb{V}(c)$ shown in Fig. 4

$$\mathbb{V}(\texttt{soft}(w) : c) = w \cdot \mathbb{V}(c)$$

to handle *statically weighted* soft constraints.

The Neighborhood

The first sensible neighborhood to consider focuses on reassigning a single variable `boat[g,p]` to a new value. This can follow the same template used for the queens problem. These moves impact the violations of *all* the constraints. However, these moves may also prove too restrictive. When an instance is near satisfaction, one can expect most of the bins in any given knapsack to be near full. A single

```
1  range Hosts = 1..13;
2  range Guests = 1..29;
3  range Periods = 1..up;
4  set{int} config[1..6];
5
6  model m {
7     var{int} boat[Guests,Periods](Hosts);
8     forall(g in Guests)
9        soft(2): alldifferent(all(p in Periods) boat[g,p]);
10    forall(p in Periods)
11       soft(2): knapsack(all(g in Guests) boat[g,p],crew,cap);
12    forall(i in Guests, j in Guests : j > i)
13       soft: atmost(1,all(p in Periods) boat[i,p] == boat[j,p]);
14 }
```

Fig. 11 The progressive party problem

variable reassignment amounts to moving an item from one bin into another and that may prove fruitless from a violation standpoint. A natural idea is to include a *second neighborhood* that considers the exchange of values between two variables appearing in the same constraint. While such swaps have no effects on the `alldifferent` constraints, they can more easily lead to violation improvements for the `knapsack` constraints. Formally, the neighborhood is therefore

$$\begin{aligned} N = \{\sigma[x/v] \quad & | \; x = \text{arg-max}_{z \in X} \downarrow_z (\sigma, \mathbb{V}(S)) \wedge v \in D(x) \setminus \{\sigma(x)\}\} \bigcup \\ \{\sigma[x/c, y/d] \; & | \; x = \text{arg-max}_{z \in X} \downarrow_z (\sigma, \mathbb{V}(S)) \wedge \\ & \quad y \in \bigcup_{c \in S \wedge x \in \text{vars}(c)} \text{vars}(c) \wedge c = \sigma(y) \wedge d = \sigma(x)\} \end{aligned}$$

where S is the set of soft constraints.

The code in Fig. 12 implements that idea. The main loop (lines 7–30) seeks to improve the overall violations of the soft constraints. The selector on line 9 picks

```
1  int tenure = 2;
2  int it = 0;
3  int tabu[Guests,Periods,Hosts] = 0;
4
5  ConstraintSystem<LS> S = m.getSoftConstraintSystem();
6  MinNeighborSelector N();
7  while (S.violations() > 0) {
8     int old = S.violations();
9     selectMax(g in Guests, p in Periods)(S.violations(boat[g,p])) {
10       forall(h in Hosts : tabu[g,p,h] <= it) {
11          neighbor(S.getAssignDelta(boat[g,p],h),N) {
12             tabu[g,p,boat[g,p]] = it + tenure;
13             boat[g,p] := h;
14          }
15       }
16       selectMin(g1 in Guests,d=S.getSwapDelta(boat[g,p],boat[g1,p]))(d) {
17          neighbor(d,N) {
18             tabu[g,p,boat[g1,p]] = it + tenure;
19             tabu[g1,p,boat[g,p]] = it + tenure;
20             boat[g,p] :=: boat[g1,p];
21          }
22       }
23    }
24    if (N.hasMove()) {
25       call(N.getMove());
26       if (violations < old && tenure > 2) tenure = tenure - 1;
27       if (violations >= old && tenure < 10) tenure = tenure + 1;
28    }
29    it = it + 1;
30 }
```

Fig. 12 The search procedure for the progressive party problem

a variable `boat[g,p]` that induces the most violations. Once the variable is selected, the two *nested* selectors (lines 10–15 and lines 16–22) implement the two parts of the neighborhood structure. Lines 10–15 select the variable with the most violations and choose a new value that decreases its violations the most. The second selector is more interesting. It picks a second variable `boat[g1,p]` in the same period `p` as the first variable `boat[g,p]` in such a way that the swap between the two variables has the largest impact on the overall violations of S. Note that both variables appear in the knapsack constraint stated over period p. The remainder of the code (lines 23–28) executes the best move in N (if one exist) and updates the variable tabu tenure.

The 30 lines of code are compact and elegant: They benefit from the automation provided by the neighborhood selector, the selectors, and the neighbor construction. Yet, they still require some effort to analyze the model and produce a code template that follows a standard recipe. In addition, this code skeleton must still be updated with classic ideas like search intensification around high-quality local minima and diversification to escape from basins of attraction. However, the steps applied in deriving this search procedure are rather systematic: They rely on an analysis of the model to recognize the presence of specific types of constraints that then suggest a particular neighborhood structure. The is the idea behind the synthesis of search procedures that is described next.

The Synthesized Search

Given that models are first-class objects, COMET can manipulate and analyze them. Here, COMET recognizes the presence of knapsack constraints and generates a *composite* neighborhood consisting of the union of variable assignments and of the variable swaps appearing in violated knapsack constraints, an idea first articulated by Van Hentenryck [19]. The synthesis process per se was described in details in [22].

The skeleton of a synthesized tabu search is depicted in Fig. 13. It uses the fact that all variables have the same domains. Lines 7–9 associate with each decision variable x_i the set of constraints mentioning x_i and susceptible to benefit from exchanging (swapping) the values of two of its variables. This initial step is a straightforward projection of the constraint array in S that only retains constraints that refer to x_i and *can use a swap*. Line 10 defines a neighborhood selector. Line 15 selects the variable x_i with the most violations among the softened constraints S. Lines 16–20 focus on the first part of the neighborhood and consider simple reassignments that lead to the largest violation decrease. All the potential neighbors are submitted to N. Lines 21–29 consider all the constraints referring to variable x_i and for which swaps can be of potential benefit. For each such constraint, the code retrieves the variables of the constraint and accumulates in the selector N all the closures modeling swaps (along with their impact on violations).

It is appealing to notice the similarities between the manual and synthetic implementation. In essence, they both capture the same idea. Yet, the generic code adapts to the model and consider exploiting all the constraints susceptible to profit

```
1  function bool refersTo(var{int}[] av,var{int} x) {
2     return or(i in av.rng()) (av[i].getId() == x.getId());
3  }
4  function void minConflictWithSwap(Model<LS> m) {
5     ConstraintSystem<LS> S = m.getSoftConstraintSystem();
6     var{int}[] X = S.getVariables();
7     set{Constraint<LS>} cx[i in X.rng()] =
8                         collect(j in S.rng() : S[j].canUseSwap() &&
9                                 refersTo(S[j].getVariables(),X[i])) S[j];
10    MinNeighborSelector N();
11    int k = 0,tenure = 20;
12    int at[X.rng()] = 0;
13    int mat[X.rng(),X.rng()] = 0;
14    forall(int k=0;S.violations() != 0;k++) {
15       selectMax(i in X.getRange())(S.violations(X[i])) {
16          forall(v in X[i].getDomain() : at[i] <= k)
17             neighbor(S.getAssignDelta(X[i],v),N) {
18                X[i] := v;
19                at[i] = k + tenure;
20             }
21          forall(c in cx[i]) {
22             var{int}[] Y = c.getVariables();
23             forall(j in Y.getRange() : mat[i,j] <= k)
24                neighbor(S.getSwapDelta(X[i],Y[j]),N) {
25                   X[i] :=: Y[j];
26                   mat[i,j] = mat[j,i] = k + tenure;
27                }
28          }
29       }
30       if (N.hasMove()) call(N.getMove());
31    }
32 }
```

Fig. 13 The synthesized search procedure for the progressive party problem

from swaps. The COMET extension required to do so is a minimal extension to query the capabilities of the constraint in the model.

As shown later, the synthesized search outperforms all published results on this problem. Note that, if the set of required constraints R was not empty (it is empty in this application), the two neighborhoods would have to discard assignments and swaps that yield nonzero values for the calls to `getAssignDelta` and `getSwapDelta` to implement the legality requirement correctly. Finally, observe that this skeleton implementation contains a very simply tabu condition to further restrict the legal moves to those that were not recently attempted; this is achieved with two simple data structures (one per neighborhood) that record the last iteration number when a move was performed. Supporting generic intensification and diversification is equally easy.

Car Sequencing

Figure 14 presents a model for car sequencing. In this application, n cars must be sequenced on an assembly line of length n. The customer demands for car configurations are specified in an array $demand$ and the total demand is, of course, n. Each car configuration may require a different sets of options, while capacity constraints on the production units restrict the possible car sequences. For a given option o, these constraints are of the form k out of m meaning that, out of m successive cars, at most k can require o. The model declares the decision variables specifying which type of car is assigned to each slot in the assembly line (line 12). It states a hard constraint specifying which cars must be produced (line 13) and then states the soft capacity constraints for each option (lines 14–15).

The handcrafted search procedure, illustrated in Fig. 15, is modeled after a conflict minimization structure. The initialization satisfies the cardinality constraint by using a random permutation of an array of values that already meet the cardinality requirement (lines 4–7). The main loop (lines 10–27) minimizes the number of violations by selecting the slot of the assembly line causing the most violations and swapping its content with another slot that delivers the most improvements. The move itself appears on line 14. The search features a *diversification* component (lines 19–25) that randomly swaps a subset of slots in the assembly line when no improvement took place for a number of iterations. Each time an improving move is found, the stability counter is reset and the best value for this stage is recorded (line 18).

It is possible to recognize the combinatorial structure present in the model thanks to the presence of global `sequence` constraints and to automatically synthesize a search procedure that matches the ideas present in the handcrafted search procedure.

```
1  // ... read parameters nbCars, nbConfigs, nbOptions
2  range Cars = 1..nbCars;
3  range Configs = 0..nbConfigs-1;
4  range Options = 1..nbOptions;
5  boolean requires[Configs,Options];
6  int demand[Configs];
7  int lb[Options],ub[Options];
8  // ... read the data ...
9
10 set{int} options[o in Options] = setof(c in Configs) requires[c,o];
11 model m {
12    var{int} line[Cars](Configs);
13    hard: atmost(demand,line);
14    forall(o in Options)
15       soft: sequence(line,options[o],lb[o],ub[o]);
16 }
```

Fig. 14 The car sequencing model

```
1  ConstraintSystem<LS> S = m.getSoftConstraintSystem();
2  Solver<LS> ls = m.getLocalSolver();
3  int best = System.getMAXINT();
4  int cars[Cars];
5  int nb = 0;
6  forall(c in Configs, n in 1..demand[c])
7      cars[++nb] = c;
8  RandomPermutation perm(Cars);
9  forall(c in Cars) lines[c] := cars[perm.get()]; // Satisfy required constraint
10 while (S.violations() != 0) {
11     selectMax(i in Cars)(S.violations(X[i])) {
12         selectMin(j in Cars : line[i] != line[j] && t[i,j] <= it)
13             (S.getSwapDelta(line[i],line[j])) {
14             X[i] :=: X[j];
15             t[i,j] = t[j,i] = it + tenure;
16         }
17     }
18     if (S.violations() < best) { best = S.violations();stable = 0;}
19     if (stable == 500) {
20         with atomic(ls)
21             forall(k in 1..5)
22                 select(a in Cars,b in Cars : line[a] != line[b]) line[a] :=: line[b];
23         stable := 0;
24         best = S.violations();
25     } else stable++;
26     it++;
27 }
```

Fig. 15 The handcrafted search procedure for car sequencing

The result is shown in Fig. 16. It depicts a generic tabu search procedure featuring several key components. Note how line 6 of the model retrieves the required constraints (line 3) and initializes the start assignment by delegating to each required constraint the task of picking a suitable assignment. A basic requirement for achieving this is the following independence property of the required constraints:

$$\forall c_i, c_j \in R \text{ s.t. } i \neq j : \text{vars}(c_i) \cap \text{vars}(c_j) = \emptyset.$$

In this case, $R = \{\texttt{atmost(demand,line)}\}$ and the sole cardinality constraint can create, in polynomial time, an array of values that meet the cardinality requirement and randomly permute it to produce the initial assignment to the variables in vars(c). The algorithm for the cardinality constraint uses a (randomized) feasible flow algorithm to match values to variables. As all variables appear in the cardinality constraint, swapping two variables is feasibility-preserving (the number of cars of each type is unchanged). Moreover, the cardinality constraint is tight, meaning that there is a bijection from variable to value occurrences. The core of the

```
1  function void tabuSearch(Model<LS> m) {
2     ConstraintSystem<LS> S = m.getSoftConstraintSystem();
3     ConstraintSystem<LS> R = m.getHardConstraintSystem();
4     Solver<LS> ls = m.getLocalSolver();
5     var{int}[] X = R.getVariables();
6     forall(i in R.getRange()) with atomic(ls) R.getConstraint(i).initialize();
7     Counter it(ls) := 0;
8     Counter stable(ls) := 0;
9     int tenure = 20;
10    int t[X.rng(),X.rng()] = 0;
11    Integer best(System.getMAXINT());
12    whenever S.violations()@changes(int o,int n)
13       if (n < best) { best := n;stable := 0;}
14    whenever it@changes() stable++;
15    whenever stable@changes()
16       if (stable == 500) {
17          with atomic(ls)
18             forall(k in 1..5)
19                select(a in X.rng(),b in X.rng() : X[a] != X[b]) X[a] :=: X[b];
20          stable := 0;
21       }
22    while (S.violations() != 0) {
23       selectMax(i in X.rng())(S.violations(X[i])) {
24          selectMin(j in X.rng() : X[i] != X[j] && t[i,j] <= it)
25             (S.getSwapDelta(X[i],X[j])) {
26             X[i] :=: X[j];
27             t[i,j] = t[j,i] = it + tenure;
28          }
29       }
30       it++;
31    }
32 }
```

Fig. 16 The synthesized Tabu-search for car sequencing

search spans lines 22–31 and features the selection of the most conflicting variable (line 23) together with the variable that yield the largest decrease in violations through a swap (lines 24–25). The actual move is performed on line 26 and the move is marked tabu in line 27.

The implementation of the diversification is more interesting. To modularize the capability, it relies on *events*. In COMET, objects like `Counter`, `Integer`, or `var{int}` are capable of dispatching *notifications* when specific events occur. For instance, a counter issues a *change* event when its value is modified. COMET provides the ability to associate a code fragment with events: The code is then executed in response to these events. This is illustrated on line 14 that states that, each time the iteration counter `it` changes, the stability counter `stable` must increase. Similarly, lines 15–21 specify that, when the stability counter changes, one should check whether it has reached a critical value, 500 in this example,

in which case a diversification step is undertaken. This architecture promotes the separation of the diversification logic from the main heuristic. Indeed, the code for the diversification simply dictates how to react to changes to the stability counter and the COMET platform automatically *weaves* that code in the proper place. Finally, note how recording improvements in the violations are also done through an event hooked on the violations of the entire soft system S.

Scene Allocation

Figure 17 features a model for the scene allocation problem that is sometimes used to compare CP and MIP solvers since it is highly symmetric. The problem consists of assigning specific days for shooting scenes in a movie. There can be at most five scenes shot per day and all actors of a scene must be present. Each actor has a fee and is paid for each day she/he plays in a scene. The goal is to minimize the production cost. The decision variable $scene$ (line 12) represents the day a scene is shot. The hard cardinality constraint (line 13) specifies the maximum number of scenes shot on any one day. The objective function minimizes the sum of actor compensations, which is the actor fee times the number of days he/she appears in a scene shot on that day.

The model in Fig. 17 is now a constraint optimization model featuring an objective function that demands a different strategy to obtain a suitable search procedure. In general, it is necessary to juggle three considerations. First, one should maintain the feasibility of the required constraints through a suitable initialization and the selection of moves that never violate the constraints in R. Second, one

```
1  int maxScene = ...; // read the data
2  int maxDay = ...; // read the data
3  range Scenes = 1..maxScene;
4  range Days = 1..maxDay;
5  enum Actor = ... ; // read the data
6  int pay[Actor] = ...; // read the data
7  set{Actor} appears[Scenes] = ...; // read the data
8  set{int} which[a in Actor] = setof(s in Scenes) member(a,appears[s]);
9  int occur[Days] = ...; // read the data
10
11 model m {
12    var{int} scene[Scenes](Days);
13    hard: atmost(occur,scene);
14    minimize: sum(a in Actor) pay[a]*
15                    (sum(d in Days) (or(s in which[a]) (scene[s] == d)));
16 }
17 generatedTabuSearch(m);
```

Fig. 17 Scene allocation model

may have to deal with softened difficult constraints (S) for whom one searches for a feasible solution by driving down the violations. Third, it is essential to drive down the value of the objective function. The latter two considerations (true objective and softened constraint) are naturally conflicting as it is easier to drive the objective function down if one violates difficult constraints and vice versa. The practical response is to rely on a statically or a dynamically weighted sum of the two objectives where the search shifts the emphasis on either considerations by altering the weights.

In this application, $S = \emptyset$ and $R = \{\texttt{atmost}(occur, scene)\}$; hence, the task is somewhat simpler as there is only one component to the objective function. In this case, the generated tabu search is driven by the sole required constraint but with a significant difference. In the previous example, the soft constraint was tight, a fact detected by the model analysis. Indeed, the cardinality constraint in car sequencing is tight because the assembly line has as many slots as the number of cars to produce. This is not the case here: there are typically fewer scenes than the number of slots in which they can be scheduled and thus the required constraint is not a bijection between variables and value occurrences. As a result, limiting the neighborhood only to swaps would preserve the feasibility of the cardinality constraint at the expense of a significant decrease in solution quality. This is not surprising since the neighborhood would no longer be connected. The model analysis however recognizes that the `atmost` constraint is not tight and also considers feasibility-preserving assignments.

The generated skeleton is depicted in Fig. 18. As before, line 5 uses the required constraints to initialize the search to a feasible assignment (once again, the task is delegated to the required constraints and the model analysis ensures that $\text{vars}(c_i) \cap \text{vars}(c_j) = \emptyset \, \forall c_i, c_j \in R$ before generating code. The core of the search spans lines 11–24 and relies on the union of two neighborhoods. Line 12 starts by selecting a variable x_i that can lead to the largest decrease in the objective function. Lines 13–15 consider all the swaps that include x_i and lead to the largest decrease in the objective function (i.e., the most negative delta). The selector on line 13 is semi-greedy and will select, uniformly at random, one of the top-3 such moves. Lines 16–19 are devoted to the second neighborhood and collect the best value to reassign x_i. Line 17 shows the conjunct that eliminates assignments that are not feasible with respect to R. The skeleton search uses a vanilla tabu data structure and omits the diversification component for simplicity.

Implementation

The implementation of a Constraint-Based Local Search system critically depends on incremental computation. Constraints and objective functions must respond to their APIs like `violations`, `increase`, `decrease`, or `getAssignDelta` extremely fast in order to consider large neighborhoods and long traces of assignment within an allotted time.

```
function void generatedTabuSearch(Model<LS> m) {
   Function<LS> obj = m.getObjective();
   ConstraintSystem<LS> R = m.getHardConstraintSystem();
   Solver<LS> ls = m.getLocalSolver();
   forall(i in R.getRange()) with atomic(ls) R.getConstraint(i).initialize();
   int it = 0,tenure = 20,best = System.getMAXINT();
   var{int}[] X = obj.getVariables();
   int tm[X.rng(),X.rng()] = 0;
   int t[X.rng()] = 0;
   MinNeighborSelector N();
   while (it < 10000) {
      selectMax(i in X.rng())(obj.decrease(X[i])) {
         selectMin[3](j in X.rng() : i != j && tm[i,j] <= it,
                      d = obj.getSwapDelta(X[i],X[j]))(d)
            neighbor(d,N) { X[i] :=: X[j];tm[i,j] = tm[j,i] = it + tenure;}
         selectMin(v in X[i].getDomain() : t[i] <= it && v != X[i] &&
                                           R.getAssignDelta(X[i],v)==0,
                   d = obj.getAssignDelta(X[i],v))(d)
            neighbor(d,N) { X[i] := v;t[i] = it + tenure;}
      }
      if (N.hasMove()) call(N.getMove());
      if (obj.evaluation() < best) best = obj.evaluation();
      it++;
   }
}
```

Fig. 18 Generated search for scene allocation

To deliver this performance, the implementation is presented in two distinct layers. The `invariant` layer is responsible for the basic incremental computation that occurs when an assignment is changed through a local move operator. The `differentiability` layer is responsible for implementing the response mechanism behind the constraints and objective function. Their implementation is primarily framed in terms of invariants. Both are highlighted in this section, starting with invariants (section "Invariants") and finishing with differentiation (section "Differentiation").

Invariants

Invariants provide a declarative concept that relieves programmers from the tedious task of maintaining complex data structures incrementally. By focusing on what to maintain, rather than how to maintain assignments under changes, programmers are relieved of an error-prone, yet critical, aspect of implementing constraints and objective functions. In essence, invariants capture so-called *one-way constraints* [1, 2, 11, 17], namely, they capture the value of an expression that must be

maintained over time under changes to the value of its variables. An acyclic network of dependencies connects all the variables of the problems and is responsible for scheduling the evaluations. This subsection reviews examples and outlines the underlying implementation.

Definition 15 (Invariant). An invariant $\mathcal{I}$ is a one-way constraint

$$\langle x_1, \ldots, x_n \rangle \leftarrow f(y_1, \ldots, y_m)$$

where $x_1, \ldots, x_n$ are called invariant variables and $y_1, \ldots, y_m$ are either decision or invariant variables. The set $O = \{x_1, \ldots, x_n\}$ is the output variables of $\mathcal{I}$ The set $I = \{y_1, \ldots, y_m\}$ are the input variables of $\mathcal{I}$. We often abuse notation and rewrite the one-way constraints as

$$O \leftarrow f(I).$$

$\mathcal{I}_O$, $\mathcal{I}_I$, and $\mathcal{I}_f$ denote the output variables, the input variables, and the function of invariant $\mathcal{I}$.

Since invariants are one-way constraints, there are some necessary syntactic restrictions. One such restriction is that no two invariants have a common output variable. This ensures that every invariant variable is defined at most once.

The declarative semantics of an invariant specifies that the one-way constraints always holds.

Definition 16 (Declarative Semantics of an Invariant). Let σ be an assignment before or after any atomic instruction of a program for which the invariant $\mathcal{I}$ has been posted. Then, it follows that

$$\langle \sigma(x_1), \ldots, \sigma(x_n) \rangle \leftarrow f(\sigma(y_1), \ldots, \sigma(y_m))$$

where $\mathcal{I}_O = \{x_1, \ldots, x_n\}$ and $\mathcal{I}_I = \{y_1, \ldots, y_m\}$. The narrative abuses notation and sometimes writes

$$\sigma(\mathcal{I}_o) \leftarrow \mathcal{I}_f(\sigma(\mathcal{I}_I)).$$

Example 8 (Expression Invariant). The numerical invariant

$$x \leftarrow y + 3 * z$$

stating that, at any point in time, the value of x in an assignment σ, i.e., $\sigma(x)$, should be the value $\sigma(y)$ plus three times the value of $\sigma(z)$.

Operationally, an invariant must maintain the link between its output and input variables under assignments to its input variables. The invariant maintains this link

by keeping a local assignment of its input variables and by implementing an update function which updates the global and local assignments to reflect the change in one of its input variables.

Definition 17 (Operational Semantics of an Invariant). Let $\mathcal{I}$ be an invariant. Operationally, $\mathcal{I}$ maintains a local store $\mathcal{I}_\sigma$ over variables $\mathcal{I}_I$ and implements an update procedure $\mathcal{I}_u : \Sigma \times X$. Let $\sigma_l = \mathcal{I}_\sigma$ be the local store of $\mathcal{I}$, σ be an assignment, and $y \in \mathcal{I}_I$. The update procedure $\mathcal{I}_u(\sigma, y)$ performs the following assignments:

$$\sigma(\mathcal{I}_O) := \mathcal{I}_f(\sigma_l[\sigma(y)/y](\mathcal{I}_I));$$
$$\sigma_l(y) := \sigma(y);$$

To propagate a collection of invariants effectively, the implementation constructs an incremental graph.

Definition 18 (Incremental Graph). An incremental graph $G(X \cup I, A)$ is a directed acyclic graph whose vertices coincides with decision variables and invariants and whose arc set correspond to the dependencies induced by the invariant. Each invariant $\mathcal{I}$ introduces a dependency $\mathcal{I} \leftarrow y$ for each $y \in \mathcal{I}_I$ and a dependency $x \leftarrow \mathcal{I}$ for each $x \in \mathcal{I}_O$.

Figure 19 depicts the dependencies of the arithmetic invariant presented earlier.

Example 9 (Summation Aggregate). A summation invariant captures the relation

$$x \leftarrow \sum_{i=0}^{n-1} y[i]$$

where n is a constant and y denotes an array of n variables. The dependencies are shown below:

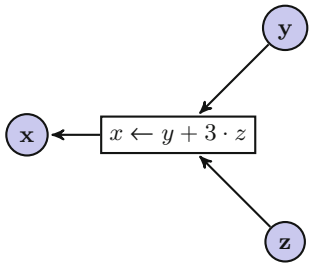


Fig. 19 The dependencies of an arithmetic invariant

The update function $\mathcal{I}_u(\sigma, y)$ implements the following code:

```
1 σ(x) := σ(x) + σ(y) − σ_I(y);
2 σ_I(y) := σ(y);
```

where $\sigma_I = \mathcal{I}_\sigma$.

Example 10 (Counting). A counting invariant $x \leftarrow \text{count}(y)$ defined over an array of variables y yields an array of variables x indexed by $R = \bigcup_{i=0}^{n-1} D(y_i)$ that maintains the relations

$$\forall\, v \in R \,:\, x_v = \sum_{i \,\in\, \text{range}(y)} (y_i = v)$$

Namely, x_v counts the number of variables in y currently assigned to v. The dependencies are as follows:

and there are $|R| - 1 + n$ of them. The update function $\mathcal{I}_u(\sigma, y)$ implements the following code:

```
1  σ(x_{σ_I(y)}) := σ(x_{σ_I(y)}) − 1;
2  σ(x_{σ(y)}) := σ(x_{σ(y)}) + 1;
3  σ_I(y) := σ(y);
```

where $\sigma_I = \mathcal{I}_\sigma$.

Incremental Computation

Given $G(X \cup I, A)$, one can obtain a topological sort r of its vertices. Indeed, each dependency $y \leftarrow x$ imposes the constraint

$$1 + r(x) \leq r(y).$$

The partial ordering expressed in r drives the propagation algorithm that updates all the variables following an assignment of new values to decision variables.

Figure 20 shows the pseudo-code for the invariant propagation algorithm. The `propagate` algorithm is invoked with the incremental graph $G(X, A)$, an assignment σ and the set of decision variables Y that have been updated. Line 3 initializes a priority queue PQ with all the invariants mentioning any member of Y as one of its sources. The priority associated with invariant $\mathcal{I}$ is its topological number $r(\mathcal{I})$. The main loop spanning lines 6–11 considers the invariants in priority order (see Line 7). The update function of the selected invariant is executed in Line 8. Line 9 collects in C the modified output variables and Line 10 enqueues the new invariant to reconsider.

The correctness of the `propagate` algorithm hinges on the facts that $G(X, A)$ is acyclic. The use of topological numbers guarantee that the invariant considered in iteration i is handled only after its sources have reached final values in σ. As long as u meets its specification, the assignment σ is guaranteed to satisfy all the

```
1 propagate(G(X, A),σ,Y)
2 {
3     PQ = ⋃_{y∈Y} {⟨I ← y, r(I)⟩ ∈ A};
4     while (PQ ≠ ∅) {
5         σ_o := σ;
6         ⟨I ← y, r(I)⟩ := extractMin(PQ);
7         I_u(σ, y);
8         C = {x ∈ I_O : σ_o(x) ≠ σ(x)};
9         forall(x ∈ C)
10            PQ = PQ ∪ {⟨I ← x, r(I)⟩ ∈ A};
11    }
12 }
```

Fig. 20 The invariant propagation

invariants considered in iterations $1 \dots i$. Computing the affected variables in C and scheduling any invariant depending on them cannot possibly schedule an earlier invariant since the graph is acyclic.

Differentiation

The implementation of constraints and objective functions relies on the foundation provided by invariants as first described in [21]. As indicated earlier, the implementation of the constraint API

```
1 interface Constraint<LS> {
2   bool holds();
3   var{int} violations();
4   var{int} violations(var{int} x);
5   int getAssignDelta(var{int} x,int v);
6 }
```

depends on an efficient, incremental evaluation of violations, variable violations, and gradients. The functions $\mathbb{E}(\sigma, e)$, $\mathbb{V}(e)$, $\uparrow_x (\sigma, e)$, and $\downarrow_x (\sigma, e)$ are essential to the evaluation of expressions and the definition of violation and gradient expressions from algebraic definitions. Yet, none of them are incremental and therefore unsuitable for *direct* use in, for instance, Example 5. Likewise, this is true for combinatorial constraints such as `alldifferent`. Invariants do provide the solution, and the subsection focuses on the implementation when constraints are expressed algebraically or combinatorially.

Algebraic and Logical Constraints

The key insight to an incremental implementation is to forsake the evaluation function $\mathbb{E}$ and adopt instead a *compilation* approach, generating invariants that evaluate an expression incrementally. This is achieved by Function $\mathbb{I} : \mathcal{E} \rightarrow X \times 2^I$

Fig. 21 Compiling expression evaluations to invariants

$$
\begin{array}{ll}
\mathbb{I}(v) & = \langle i_v, \{i_v \leftarrow v \Leftrightarrow v \in \mathbb{R}\}\rangle \\
\mathbb{I}(x) & = \langle i_x, \{i_x \leftarrow x\}\rangle \\
\mathbb{I}(e_1 + e_2) & = \text{let } \langle i_1, S_1\rangle = \mathbb{I}(e_1), \langle i_2, S_2\rangle = \mathbb{I}(e_2) \\
 & \quad \text{in } \langle i_+, \{i_+ \leftarrow i_1 + i_2\} \cup S_1 \cup S_2\rangle \\
\mathbb{I}(e_1 - e_2) & = \text{let } \langle i_1, S_1\rangle = \mathbb{I}(e_1), \langle i_2, S_2\rangle = \mathbb{I}(e_2) \\
 & \quad \text{in } \langle i_-, \{i_- \leftarrow i_1 - i_2\} \cup S_1 \cup S_2\rangle \\
\mathbb{I}(e_1 \times e_2) & = \text{let } \langle i_1, S_1\rangle = \mathbb{I}(e_1), \langle i_2, S_2\rangle = \mathbb{I}(e_2) \\
 & \quad \text{in } \langle i_\times, \{i_\times \leftarrow i_1 \times i_2\} \cup S_1 \cup S_2\rangle \\
\mathbb{I}(\textsc{Abs}(e)) & = \text{let } \langle i_e, S_e\rangle = \mathbb{I}(e) \\
 & \quad \text{in } \langle i_{abs}, \{i_{abs} \leftarrow \textsc{Abs}(i_e)\} \cup S_e\rangle \\
\mathbb{I}(\min(e_1, e_2)) & = \text{let } \langle i_1, S_1\rangle = \mathbb{I}(e_1), \langle i_2, S_2\rangle = \mathbb{I}(e_2) \\
 & \quad \text{in } \langle i_{min}, \{i_{min} \leftarrow min(i_1, i_2)\} \cup S_1 \cup S_2\rangle \\
\mathbb{I}(\sum_{k=0}^{n} e_k) & = \text{let } \langle i_k, S_k\rangle = \mathbb{I}(e_k) \ \forall\ k \ \in 0..n \\
 & \quad \text{in } \langle i_\Sigma, \{i_\Sigma \leftarrow \sum_{k=0}^{n} i_k\} \cup \bigcup_{k=0}^{n} S_k\rangle
\end{array}
$$

(which is partly shown in Fig. 21) and defined inductively on the structure of expressions. Specifically, a call $\mathbb{I}(e)$ on an expression e produces a decision variable holding the value of the expression and a set of invariants that define this variable.

Note how each line of the inductive definition obtains the variable and invariants supporting the operand and produces a fresh variable alongside an additional invariant based on the variables obtained from the inductive calls on the operands. For instance, the last line of Fig. 21 shows that the compilation of a summation expression inductively obtains a fresh decision variable i_k for each term e_k, as well as the invariants supporting i_k's definition in S_k. It then creates a new fresh variable i_Σ and the summation aggregate invariant that defines it. It finally adds all the invariants in S_k.

Relations simply give rise to arithmetic expressions through the function $\mathbb{V}$ whose definition is in Fig. 3. To obtain an incremental evaluation of the violations of an arbitrary relation $e_1 \diamond e_2$, one can simply obtain the violation expression and compile it with:

$$\langle i_{e_1 \diamond e_2}, S_{e_1 \diamond e_2}\rangle = \mathbb{I}(\mathbb{V}(e_1 \diamond e_2))$$

to retrieve a set of invariants (which it states) and an output variable $i_{e_1 \diamond e_2}$ whose value $\sigma(i_{e_1 \diamond e_2})$ denotes the violations of $e_1 \diamond e_2$ with respect to σ. At this point, the implementation of method `violation()` is straightforward and reduces to returning $\sigma(i_{e_1 \diamond e_2})$. The incremental evaluation of gradients proceeds similarly with the generation of an expression modeling the gradient of e and its compilation with $\mathbb{I}$, i.e.,

$$\langle i_{\downarrow_x(e)}, S_{\downarrow_x(e)}\rangle = \mathbb{I}(\downarrow_x (e))$$

and $\downarrow_x (e)$ is an expression (independent of σ) whose evaluation w.r.t. σ would yield $\downarrow_x (\sigma, e)$. Similarly, a method call `violations(x)` must simply return $\sigma(i_{\downarrow_x(e)})$.

Finally, objective functions make a direct use of expressions as well as $\uparrow_x$ and $\downarrow_x$ and are therefore handled exactly like constraints.

Combinatorial Constraints

While combinatorial constraints could be implemented in terms of expressions, it is often preferable to exploit the semantics of the constraints to directly produce an incremental implementation. To illustrate the idea, consider the `alldifferent(x)` constraint used in the introductory example and responsible for ensuring that no two entries in x have the same value.

Fundamentally, the constraint should maintain the cardinality of each value used in x and require that no two values have a cardinality larger than 1 to satisfy the constraint. The variable violation for x_i would, in this case, simply be the excess in the number of variables assigned to $\sigma(x_i)$. In essence, when the constraint `alldifferent(x)` is added on an array x with n variables where $\bigcup_{j=1}^{n} D(x_j) = V$, it is sufficient to state the following invariants:

$$\begin{aligned} c &\leftarrow \text{count}(x) \\ v_i &\leftarrow \max(0, c_i - 1) \ \forall i \in V \\ cv &\leftarrow \textstyle\sum_{k \in V} v_i \\ vv_j &\leftarrow v_{x_j} \qquad\qquad \forall j \in 1 \ldots n. \end{aligned}$$

The implementation of the constraint then reduces to

```
1  class alldifferent implements Constraint<LS> {
2    bool holds() { return σ(cv) == 0;}
3    var{int} violations() { return cv;}
4    var{int} violations(var{int} x) { return vv_id(x);}
5    int getAssignDelta(var{int} x,int v) {
6      if σ(x) == v
7        return 0;
8      else
9        return (σ(c_v) ≥ 1) − (σ(x) ≥ 2);
10   }
11 }
```

where the function `id` is used to identify the variable x by an integer. Note the simplicity of the method implementations that simply leverage the work done by invariants. Additionally, the implementation does not have to provide any imperative code to handle the changes of decision variables as all of this logic is handled through the invariants. Finally, even the `getAssignDelta(x,v)` implementation remains straightforward. When the current value assignment to variable x is identical to the tentative assignment (v), the function returns 0. Otherwise, it returns the amount of change. Namely, if value v is already used once or more, the number of violations will increase by 1. Similarly, if $\sigma(x)$ is used twice or more, a violation is necessarily lost.

Empirical Results

Offering a comprehensive empirical evaluation of COMET is beyond the scope of this chapter. Yet, the monograph [23] contains an extensive empirical evaluation on many problems and discusses the impact of modeling techniques and search.

Instead, this section focuses on demonstrating the potential behind the synthesis of search procedures. In particular, it explores the performance of the search procedures shown in Figs. 13, 16, and 18 and contrasts them with the results obtained from purely synthesized search procedures in the spirit of [22]. In all cases, the results were obtained on a Core i7 machine clocked at 1.8 Ghz and running OSX 10.10 and the reported results are based on averages collected from 100 runs of each algorithms.

Progressive Party

Two instances of the problem (5–7 and 6–7) were used in the evaluation in the following table.

Type	Choice	$\|P\|$	μ_{iter}	σ_{iter}	μ_T (msec.)	σ_T (msec.)
Manual	5	7	2,360.9	1,386.1	723.9	377.3
Synthetic	5	7	2,282.2	1,495.9	842.5	476.1
Manual	6	7	6,847.8	5,714.7	1,682.0	1,324.4
Synthetic	6	7	4,532.2	4,724.5	1,338.1	1,281.6

The search procedures are extremely similar and only differ in the presence of an adaptive tabu list within the synthetic implementation. It is not surprising to note that both implementation are very close with a slight win for the synthetic search without having to invest any effort in parameter tuning.

Car Sequencing

One instance (4–72) was used for car sequencing. In this case, the two implementations seem to exhibit significantly different behaviors as the number of iterations is almost 4 times as high (on average) for the manual implementation. This is most likely due to the difference in parameters and in the components of the meta-heuristics within the synthetic search. It also shows that, without exploring variants of the search procedure, it is not obvious to produce high-quality results. Yet, the ability to easily exploit, without further ado, restarts, diversification and intensification components is quite valuable. Lastly, note that the time per iteration of the two searches is quite close showing that the differences only come from the search heuristic and meta-heuristic, not the incremental computation.

Type	Instance	μ_{iter}	σ_{iter}	$\mu_T(s.)$	$\sigma_T(s.)$
Manual	4–72	162,944.3	164,853.9	19.9	20.5
Synthetic	4–72	49,598.8	57,815.4	5.4	6.3

Scene Allocation

The scene allocation benchmark was used to compare three variants: the manual implementation in Fig. 18 and two synthetic search procedures with different parameters, namely, one of them uses 10,000 iterations and no restarts, while the other uses 10 restarts and 1000 iterations per restart. Here, the synthetic search relying on restarting is very close to the custom implementation. Its success rate is 99/100, just shy of a perfect 100 like the custom search. While the synthetic search uses more iterations (on average) to get to the optimum, the runtimes are very close. An examination of the synthesized search reveals that the root cause of the difference is the search heuristic. The equivalent of lines 13 and 17 in Fig. 18 in the synthesized search rely on a purely random selector rather than the more aggressive semi-greedy and greedy selectors used in the custom implementation. This difference explains the loss in greediness and explains the positive impact of restarts.

Type	$\mu_{\text{iter}}*$	$\sigma_{\text{iter}}*$	μ_{f*}	σ_{f*}	#Opt	μ_T(msec.)	σ_T(msec.)
Manual	860	982	334,144	0	100	567.4	46.9
Synthetic(10,000 iters, 1 restart)	2,254	3,104	334,960	1,093	64	731.8	175.6
Synthetic(1000 iters, 10 restarts)	1,954	2,212	334,167	227	99	730.1	71.7

Conclusion

Constraint-Based Local Search is an appealing framework for the design and implementation of local search models for any number of applications. It adopts the core practices of constraint programming through the support of separated components for expressing a declarative model and for programming the search. The declarative models rely on a rich language seamlessly blending algebraic, logical, and combinatorial constraints. The search component itself is exclusively devoted to the automation of the most tedious and error-prone activities that arise when implementing a variety of heuristics and meta-heuristics. Perhaps even more crucially, search procedures can be written completely independently of the model, making them highly reusable and generic. The culmination of this effort is the availability of synthetic search procedures that take advantage of an analysis of the declarative model to produce sensible search procedures that often compete with tailored, hand-written implementations.

The entire framework competitiveness relies on incremental implementations that are constructed on top of invariants and differentiable abstractions such as constraints and objectives. The net result is a platform for practitioners who would take advantage of the capabilities of local search techniques without the significant investment necessary to produce an efficient implementation.

Cross-References

- Ant Colony Optimization: A Component-Wise Overview
- Guided Local Search
- Iterated Greedy
- Restart Strategies
- Scatter Search
- Tabu Search
- Theory of Local Search
- Variable Neighborhood Descent
- Variable Neighborhood Search

References

1. Borning A (1981) The programming language aspects of thinglab, a constraint-oriented simulation laboratory. ACM Trans Program Lang Syst 3(4):353–387
2. Borning A, Duisberg R (1986) Constraint-based tools for building user interfaces. ACM Trans Comput Graph 5(4):345–374
3. Dincbas M, Simonis H, Van Hentenryck P (1988) Solving the car sequencing problem in constraint logic programming. In: ECAI-88, Aug 1988
4. Feo T, Resende M (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
5. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Boston/Dordrecht/London
6. Kirkpatrick S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680
7. Laguna M (2002) Scatter search. In: Pardalos PM, Resende MGC (eds) Handbook of applied optimization. Oxford University Press, New York, pp 183–193
8. Michel L (1998) Localizer: a modeling language for local search. PhD thesis, Brown University
9. Michel L, Van Hentenryck P (1997) Localizer: a modeling language for local search. In: Third international conference on the principles and practice of constraint programming (CP'97), Lintz, Oct 1997
10. Minton S, Johnston MD, Philips AB (1990) Solving large-scale constraint satisfaction and scheduling problems using a Heuristic repair method. In: AAAI-90, Aug 1990
11. Myers B, Guise D, Dannenberg R, Vander Zanden B, Kosbie D, Pervin E, Mickish A, Marchal P (1990) GARNET: comprehensive support for graphical, highly interactive user interfaces. IEEE Comput 23(11):71–85
12. Pham Q-D, Deville Y, Van Hentenryck P (2012) Ls(graph): a constraint-based local search for constraint optimization on trees and paths. Constraints 17(4):357–408
13. Selman B, Kautz H (1993) An empirical study of greedy local search for satisfiability testing. In: AAAI-93, pp 46–51

14. Selman B, Levesque H, Mitchell D (1992) A new method for solving hard satisfiability problems. In: AAAI-92, pp 440–446
15. Selman B, Kautz H, Cohen B (1996) Local search strategies for satisfiability testing. In: DIMACS series in discrete mathematics and theoretical computer science, vol 26. American Mathematical Society Publications. DIMACS
16. Smith BM, Brailsford SC, Hubbard PM, Williams HP (1996) The progressive party problem: integer linear programming and constraint programming compared. Constraints 1:119–138
17. Sutherland IE (1963) SKETCHPAD: a man-machine graphical communication system. MIT Lincoln Labs, Cambridge
18. Van Hentenryck P (2002) Constraint and integer programming in OPL. Inform J Comput 14(4):345–372
19. Van Hentenryck P (2006) Constraint programming as declarative algorithmics. ACP award for research excellence in constraint programming. Available at http://www.cs.brown.edu/people/pvh/acp.pdf
20. Van Hentenryck P, Michel L (2005) Control abstractions for local search. Constraints 10(2):137–157
21. Van Hentenryck P, Michel L (2006) Differentiable invariants. In: 12th international conference on principles and practice of constraint programming (CP'06), Nantes, Sept 2006. Lecture notes in computer science
22. Van Hentenryck P, Michel L (2007) Synthesis of constraint-based local search algorithms from high-level models. In: Proceedings of the 22nd national conference on artificial intelligence – volume 1, AAAI'07. AAAI Press, pp 273–278
23. Van Hentenryck P, Michel L (2009) Constraint-based local search. The MIT Press, Cambridge
24. Van Hentenryck P, Michel L, Liu L (2005) Constraint-based combinators for local search. Constraints 10(3):363–384
25. Walser JP (1999) Integer optimization by local search: a domain-independent approach. Springer, Berlin/Heidelberg. ISBN:3-540-66367-3. http://www.springer.com/us/book/9783540663676

Guided Local Search

10

Abdullah Alsheddy, Christos Voudouris, Edward P. K. Tsang, and Ahmad Alhindi

Contents

Introduction . . . 262
Guided Local Search . . . 264
  Implementation Guideline . . . 265
  Possible Features for Common Applications . . . 268
Guided Fast Local Search . . . 270
  Implementation Guideline . . . 271
GLS Extensions, Hybrids, and Variations . . . 275
  Extensions to GLS . . . 275
  GLS Hybrids . . . 276
  Variations of GLS . . . 277
GLS for Multi-objective Optimization . . . 278
  Pareto Local Search . . . 278
  Guided Pareto Local Search . . . 279
  Other Attempts . . . 281
GLS Implementation on the Traveling Salesman Problem . . . 281
  Problem Description . . . 281
  Local Search . . . 282
  Guided Local Search . . . 283
  Guided Fast Local Search . . . 284

A. Alsheddy (✉)
College of Computer and Information Sciences (CCIS), Al-Imam Muhammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia
e-mail: asheddy@imamu.edu.sa

C. Voudouris · E. P. K. Tsang
Department of Computer Science, University of Essex, Colchester, UK
e-mail: voudcx@essex.ac.uk; christos.voudouris@gmail.com; edward@essex.ac.uk

A. Alhindi
Department of Computer Science, Umm Al-Qura University, Makkah, Saudi Arabia
e-mail: ahhindi@uqu.edu.sa

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_2

GLS/GPLS Implementation on a Workforce Scheduling Problem 284
Problem Description 285
Local Search 287
Feature Definition 288
Overview of GLS Applications 288
Routing/Scheduling Problems 288
Assignment Problems 289
Resource Allocation Problems 290
Constrained Optimization Problem 291
GLS in Commercial Packages 291
Conclusions 291
Cross-References 292
References 293

Abstract

Guided local search (GLS) is a meta-heuristic method proposed to solve combinatorial optimization problems. It is a high-level strategy that applies an efficient penalty-based approach to interact with the local improvement procedure. This interaction creates a process capable of escaping from local optima, which improves the efficiency and robustness of the underlying local search algorithms. Fast local search (FLS) is a way of reducing the size of the neighborhood to improve the efficiency of local search. GLS can be efficiently combined with FLS in the form of guided fast local search (GFLS). This chapter describes the principles of GLS and provides guidance for implementing and using GLS, FLS, and GFLS. It also surveys GLS extensions, hybrids, and applications to optimization, including multi-objective optimization.

Keywords

Heuristic search · Meta-heuristics · Penalty-based methods · Guided local search · Tabu search · Constraint satisfaction

Introduction

Many practical problems are NP-hard in nature, which means complete, constructive search is unlikely to satisfy our computational demand. Many real-life problems cannot be realistically and reliably solved by complete search. This motivates the development of local search or heuristic methods.

Local search (LS) is the basis of most heuristic search methods. It searches in the space of candidate solutions.The solution representation issue is significant, though it is not the subject of our discussion here. In the basic variant of LS, known as hill climbing, LS starts from a (possibly randomly generated) candidate solution and then moves to a “neighbor” that is “better” than the current candidate solution

according to the objective function. The search naturally stops when all neighbors are inferior to the current solution.

LS can find good solutions very quickly. However, it can be trapped in local optima – positions in the search space that are better than all their neighbors, but not necessarily representing the best possible solution (the global optimum). To improve the effectiveness of LS, various techniques have been introduced over the years. Simulated annealing (SA), tabu search (TS), and guided local search (GLS) all attempt to help LS escape local optimum. This chapter focuses on GLS [79], a general meta-heuristic algorithm, and its applications. GLS has been applied to a nontrivial number of problems and found to be efficient and effective. It is relatively simple to implement and apply, with only a few parameters to tune.

GLS can be seen as a generalization of its predecessors GENET [23, 77] which was developed for constraint satisfaction problems. GLS also relates to ideas from the area of search theory on how to distribute the search effort. In particular, incremental distribution of the search effort according to information in a probabilistic framework can be found in a class of methods deriving themselves from the optimal search theory of Koopman [42, 69].

The principles of GLS can be summarized as follows. As a meta-heuristic method, GLS sits on top of LS algorithms. To apply GLS, one defines a set of features for the candidate solutions. When LS is trapped in local optima, certain features are selected and penalized. LS searches the solution space using the objective function augmented by the accumulated penalties.

The novelty of GLS is in the way that it selects features to penalize. GLS effectively distributes the search effort in the search space, favoring promising areas. Penalty modifications *regularize* the solutions generated by local search to be in accordance to prior or gathered during search information. The approach taken by GLS is similar to that of regularization methods for "ill-posed" problems [75]. The idea behind regularization methods and GLS up to an extent is the use of prior information to help in solving an approximation problem. Prior information translates to constraints which further define our problem reducing so the number of candidate solutions to be considered.

The structure of the chapter is as follows. In section "Guided Local Search", we describe GLS with details about implementing its components. Similarly, the implementation of combining GLS with fast local search is discussed in section "Guided Fast Local Search". Next, in section "GLS Extensions, Hybrids, and Variations", we review other algorithms that extend GLS or hybridize it with other techniques. We then present an extension of GLS to handle multi-objective optimization problems in section "GLS for Multi-objective Optimization". The applications of GLS to the traveling salesman problem and a workforce scheduling problem are explained in sections "GLS Implementation on the Traveling Salesman Problem" and "GLS/GPLS Implementation on a Workforce Scheduling Problem", respectively. We give comprehensive references to the applications of GLS and its variants in section "Overview of GLS Applications", and we conclude in section "Conclusions".

Guided Local Search

As mentioned earlier, GLS augments the given objective function with penalties. To apply GLS, one needs to define features for the problem. For example, in the traveling salesman problem [28], a feature could be *whether the candidate tour travels immediately from city A to city B*. GLS associates a cost and a penalty with each feature. The costs can often be defined by taking the terms and their coefficients from the objective function. For example, in the traveling salesman problem, the cost of the above feature can simply be the distance between cities A and B. The penalties are initialized to 0 and will only be increased when the local search reaches a local optimum. Given an objective function g that maps every candidate solution s to a numerical value, GLS defines a function h that will be used by LS (replacing g):

$$h(s) = g(s) + \lambda \times \sum_{i \text{ is a feature}} (p_i \times I_i(s)) \tag{1}$$

where s is a candidate solution, λ is a parameter of the GLS algorithm, i ranges over the features, p_i is the penalty for feature i (all p_i's are initialized to 0), and I_i is an indication of whether s exhibits feature i:

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i; 0 \text{ otherwise.} \tag{2}$$

Sitting on top of local search algorithms, GLS helps them to escape local optima in the following way. Whenever the local search algorithm settles in a local optimum, GLS augments the cost function by adding penalties to selected features. The novelty of GLS is mainly in the way that it selects features to penalize. The intention is to penalize "unfavorable features" or features that "matter most" when a local search settles in a local optimum. A feature with high cost has more impact on the overall cost. Another factor that should be considered is the current penalty value of that feature. The utility of penalizing feature i, $util_i$, under a local optimum s_*, is defined as follows:

$$util_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \tag{3}$$

where c_i is the cost and p_i is the current penalty value of feature i. In other words, if a feature is not exhibited in the local optimum (indicated by I_i), then the utility of penalizing it is 0. The higher the cost of this feature (the greater c_i), the greater the utility of penalizing it. Besides, the larger the number of times it has been penalized (the greater p_i), the lower the utility of penalizing it again. In a local optimum, the feature with the greatest $util$ value will be penalized. When a feature is penalized, its penalty value is always increased by 1. The scaling of the penalty is adjusted by λ.

By taking the cost and current penalty into consideration in selecting the feature to penalize, GLS focuses its search effort on more promising areas of the search

space: areas that contain candidate solutions that exhibit "good features," i.e., features involving lower cost. On the other hand, penalties help to prevent the search from directing all effort to any particular region of the search space.

Naturally, the choice of the features, their costs, and the setting of λ may affect the efficiency of a search. Experience shows that the features and their costs normally come directly from the objective function. In many problems, the performance of GLS is not too sensitive to the value λ. It means that not too much effort is required to apply GLS to a new problem. In certain problems, one needs expertise in selecting the features and the λ parameter. Research aiming to reduce the sensitivity of the λ parameter in such cases is reported in [54].

Implementation Guideline

A local search procedure for the particular problem is required for the algorithm to be implemented. Guided local search is repeatedly using this procedure to optimize the augmented objective function of the problem. Each time the local search procedure reaches a local minimum, the augmented objective function is modified by increasing the penalties of one or more of the features present in the local minimum. These features are selected by using the utility function (Eq. 3). The section below presents and explains the pseudo-code for implementing a guided local search method.

Algorithm 1 depicts the pseudo-code for the guided local search procedure, where P is the problem, g is the objective function, h is the augmented objective function, λ is a parameter, I_i is the indicator function of feature i, c_i is the cost of feature i, M is the number of features, p_i is the penalty of feature i, $ConstructionMethod(P)$ is the method for constructing an initial solution for problem p, and $ImprovementMethod(s_k, h)$ is the method for improving solution s_k according to the augmented objective function h.

To understand the pseudo-code, let us first explain the methods for constructing and improving a solution, as they are both prerequisites for building a GLS algorithm.

Construction Method

As with other meta-heuristics, GLS requires a construction method to generate an initial (starting) solution for the problem. In the pseudo-code, this is denoted by $ConstructionMethod$. This method can generate a random solution or a heuristic solution based on some known technique for constructing solutions for the particular problem. GLS is not very sensitive to the starting solution given that sufficient time is allocated to the algorithm to explore the search space of the problem.

Improvement Method

A method for improving the solution is also required. In the pseudo-code, this is denoted by $ImprovementMethod$. This method can be a simple local search algorithm or a more sophisticated one such as variable neighborhood search [36],

Algorithm 1: The guided local search algorithm

GuidedLocalSearch$(P, g, \lambda, [I_1, \ldots, I_M], [c_1, \ldots, c_M], M)$

```
k ← 0;
s_0 ← ConstructionMethod(P);
{set all penalties to 0}
for i ← 1 until M do
    p_i ← 0;
end for
{define the augmented objective function}
h ← g + λ * Σ p_i * I_i;
while StoppingCriterion do
    s_{k+1} ← ImprovementMethod(s_k, h);
    {compute the utility of features}
    for i ← 1 until M do
        util_i ← I_i(s_{k+1}) * c_i/(1 + p_i);
    end for
    {penalize features with maximum utility}
    for all i such that util_i is maximum do
        p_i ← p_i + 1;
    end for
    k ← k + 1;
end while
s* ← best solution found with respect to objective function g;
return s*;
```

variable depth descent [49], ejection chains [32], or combinations of local search methods with exact search algorithms [60].

It is not essential for the improvement method to generate high-quality local minima. Experiments with GLS and various local heuristics reported in [82] have shown that High-quality local minima take time to produce, resulting in less intervention by GLS in the overall allocated search time. This may sometimes lead to inferior results compared to a simple but more computationally efficient improvement method.

Note also that the improvement method is using the augmented objective function instead of the original one.

Indicator Functions and Feature Penalization

Given that a construction and an improvement method are available for the problem, the rest of the pseudo-code is straightforward to apply. The penalties of features are initialized to zero, and they are incremented for features that maximize the utility formula, after each call to the improvement method.

The indicator functions I_i for the features rarely need to be implemented. Looking at the values of the decision variables can directly identify the features present in a local minimum. When this is not possible, data structures with constant time deletion/addition operations (e.g., based on double-linked lists) can incrementally maintain the set of features present in the working solution, thus avoiding the need for an expensive computation when GLS reaches a local minimum.

The selection of features to penalize can be efficiently implemented by using the same loop for computing the utility formula for features present in the local minimum (the other features can be ignored) and also placing features with maximum utility in a vector. With a second loop, the features with maximum utility contained in this vector have their penalties incremented by one.

Parameter λ

Parameter λ is the only parameter of the GLS method (at least in its basic version) and in general is instance dependent. Fortunately, for several problems, it has been observed that good values for λ can be found by dividing the value of the objective function of a local minimum with the number of features present in it. In these problems, λ is dynamically computed after the first local minimum and before penalties are applied to features for the first time. The user only provides parameter α, which is relatively instance independent (i.e., tuning α can result in λ values, which work for many instances of a problem class). The recommended formula for λ as a function of α is the following:

$$\lambda = \alpha * g(x^*)/(\text{no. of features present in } x^*) \tag{4}$$

where g is the objective function of the problem and x_* is a local minimum. Tuning α can result in λ values, which work for many instances of a problem class. Another benefit from using α is that, once tuned, it can be fixed in industrialized versions of the software, resulting in ready-to-use GLS algorithms for the end user.

Augmented Objective Function and Move Evaluations

With regard to the objective function and the augmented objective function, the program should keep track of the actual objective value in all operations related to storing the best solution or finding a new best solution. Keeping track of the value of the augmented objective value (e.g., after adding the penalties) is not necessary since local search methods will be looking only at the differences in the augmented objective value when evaluating moves.

However, the move evaluation mechanism needs to be revised to work efficiently with the augmented objective function. Normally, the move evaluation mechanism is not directly evaluating the objective value of the new solution generated by the move. Instead, it calculates the difference Δg in the objective function. This difference should be combined with the difference in the amount of penalties. This can be easily done and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only features whose state changes (being deleted or added). The penalties of the features deleted are summed together.

The same is done for the penalties of added features. The change in the amount of penalties due to the move is then simply given by the difference:

$$\sum_{\text{over all features } j \text{ added}} p_j - \sum_{\text{over all features } k \text{ deleted}} p_k \tag{5}$$

which then has to be multiplied by λ and added to Δg.

Another minor improvement is to monitor the actual objective value not only for the solutions accepted by the local search but also for those evaluated. Since local search is using the augmented objective function, a move that generates a new best solution may be missed. From our experience, this modification does not improve significantly the performance of the algorithm although it can be useful when GLS is used to find new best-known solutions to hard benchmark instances.

Stopping Criterion

There are many choices possible for the $StoppingCritetion$. Since GLS is not trapped in local minima, it is not clear when to stop the algorithm. Like other metaheuristics, we usually resort to a measure related to the length of the search process. For example, we may choose to set a limit on the number of moves performed, the number of moves evaluated, or the CPU time spent by the algorithm. If a lower bound is known, we can utilize it as a stopping criterion by setting the gap to be achieved between the best-known solution and the lower bound. Criteria can also be combined to allow for a more flexible way to stop the GLS method.

Possible Features for Common Applications

Applying guided local search to a problem requires identifying a *suitable* set of features to guide the search process. Features provide the heuristic search expert with quite a powerful tool since any solution property can be potentially captured and used to guide local search. Usually, we are looking for solution properties, which have a direct impact on the objective function. These can be modeled as features with costs equal or analogous to their contribution to the objective function value. By applying penalties to features, GLS can guide the improvement method to avoid costly ("bad") properties, converging faster toward areas of the search space, which are of high quality.

Features are not necessarily specific to a particular problem, and they can be used in several problems of similar structure. Real-world problems, which sometimes incorporate elements from several academic problems, can benefit from using more than one feature set to guide the local search in better optimizing the different terms of a complex objective function.

Below, we provide examples of useful features for several representative problems from various domains.

Routing/Scheduling Problems

In routing/scheduling problems, one is seeking to minimize the time required by a vehicle to travel between customers or for a resource to be set up from one activity to the next. Problems in this category include the traveling salesman problem, vehicle routing problem, machine scheduling with sequence-dependent setup times, and others.

Travel or setup times are modeled as edges in a path or graph structure commonly used to represent the solution of these problems. The objective function (or at least part of it) is given by the sum of lengths for the edges used in the solution.

Edges are ideal GLS features. A solution either contains an edge or not. Furthermore, each edge has a cost equal to its length. We can define a feature for each possible edge and assign a cost to it equal to the edge length. GLS quickly identifies and penalizes long and costly edges guiding local search to high-quality solutions, which use as much as possible the short edges available.

Assignment Problems

In assignment problems, a set of items has to be assigned to another set of items (e.g., airplanes to flights, locations to facilities, people to work, etc.). Each assignment of item i to item j usually carries a cost, and depending on the problem, a number of constraints are required to be satisfied (e.g., capacity or compatibility constraints). The assignment of item i to item j can be seen as a solution property which is either present in the solution or not. Since each assignment also carries a cost, this is another good example of a feature to be used in a GLS implementation.

In some variations of the problem such as the quadratic assignment problem, the cost function is more complicated, and assignments have an indirect impact on the cost. Even in these cases, we found that GLS can generate good results by assigning the same feature costs to all features (e.g., equal to 1 or some other arbitrary value). Although GLS is not guiding the improvement method to good solutions (since this information is difficult to extract from the objective function), it can still diversify the search because of the penalty memory incorporated, and it is capable of producing results comparable to popular heuristic methods.

Resource Allocation Problem

Assignment problems can be used to model resource allocation applications. A special but important case in resource allocation is when the resources available are not sufficient to service all requests. Usually, the objective function will contain a sum of costs for the unallocated requests, which is to be minimized. The cost incurred when a request is unallocated will reflect the importance of the request or the revenue lost in the particular scenario.

A possible feature to consider for these problems is whether a request is unallocated or not. If the request is unallocated, then a cost is incurred in the objective function, which we can use as the feature cost to guide local search. The number of features in a problem is equal to the number of requests that may be left unallocated, one for each request. There may be hard constraints which state that certain requests should always be allocated a resource, in which case there is

no need to define a feature for them. Problems in this category include the path assignment problem [8], maximum channel assignment problem [68], workforce scheduling problem [9, 15], and others.

Constrained Optimization Problems

Constraints are very important in capturing processes and systems in the real world. A number of combinatorial optimization problems deal with finding a solution, which satisfies a set of constraints or, if that is not possible, minimizes the number of constraint violations (relaxations). Constraint violations may have costs (weights) associated with them, in which case the sum of constraint violation costs is to be minimized.

Local search usually considers the number of constraint violations (or their weighted sum) as the objective function even in cases where the goal is to find a solution which satisfies all the constraints. Constraints by their nature can be easily used as features. They can be modeled by indicator functions, and they also incur a cost (i.e., when violated/relaxed), which can be used as their feature cost. Problems which can benefit from this modeling include the constraint satisfaction and partial constraint satisfaction problem, the famous SAT and its MAX-SAT variant, graph coloring, various frequency assignment problems [1, 56], and others.

Guided Fast Local Search

In this section, we look at the combination of guided local search with fast local search, a generalized algorithm for speeding up local search, resulting in the guided fast local search method. Guided fast local search addresses the issue of slow local search procedures, and it is particularly useful when applying GLS to tackle large-scale problem instances, especially when repeatedly and exhaustively searching the whole neighborhood is computationally expensive.

One factor which affects the efficiency of a local search algorithm is the size of the neighborhood. If too many neighbors are considered, then the search could be very costly. This is especially true if the search takes many steps to reach a local optimum and/or each evaluation of the objective function requires a significant amount of computation. Bentley presented in [12] the *approximate* 2-Opt method to reduce the neighborhood of 2-Opt in the TSP. We have generalized this method to a method called *fast local search* (FLS). The principle is to use heuristics to identify (and ignore) neighbors that are unlikely to lead to improving moves in order to enhance the efficiency of a search.

The neighborhood chosen for the problem is broken down into a number of small sub-neighborhoods, and an activation bit is attached to each one of them. The idea is to scan continuously the sub-neighborhoods in a given order, searching only those with the activation bit set to 1. These sub-neighborhoods are called active sub-neighborhoods. Sub-neighborhoods with the bit set to 0 are called inactive sub-neighborhoods, and they are not being searched. The neighborhood search process does not restart whenever we find a better solution, but it continues with the next

sub-neighborhood in the given order. This order may be static or dynamic (i.e., change as a result of the moves performed).

Initially, all sub-neighborhoods are active. If a sub-neighborhood is examined and does not contain any improving moves, then it becomes inactive. Otherwise, it remains active and the improving move found is performed. Depending on the move performed, a number of other sub-neighborhoods are also activated. In particular, we activate all the sub-neighborhoods where we expect other improving moves to occur as a result of the move just performed. As the solution improves, the process dies out with fewer and fewer sub-neighborhoods being active until all the sub-neighborhood bits turn to 0. The solution formed up to that point is returned as an approximate local optimum.

The overall procedure could be many times faster than conventional local search. The bit setting scheme encourages chains of moves that improve specific parts of the overall solution. As the solution becomes locally better, the process is settling down, examining fewer moves and saving enormous amounts of time which would otherwise be spent on examining predominantly bad moves.

Although FLS procedures do not generally find very good solutions, when they are combined with GLS, they become very powerful optimization tools. Combining GLS with FLS is straightforward. The key idea is to associate features to sub-neighborhoods. The associations to be made are such that for each feature we know which sub-neighborhoods contain moves that have an immediate effect upon the state of the feature (i.e., moves that remove the feature from the solution).

By reducing the size of the neighborhood, one may significantly reduce the amount of computation involved in each local search iteration. The idea is to enable more local search iterations in a fixed amount of time. The danger of ignoring certain neighbors is that some improvements may be missed. The hope is that the gain in "search speed" outweighs the loss in "search quality."

Implementation Guideline

Guided fast local search (GFLS) is more sophisticated than the basic GLS algorithm as it uses a number of sub-neighborhoods, which are enabled/disabled during the search process. The main advantage of GFLS lies in its ability to focus the search after the penalties of features are increased. This can dramatically shorten the time required by an improvement method to re-optimize the solution each time the augmented objective function is modified.

The following provide the pseudo-code for the method and also some suggestions on how to achieve an efficient implementation. We first look at the pseudo-code for fast local search, which is part of the overall guided fast local search algorithm.

Fast Local Search

The pseudo-code for fast local search is given in Algorithm 2, where s is the solution, h is the augmented objective function, L is the number of sub-neighborhoods, bit_i is the activation bit for sub-neighborhood i, $MovesForSubneighborhood(i)$

is the method which returns the set of moves contained in sub-neighborhood i, and $SubneighborhoodsForMove(m)$ is the method which returns the set of sub-neighborhoods to activate when move m is performed.

Algorithm 2: The fast local search algorithm

FastLocalSearch(s, h, $[bit_1, \ldots, bit_L]$, L)

```
while ∃bit, bit = 1 do
  {i.e., while active sub-neighborhood exists}
  for i ← 1 until L do
    if bit_i = 1 then
      {search sub-neighborhood i}
      Moves ← MovesForSubneighbourhood(i);
      for all move m in Moves do
        s' ← m(s);
        {s' is the result of move m}
        if h(s') < h(s) then
          {spread activation}
          ActivateSet ← SubneighbourhoodsForMove(m);
          for all sub-neighborhood j in ActivateSet do
            bit_j ← 1;
          end for
          s ← s';
          GOTO: ImprovingMoveFound
        end if
      end for
      {no improving move found}
      bit_i ← 0;
    end if
    ImprovingMoveFound: continue
  end for
end while
return s;
```

As explained earlier, the problem's neighborhood is broken down into a number of sub-neighborhoods, and an activation bit is attached to each one of them. The idea is to examine sub-neighborhoods in a given order, searching only those with the activation bit set to 1. The neighborhood search process does not restart whenever we find a better solution, but it continues with the next sub-neighborhood in the given order. The pseudo-code given above is flexible since it does not specify which bits are initially switched on or off, something which is an input to the procedure.

This allows the procedure to be focused to certain sub-neighborhoods and not the whole neighborhood, which may be a large one.

The procedure has two points that need to be customized. The first is the breaking down of the neighborhood into sub-neighborhoods (*MovesForSubneighborhood* method in pseudo-code). The second is the mapping from moves to sub-neighborhoods for spreading activation (*SubneighbourhoodsForMove* method in pseudo-code). Both points are strongly dependent on the move operator used.

In general, the move operator depends on the solution representation. Fortunately, several problems share the same solution representation which is typically based on some well-known simple or composite combinatorial structure (e.g., selection, permutation, partition, composition, path, cyclic path, tree, graph, etc.). This allows us to use the same move operators for many different problems (e.g., 1-Opt, 2-Opt, swaps, insertions, etc.).

The method for mapping sub-neighborhoods to moves, which is denoted in the pseudo-code by *SubneighbourhoodToMoves*, can be defined by looking at the implementation of a typical local search procedure for the problem. This implementation, at its core, will usually contain a number of nested for-loops for generating all possible move combinations. The variable in the outermost loop in the move generation code can be used to define the sub-neighborhoods. The moves in each sub-neighborhood will be those generated by the inner loops for the particular sub-neighborhood index value at the outermost loop.

In general, the sub-neighborhoods can be overlapping. Fast local search is usually examining a limited number of moves compared to exhaustive neighborhood search methods, and therefore duplication of moves is not a problem. Moreover, this can be desirable sometimes to give a greater range to each sub-neighborhood. Since not all sub-neighborhoods are active in the same iteration, if there is no overlapping, some improving moves may be missed.

The method for spreading activation, denoted by *SubneighbourhoodsForMove*, returns a set of sub-neighborhoods to activate after a move is performed. The lower bound for this set is the sub-neighborhood where the move originated. The upper bound (although not useful) is all the sub-neighborhoods in the problem.

A way to define this method is to look at the particular move operator used. Moves will affect part of the solution directly or indirectly while leaving other parts unaffected. If a sub-neighborhood contains affected parts, then it needs to be activated since an opportunity could arise there for an improving move as a result of the original move performed.

The fast local search loop is settling down in a local minimum when all the bits of sub-neighborhoods turn to zero (i.e., no improving move can be found in any of the sub-neighborhoods). Fast local search in that respect is similar to other local search procedures. The main differences are that the method can be focused to search particular parts of the overall neighborhood and, secondly, it is working in an opportunistic way looking at parts of the solution which are likely to contain improving moves rather than the whole solution. In the next section, we look at guided fast local search, which uses fast local search as its improvement method.

Guided Fast Local Search

The pseudo-code for guided fast local search is given in Algorithm 3, where $FastLocalSearch$ is the fast local search method as described in section "Fast Local Search", $SubneighbourhoodsForFeature(i)$ is the method which returns the set of sub-neighborhoods to activate when feature i is penalized, and the rest of the definitions are the same than those used in the pseudo-code for GLS described in section "Implementation Guideline".

Algorithm 3: The guided fast local search algorithm

GuidedFastLocalSearch(p, g, λ, $[I_1, \ldots, I_M]$, $[c_1, \ldots, c_M]$, M, L)

```
k ← 0;
s_0 ← ConstructionMethod(p);
{set all penalties to 0}
for i ← 1 until M do
    p_i ← 0;
end for
{set all sub-neighborhoods to the active state}
for i ← 1 until L do
    bit_i ← 1;
    {define the augmented objective function}
end for
h ← g + λ * ∑ p_i * I_i;
while StoppingCriterion do
    s_{k+1} ← FastLocalSearch(s_k, h, [bit_1, ..., bit_L], L);
    {compute the utility of features}
    for i ← 1 until M do
        util_i ← I_i(s_{k+1}) * c_i/(1 + p_i);
        {penalize features with maximum utility}
    end for
    for all i such that util_i is maximum do
        p_i ← p_i + 1;
        {activate sub-neighborhoods related to penalized feature i}
        ActivateSet ← SubneighbourhoodsForFeature(i);
        for all sub-neighborhood j in ActivateSet do
            bit_j ← 1;
        end for
    end for
    k ← k + 1;
end while
s* ← best solution found with respect to objective function g;
return s*;
```

This pseudo-code is similar to that of GLS explained in section "Implementation Guideline". All differences relate to the manipulation of activation bits for the purpose of focusing fast local search. These bits are initialized to 1. As a result, the first call to fast local search is examining the whole neighborhood for improving moves. Subsequent calls to fast local search examine only part of the neighborhood and in particular all the sub-neighborhoods that relate to the features penalized by GLS.

Identifying the sub-neighborhoods that are related to a penalized feature is the task of *SubneighbourhoodsForFeature* method. The role of this method is similar to that of *SubneighbourhoodsForMove* method in fast local search (see section "Fast Local Search"). The *SubneighbourhoodsForFeature* method is usually defined based on an analysis of the move operator. After the application of penalties, we are looking for moves which remove or have the potential to remove penalized features from the solution. The sub-neighborhoods, which contain such moves, are prime candidates for activation. Specific examples will be given later in the chapter and in the context of GLS applications.

GLS Extensions, Hybrids, and Variations

GLS is closely related to other heuristic and meta-heuristic methods. In this section, we shall review the different variations, hybrids, and extensions of GLS and FLS that have been developed in recent years.

Extensions to GLS

GLS is closely related to other heuristic and meta-heuristic methods. This motivates the adoption of ideas borrowed from other meta-heuristics in GLS. For example, taboo lists and aspiration ideas from tabu search have been used in later versions of GLS. Resembling the tabu lists idea, a limited number of penalties are used when GLS is applied to the radio link frequency assignment problem [56]. When the list is full, old penalties are overwritten [81]. The motive is that if too many penalties are built up during the search, the local search could be misguided. In another GLS work, aspiration is used to favor promising moves [54].

Tairan and Zhang in [71] studied how to enhance the performance of GLS through designing a cooperative mechanism based on the proximate optimality principle (POP), resulting in a population-based GLS framework. The idea is to run multiple agents of GLS, and during the search the agents exchange their obtained information about the previous search to make their further search more rational. Based on POP, they suggested that common features that appear in many locally optimal solutions of GLS agents are more likely to be parts of the globally optimal solution. Thus, this property should be taken into consideration in the penalization stage during the search. The effectiveness of the proposed cooperative method was demonstrated through high-quality results obtained in the TSP.

GLS Hybrids

Being simple and general, GLS ideas can easily be combined with other techniques. GLS has been hybridized with several meta-heuristics creating efficient frameworks which were successfully applied to several applications. Below, we review and comment on some of these hybrids of GLS.

As a meta-heuristic method, GLS can also sit on top of genetic algorithms (GA). This has been demonstrated in guided genetic algorithm (GGA) [45–47]. GGA is a hybrid of GA and GLS. It is designed to extend the domain of both GA and GLS. One major objective is to further improve the robustness of GLS. It can be seen as a GA with GLS to bring it out of local optima: if no progress has been made after a specific of iterations (this number is a parameter of GGA), GLS modifies the fitness function (which is the objective function) by means of penalties, using the criteria defined in (Eq. 3). GA will then use the modified fitness function in future generations. The penalties are also used to bias crossover and mutation in GA – genes that contribute more to the penalties are more likely to be changed by these two GA operators. This allows GGA to be more focused in its search.

On the other hand, GGA can roughly be seen as a number of GLSs running in parallel from different starting points and exchanging material in a GA manner. The difference is that only one set of penalties is used in GGA, whereas parallel GLSs could have used one independent set of penalties per run. Besides, learning in GGA is more selective than parallel GLS: the updating of penalties is only based on the best chromosome found at the point of penalization.

GLS was hybridized with two major evolutionary computation (EC) techniques, namely, estimate distribution algorithm (EDA) and evolution strategy (ES). The hybrid of GLS with EDA was introduced by Zhang et al. [88]. They proposed a framework that incorporates GLS within EDA, in which GLS is applied to each solution in the population of EDA. The framework is successfully applied to the quadratic assignment problem. The results show the superiority of EDA/GLS over GLS alone.

The hybrid of GLS with ES was first studied by Mester and Braysy [51]. The resulting framework combines GLS and ES into an iterative two-stage procedure. GLS is used in both phases to improve the local search in the first stage and to regulate the objective function and the neighborhood of the modified ES in the second stage. The principle of FLS is also incorporated into the idea of penalty variable neighborhood in which the neighborhood considered by the local search is limited to a small set of the neighbors of the penalized feature.

GLS has also been hybridized with variable neighborhood search (VNS) and large neighborhood search (LNS). Kytojoki et al. [43] combine GLS with VNS in an efficient variable neighborhood search heuristic, named guided VNS (GVNS), which was applied to the vehicle routing problem. The addition to VNS is the use of GLS to escape local minima. The idea of threshold value borrowed from threshold accepting (TA) is used as a termination condition for every GLS stage. The hybrid of GLS with LNS is introduced in [84]. In the proposed framework,

LNS is applied when the GLS cannot escape a local optimum after a number of penalizations, with the aim of increasing the diversity and exploring more promising parts of the search space. The effectiveness of this hybrid was demonstrated through high-quality results obtained in a planning optimization problem.

Guided tabu search (GTS) is a hybrid meta-heuristic which combines GLS with TS proposed by Tarantilis et al. [73, 74] to solve the vehicle routing problem with heterogeneous fleet and then extended to solve another variant of the same general problem. The basic idea is to control the exploration of TS by a guiding mechanism, based on GLS, that continuously modifies the objective function of the problem. The authors propose a new arc (as a feature) selection strategy which considers the relative arc length according to the rest of customers ($d_{i,j}/\text{avg}_{i,j}$ rather than $d_{i,j}$, where $\text{avg}_{i,j}$ is the average value of all outgoing arcs from i and j). They argue that this would lead to a more balanced arc selection, which should improve upon the most frequently employed strategy based on $d_{i,j}$ only. Experimental results confirm the effectiveness of GTS, producing new best results for several benchmarks. De Backer et al. [10] also proposed a guided tabu search hybrid in their work on the VRP.

GLS has been also successfully hybridized with ant colony optimization (ACO) by Hani et al. [35]. This hybrid algorithm was applied to the facility layout problem, a variant of the quadratic assignment problem (QAP). The basic idea is simple: GLS sits on top of the basic LS in the ACO.

The hybridization of GLS and constraint programming (CP) was introduced by Gomes et al. [34]. This method, named guided constraint search, borrows ideas from GLS to improve the efficiency of pure CP methods. The basic principle is to use a fitness function to choose at each iteration only the N most promising values of each variable's domain, defining a subspace for the CP method. The selection strategy is inspired from GLS; for each pair, a utility function, penalty parameter, and cost are defined. At each iteration, those features (variable/value pairs) which were considered but did not belong to a new best solution are deemed bad features and are penalized.

Variations of GLS

The success of GLS motivated researchers to invent new algorithms inspired from GLS, borrowing the ideas of features, penalties, and utilities. Below, we briefly describe such GLS-inspired algorithms.

Partially based on GLS, which is a centralized algorithm, Basharu et al. [11] introduce an improved version for solving distributed constraint satisfaction problems. The distributed guided local search (Dis-GLS) incorporates additional heuristics to enhance its efficiency in distributed scenarios. The algorithm has been successfully applied to the distributed version of the graph coloring problem producing promising results compared to other distributed search algorithms.

Hifi et al. [37] introduced a variant of GLS by proposing a new penalization strategy. The principle is to distinguish two phases in the search process, namely, the penalty and normal phases. The search process switches between the two phases in order to either escape local optima or diversify the search to explore another feasible space. The computational results confirm the high quality of solutions obtained by the proposed variant.

Tamura et al. [72] propose an improved version of GLS, named the objective function adjustment (OA) algorithm which incorporates the idea of features (from GLS) alongside the concept of energy function.

GLS for Multi-objective Optimization

Most real-world optimization problems are multi-objective in nature. The multi-objective optimization problem (MOP) concerns the optimization of two or more objectives simultaneously. Instead of searching for a global optimum solution as in single-objective optimization problems, the search in MOPs targets a set of solutions representing the optimum set of trade-offs between the objectives. This set is known interchangeably as the Pareto optimum set or the efficient solutions, and the objective values of these solutions are located at the Pareto front (PF). Efficient solutions are nondominant solutions in the sense that improving the value of any one of their objectives must be at the expense of degrading the quality of one or more of the other objectives. Thus, all efficient solutions are considered equivalent as long as there is no further information regarding the relative importance of each of the objectives.

Pareto Local Search

A simple, intuitive adaptation of local search to contain multiple objectives, is to apply the Pareto domination as an acceptance criterion when comparing the current solution to the new one. An archive of nondominant solutions discovered during the search is maintained in order to produce an approximation to the PF. The algorithm stops when the neighborhood of all solutions in the archive have been explored, i.e., the archive is a Pareto local optimum set. Recently, this idea was termed Pareto local search (PLS) and has been developed and extended by several researchers such as in [58]. PLS can be outlined, from a high-level perspective, as follows:

1. It starts with a randomly or heuristically generated solution that is added to the $archive$.
2. A non-visited candidate solution in the $archive$ is randomly chosen, and its neighborhood is explored while applying a first-improvement strategy (i.e., any better solution found is accepted immediately, and its neighborhood is explored for further improvement). This is an iterative step which is applied every time a new better solution is found and accepted.

3. The $archive$ is updated with any nondominated neighbor, i.e., the new neighbor is added to the $archive$ only if it is nondominated by another solution in the $archive$, and then all existing solutions that are dominated by the new neighbor are removed from the $archive$.
4. The exploration of the neighborhood of the current solution stops when all of the neighbors are visited. When the exploration stops, the current solution is considered as a Pareto local optimum.
5. The current solution is marked as being visited.
6. The procedure continues at step 2, while there is a solution in the $archive$ that is not visited.

The PLS returns the $archive$, which is a *Pareto local optimal set*.

Guided Pareto Local Search

Guided Pareto local search (GPLS) [4, 6] extends the guidance approach in GLS to guide PLS to escape Pareto local optimum sets. The only change that GPLS makes to the underlying PLS is the replacement of each objective function g_i with an augmented objective function h_i during the evaluation of neighbor solutions. The augmented functions are not used in the updating procedure of the *archive*, which always depends only on the original objective functions.

In GPLS, the definition of features has to be derived from all objectives. Thus, a set (or multiple sets) of features has to be defined, which will be shared by all solutions. In multi-objective optimization problems, one should take into consideration two different scenarios:

1. All objectives have the same structure (e.g., putting an item in all knapsacks in the knapsack problem or an edge in the multi-objective TSP), and therefore they share one defined feature set. However, the cost of a feature varies according to a particular objective. In order to define the cost of a feature in this case, the influence of the feature on each objective has to be considered and modeled into a single cost function. Such models include (weighted) aggregating approaches and other general functions such as min, max, or mean.
2. A distinct feature set needs to be defined for each objective, and a cost is associated with each feature to describe its influence on the corresponding objective. In this case, features from all feature sets should be considered at the penalization phase to pick one or more features to penalize.

The guidance strategy that GLS employs relies on penalizing some features exhibited by the recent local optimum. As described in section "Guided Local Search", the novelty of GLS is mainly in the way it selects features to penalize. The target is to penalize "bad features" when the local search settles in a local optimum. Two factors affect the utility of a feature, namely, its cost and the frequency of penalizing this feature in previous penalizations.

GPLS deals with a Pareto local optimum set instead of a local optimum. A straightforward penalization strategy is to evaluate the utility of all features exhibited by *any* nondominant solution in the *archive* and penalize features with maximum utility (Eq. 3). However, this simple strategy does not incorporate any information from the *archive* (i.e., the Pareto local optimum set). An example of important information that can be extracted from the *archive* is the number of nondominant solutions that exhibit a particular feature. This is another factor that can be incorporated into the utility function, such that the more Pareto local optima that exhibit a feature, the greater its utility of penalization. Recall that when a feature is penalized, only those Pareto local optima that exhibit the penalized feature will have the chance to escape. Therefore, increasing the utility of penalizing a feature that occurs in many Pareto local optima would enhance the chance of escaping a Pareto local optimum set by restarting from more solutions. It would also help to prevent the search from directing all its efforts toward any particular region of the PF, which therefore leads to a better spread over the PF. Bad features, in terms of their cost, are hoped to be removed either during the next calls of PLS or by future penalization.

The utility function, as stated in Eq. 3, is redefined to incorporate the number of solutions in the Pareto local optimum set that exhibits this feature (γ_i):

$$util_i(archive) = I_i(archive) \times \frac{c_i \times (\gamma_i / |archive|)}{1 + p_i} \tag{6}$$

where *archive* is a Pareto local optimum set, $I_i(archive)$ indicates whether "at least" one solution in the *archive* exhibits feature i, c_i is the feature's cost, p_i is the penalty, and $|archive|$ is the size of the Pareto local optimum set. The feature with the greatest $util$ value will be penalized. When a feature is penalized, its penalty value is always incremented by 1.

This utility function redefines the term "bad feature." If a feature is not exhibited in the Pareto local optimum set (indicated by I_i), then the utility of penalizing it is 0. The higher the cost of this feature (the greater c_i) and the more nondominant solutions exhibiting it (the greater γ_i), the greater the utility of penalizing it. Furthermore, the more times that it has been penalized (the greater p_i), the lower the utility of penalizing it again.

Having penalized a feature, all solutions in the *archive* that exhibit this feature need to be marked as "non-visited," so as to be considered by the PLS method in the next iteration.

The lambda parameter λ is the only parameter to GLS which determines the scaling of the penalty. In multi-objective scenarios each objective ideally has its own lambda, which is calculated as a function of a local optimum with respect to the correspondent objective. Thus, GPLS requires a set of lambda parameters: $[\lambda_1, \cdots, \lambda_k]$, where λ_i is a parameter for the objective h_i. Recall that lambda can be dynamically computed after the first local optimum and before penalties are applied to features for the first time (Eq. 4).

GPLS proves its effectiveness to converge quite quickly, however, at the middle area of the Pareto front [5]. In [6], the speedy convergence property of GPLS is utilized, and the performance of GPLS is further enhanced by coupling it with an efficient initial solution set. Two GPLS-based frameworks are proposed, both of which require such an efficient initial solution set. The first framework applies the standard GPLS, after filling its archive with the initial set. The second framework is a parallel version of GPLS, where each independent GPLS run takes a solution from the initial set as a starting point. GPLS and its frameworks are successfully applied to the 0/1 multi-objective knapsack problem [5], the multi-objective TSP[4], and the empowerment-based workforce scheduling problem [4].

Other Attempts

Apart from GPLS, Alhindi and Zhang [2, 3] propose an idea of using GLS, as a heuristic local search procedure specific for single-objective optimization, in multi-objective evolutionary algorithms (MOEAs). In this work GLS and multi-objective evolutionary algorithm based on decomposition (MOEA/D) [87] are combined to propose a highly efficient algorithm for solving MOPs. To this end, a combination of MOEA/D with GLS, called MOEA/D-GLS, is proposed. In MOEA/D-GLS, a MOP is decomposed into a number of single-objective subproblems. The subproblems are optimized in parallel by using neighborhood information and problem-specific knowledge. In the proposed work, GLS alternates between those subproblems to help them escape local Pareto optimal solutions. More specifically, for a subproblem trapped in a local Pareto optimal solution, the GLS starts from its solution and constructs a transformation function (i.e., augmented function). Then, GLS optimizes the augmented function in order to identify a new better solution. The algorithm has been successfully applied to the multi-objective traveling salesman problem.

GLS Implementation on the Traveling Salesman Problem

For illustration, the implementation details of GLS on the TSP is discussed in this section. The TSP is chosen here as it is the most significant application of GLS. The full details about the application of GLS to the TSP is given in [82].

Problem Description

There are many variations of the TSP. Here, we examine the classic symmetric TSP. The problem is defined by N cities and a symmetric distance matrix $D = [d_{ij}]$ which gives the distance between any two cities i and j. The goal is to find a tour (i.e., closed path), which visits each city exactly once and is of minimum length. A tour can be represented as a cyclic permutation π on the N cities if we interpret

$\pi(i)$ to be the city visited after city i, $i = 1, \ldots, N$. The cost of a permutation is defined as

$$g(\pi) = \sum_{i=1}^{N} d_{i\pi(i)} \tag{7}$$

and gives the cost function of the TSP.

Local Search

Solution Representation

The solution representation usually adopted for the TSP is that of a vector which contains the order of the cities in the tour. For example, the i-th element of the vector will contain an identifier for the i-th city to be visited. Since the solution of the TSP is a closed path, there is an edge implied from the last city in the vector to the first one in order to close the tour. The solution space of the problem is made of all possible permutations of the cities as represented by the vector.

Construction Method

A simple construction method is to generate a random tour. If the above solution representation is adopted, then all that is required is a simple procedure, which generates a random permutation of the identifiers of the cities. More advanced TSP heuristics can be used if we require a higher-quality starting solution to be generated. This is useful in real-time/online applications where a good tour may be needed very early in the search process in case the user interrupts the algorithm. If there are no such concerns, then a random tour generator suffices since the GLS meta-heuristic tends to be relatively insensitive to the starting solution and capable of finding high-quality solutions even if it runs for a relatively short time.

Improvement Method

Most improvement methods for the TSP are based on the k-Opt moves. Using k-Opt moves, neighboring solutions can be obtained by deleting k edges from the current tour and reconnecting the resulting paths using k new edges. The k-Opt moves are the basis of the three most famous local search heuristics for the TSP, namely, 2-Opt [20], 3-Opt [48], and *Lin-Kernighan (LK)* [49].

The reader can consider using the simple 2-Opt method, which in addition to its simplicity is very effective when combined with GLS. With 2-Opt, a neighboring solution is obtained from the current solution by deleting two edges, reversing one of the resulting paths, and reconnecting the tour. In practical terms, this means reversing the order of the cities in a contiguous section of the vector or its remainder depending on which one is the shortest in length.

Computing incrementally the change in solution cost by a 2-Opt move is relatively simple. Let us assume that edges e1, e2 are removed and edges e3, e4 are

added with lengths d1, d2, d3, d4, respectively. The change in cost is the following:

$$d3 + d4 - d1 - d2 \tag{8}$$

When we discuss the features used in the TSP, we will explain how this evaluation mechanism is revised to account for penalty changes in the augmented objective function.

Guided Local Search

For the TSP, a tour includes a number of edges, and the solution cost (tour length) is given by the sum of the lengths of the edges in the tour (see Eq. (7)). As mentioned in section "Routing/Scheduling Problems", edges are ideal features for routing problems such as the TSP. First, a tour either includes an edge or not, and second, each edge incurs a cost in the objective function which is equal to the edge length, as given by the distance matrix $D = [d_{ij}]$ of the problem. A set of features can be defined by considering all possible undirected edges e_{ij} $(i = 1 \ldots N, j = i + 1 \ldots N, i \neq j)$ that may appear in a tour with feature costs given by the edge lengths d_{ij}. With each edge e_{ij} connecting cities i and j is attached a penalty p_{ij} initially set to 0 which is increased by GLS during the search. When implementing the GLS algorithm for the TSP, the edge penalties can be arranged in a symmetric penalty matrix $P = [p_{ij}]$. As mentioned in section "Guided Local Search", penalties have to be combined with the problem's objective function to form the augmented objective function which is minimized by local search. We therefore need to consider the auxiliary distance matrix:

$$D' = D + \lambda \cdot P = [d_{ij} + \lambda \cdot p_{ij}] \tag{9}$$

Local search must use D' instead of D in move evaluations. GLS modifies P and (through that) D' whenever the local search reaches a local minimum.

In order to implement this, we revise the incremental move evaluation formula (Eq. 8) to take into account the edge penalties and also parameter λ. If $p1$, $p2$, $p3$, $p4$ are the penalties associated with edges $e1$, $e2$, $e3$, and $e4$, respectively, the revised version of (Eq. 8) is as follows:

$$(d3 + d4 - d1 - d2) + \lambda * (p3 + p4 - p1 - p2) \tag{10}$$

Similarly, we can implement GLS for higher-order k-Opt moves.

The edges penalized in a local minimum are selected according to the utility function (Eq. 3), which for the TSP takes the form:

$$Util(tour, e_{ij}) = I_{e_{ij}}(tour) \cdot \frac{d_{ij}}{1 + p_{ij}}, \tag{11}$$

where

$$I_{e_{ij}}(tour) = \begin{cases} 1, & e_{ij} \in tour \\ 0, & e_{ij} \notin tour \end{cases} \tag{12}$$

The only parameter of GLS that requires tuning is parameter λ. Alternatively, we can tune the α parameter which is defined in section "Implementation Guideline" and is relatively instance independent. Experimenting with α on the TSP, we found that there is an inverse relation between α and local search effectiveness. Not so effective local search heuristics such as 2-Opt require higher α values compared to more effective heuristics such as 3-Opt and LK. This is probably because the amount of penalty needed to escape from local minima decreases as the effectiveness of the heuristic increases explaining why lower values for α (and consequently for λ which is a function of α) work better with 3-Opt and LK. For 2-Opt, the following range for α generates high-quality solutions for instances in the TSPLIB [62]:

$$1/8 \leq \alpha \leq 1/2 \tag{13}$$

The reader may refer to [82] for more details on the experimentation procedure and the full set of results.

Guided Fast Local Search

We can exploit the way local search works on the TSP to partition the neighborhood in sub-neighborhoods as required by guided fast local search. Each city in the problem may be seen as defining a sub-neighborhood, which contains all 2-Opt edge exchanges removing one of the edges adjacent to the city. For a problem with N cities, the neighborhood is partitioned into N sub-neighborhoods, one for each city in the instance.

The sub-neighborhoods to be activated after a move is executed are those of the cities at the ends of the edges removed or added by the move.

Finally, the sub-neighborhoods activated after penalization are those defined by the cities at the ends of the edge(s) penalized. There is a good chance that these sub-neighborhoods will include moves that remove one or more of the penalized edges.

GLS/GPLS Implementation on a Workforce Scheduling Problem

The workforce scheduling problem (WSP) is another application of GLS, which is chosen here as another illustrative example. It is chosen since both GLS and GPLS have been applied to two different versions of WSP. The implementation of GLS/GPLS components is provided here, and for full details, the reader is referred to [4, 7, 76].

Problem Description

The WSP is basically the problem of allocating a set of technicians (resources), $R = \{r_1, r_2, \ldots, r_{|R|}\}$, to a set of tasks, $T = \{t_1, t_2, \ldots, t_{|T|}\}$. A task t is described by a 5-tuple:

$$< c_t, dur_t, reqSkill_t, [start_t, end_t], loc_t >$$

where c_t is a predefined priority which determines its importance to the company. The higher the value of c_t, the more important the task will be, and $c_t \in \Re$. dur_t is the expected duration a technician requires to finish this task. Each task requires a technician with a particular skill $reqSkill_t \in SkillSet$, where $SkillSet$ is the set of all skills: $SkillSet = \{skill_1, \ldots, skill_{|S|}\}$. A task t must be serviced within a predefined time window described by $[start_t, end_t]$. Tasks are geographically distributed, and the location of a task is denoted by loc_t.

Each technician $r \in R$ is described by the following triple:

$$< [start_r, end_r], skills_r, loc_r >$$

Each technician has limited shift hours where the beginning and end of the shift are expressed by $[br_r, er_r]$. $skills_r$ denotes the skill(s) a technician has, where $skills_r \subseteq SkillSet$. There is no single depot for technicians to start from, as they can start from home or from one of predefined depots. The location of a technician is denoted by loc_r.

There are two sets of decision variables in the WSP: the allocation variables $X = \{x_{rt} | r \in R; t \in T; x_{rt} \in \{0, 1\}\}$ and the service times $ServTime = \{st_t | t \in T\}$. A variable x_{rt} is set to 1 if the technician r is allocated to the task t and 0 otherwise. A variable st_t denotes the start time of service for task t.

Having decided these variables, a set of routes $\pi = \{\pi_r | r \in R\}$ are defined. A route π_r is a sequence of tasks ($\subseteq T$) that are to be visited by technician r; $\pi_r = (\pi_{r1}, \cdots, \pi_{r\sigma_r}), 1 \leq \sigma_r \leq |T|$.

A main objective is to maximize the number of allocated tasks with respect to their priorities, while satisfying all assignment and routing constraints.

The WSP can, then, be mathematically modeled as follows:

$$\max \qquad \frac{\sum_{r \in R} \sum_{t \in T} c_t x_{rt}}{\sum_{t \in T} c_t} \tag{14}$$

subject to

$$\sum_{r \in R} x_{rt} \leqslant 1 \quad \forall t \in T \tag{15}$$

$$reqSkill_t \in skills_r \quad \forall x_{rt} = 1, t \in T, r \in R \tag{16}$$

$$start_t \leqslant st_t \quad \forall t \in T \tag{17}$$

$$st_t + dur_t \leqslant end_t \quad \forall t \in T \tag{18}$$

$$start_r \leqslant st_{\pi_{ri}} \quad \forall r \in R, i = \{1..\sigma_r\} \tag{19}$$

$$st_{\pi_{ri}} + dur_{\pi_{ri}} \leqslant end_r \quad \forall r \in R, i = \{1..\sigma_r\} \tag{20}$$

$$start_r + trvl_{loc_r loc_{\pi_{r1}}} \leqslant st_{\pi_{r1}} \forall r \in R \tag{21}$$

$$st_{t_i} + dur_{t_i} + trvl_{loc_{t_i} loc_{t_i+1}} \leqslant st_{t_{i+1}} \quad \forall t_i \in \pi_r, \pi_r \in \pi \tag{22}$$

The objective function is represented by (1) which is normalized by dividing by the sum of all the costs associated with tasks. Constraint (2) imposes that each task is visited at most once. The skill constraint is expressed in (3). The time window constraints of all tasks are assured by (4) and (5). (6) and (7) ensure that all tasks which are assigned to a technician must be within the technician's working time. Finally, constraints (8) and (9) ensure route validity by considering the traveling time between a technician's base location and the first task, as well as between subsequent tasks in technician's route.

Empowerment in Workforce Scheduling

Empowerment scheduling [4, 7] is a term introduced to refer to scheduling approaches that incorporate the empowerment concept in the scheduling system by involving technicians in the task assignment decision. The hope of empowering human resources is to boost their contribution and commitment to the organization. In [4], an empowerment scheduling model is proposed for the workforce scheduling problem, named as the EmS-WSP, by enabling technicians to express their requests/preferences and submit plans for consideration in the allocation process. Naturally all plans can be expressed as a constraint relationship. These constraints will be considered to be satisfied in the scheduling process. Workforce scheduling problems are very complex, and thus the promise of satisfying all the plans cannot be guaranteed. One direct way to handle employees' work plans is to treat them as soft constraints and associate a cost to each plan when it is not satisfied. In this case the task involves finding a schedule that satisfies the majority of constraints, rather than all the constraints. A key property of the empowerment scheduling model proposed in [7] is in the efficient strategy that dynamically determines the incurred cost of violating each plan (see [7] for more details).

Formally, every technician $r \in R$ is given an opportunity to provide a work plan (wp_r) every day. A plan is basically a constraint (ρ) by which a technician can describe the jobs he/she wants to undertake. For instance, a technician r might want to be allocated to jobs of a particular set of skills or jobs in a particular region. Each plan is associated with a weight (ω); in our case this is equal to the EP value of r.

An indicator y_r ($\forall r \in R$) is defined which is equal to 1 if the plan ρ_r is satisfied and 0 otherwise.

The WSP is reformulated in the EmS-WSP to not only maximize the productivity in terms of the number of served jobs (Eq. 14) but also to satisfy the maximum number of working plans. The formulation of the additional objective is as follows:

$$wp_r = < \rho_r, \omega_r > \quad \forall r \in R, \quad 0 \leqslant \omega_r \leqslant 1 \tag{23}$$

$$\max \quad \frac{\sum_{r \in R} y_r \omega_r}{\sum_{r \in R} \omega_r} \tag{24}$$

Local Search

Before going to the implementation details, it is worth mentioning that the two objectives in the EmS-WSP can be transformed to a single-objective using an aggregation approach and then solved using a single-objective optimizer such as GLS. Otherwise, a multi-objective optimizer such as GPLS can be used to optimize the two objectives simultaneously. Since both GLS and GPLS are similar in the definition of their basic components, the following implementation details are for both algorithms.

Solution Representation

A candidate solution (i.e., a possible schedule) by a permutation of the jobs. Each permutation is mapped into a schedule using the deterministic algorithm described below:
procedure **Evaluation** (input: one particular permutation of jobs)

1. For each job, order the qualified engineers in ascending order of the distances between their bases and the job (such orderings only need to be computed once and recorded for evaluating other permutations).
2. Process one job at a time, following their ordering in the input permutation. For each job x, try to allocate it to an engineer according to the ordered list of qualified engineers:
   2.1 To check if engineer g can do job x, make x the first job of g; if that fails to satisfy any of the constraints, make it the second job of g, and so on;
   2.2 If job x can fit into engineer g's current tour, then try to improve g's new tour (now with x in it): the improvement is done by a simple 2-Opt algorithm (see section "GLS Implementation on the Traveling Salesman Problem"), modified in such a way that only better tours which satisfy the relevant constraints will be accepted;
   2.3 If job x cannot fit into engineer g's current tour, then consider the next engineer in the ordered list of qualified engineers for x; the job is unallocated if it cannot fit into any engineer's current tour.
3. The costs of the input permutation, which is the cost (as defined in Eqs. 14 and 24) of the schedule thus created, are returned. These costs will be somehow aggregated in a single cost by GLS or maintained simultaneously by GPLS.

Improvement Method

Given a permutation, the local search is performed in a simple way: the pairs of jobs are examined one at a time. Two jobs are swapped to generate a new permutation to be evaluated (using the evaluation procedure above) and compared to the current permutation. Note here that since the problem is also close to the vehicle routing problem (VRP), one may follow a totally different approach considering VRP move operators such as insertions, swaps, etc. In this case, the solution representation and construction methods need to be revised. The reader may refer to other works (e.g., [10]) for more information on the application of GLS to the VRP.

Feature Definition

Since there are two objectives of different natures in the EmS-WSP, two sets of features need to be defined. As maximizing the total allocated tasks is an objective, each feature in the first set represents the inability to serving a task, so as to bias the local search to serve tasks of high importance. Thus, for each task t, we define a feature $I_t(schedule)$, which is equal to 1 if t is unallocated in $schedule$ and 0 otherwise. The cost of this feature is given by the task priority (c_t) which is equal to the cost incurred in the cost function (Eq. 14) when a task is unallocated.

The second objective concerns maximizing the total satisfied employees' plans, and therefore each feature in the second feature set represents the inability to satisfy a work plan, so as to bias the (Pareto) local search to satisfy plans of high weight. Thus, for each technician r in the problem, we define a feature $I_r(schedule) = y_r$, where y_r is equal to 1 if the work plan of r (ρ_r) is satisfied and 0 otherwise. The cost of this feature is given by the working plan's weight (ω) which is equal to the cost incurred in the cost function (Eq. 24) when a plan is unsatisfied.

Overview of GLS Applications

GLS and its descendants have been applied to a number of nontrivial problems and have achieved state-of-the-art results.

Routing/Scheduling Problems

One of the most successful application of GLS and FLS is the *traveling salesman problem* (TSP). The Lin-Kernighan algorithm (LK) is a specialized algorithm for the TSP that has long been perceived as the champion of this problem [49, 50]. We tested GLS + FLS + 2Opt against LK [82] on a set of benchmark problems from a public TSP library [62]. Given the same amount of time, GLS + FLS + 2Opt found better results than LK on average. The outstanding performance of GLS+FLS+2Opt was also demonstrated in comparison to simulated annealing [39], tabu search [41], and genetic algorithm [30] implementations for the TSP. One must be cautious

when interpreting such empirical results as they could be affected by many factors, including implementation details. But given that the TSP is an extensively studied problem, it takes something special for an algorithm to outperform the champions under any reasonable measure ("find the best results within a given amount of time" must be a realistic requirement). It must be emphasized that LK is specialized for the TSP, but GLS and FLS are much simpler general-purpose algorithms.

GLS extensions and hybrids have also been proposed for the TSP, including a population-based GLS [71] and the combination of GLS with memetic algorithms [38] and also with the dynamic programming-based dynasearch technique with encouraging preliminary results reported in [18]. The multi-objective TSP has been tackled by GPLS [4] and a GLS hybrid with MOEA/D [3].

Padron and Balaguer [57] have applied GLS to the related rural postman problem (RPP), Vansteenwegen et al. [78] applied GLS to the related team orienteering problem (TOP), and Mester et al. [52] applied the guided evolution strategy hybrid meta-heuristic to a genetic ordering problem (a Unidimensional Wandering Salesperson Problem, UWSP).

In a vehicle routing problem, one is given a set of vehicles, each with its specific capacity and availability, and a set of customers to serve, each with specific weight and/or time demand on the vehicles. The vehicles are grouped at one or more depots. Both the depots and the customers are geographically distributed. The task is to serve the customers using the vehicles, satisfying time and capacity constraints. This is a practical problem which, like many practical problems, is NP-hard.

Kilby et al. applied GLS to vehicle routing problems and achieved outstanding results [40]. As a result, their work was incorporated in Dispatcher, a commercial package developed by ILOG [10]. Recently, the application of GLS and its hybrids to the VRP have been considerably extended to several variants of the problem. GLS has been applied to the vehicle routing problem with backhauls and time windows [89] and to the capacitated arc routing problem [13]. Guided tabu search has been applied to the VRP with time window [73, 74], and also extended to other variants of the VRP, namely, the VRP with two-dimensional loading constraints [85], the VRP with simultaneous pickup and delivery [86], and the VRP with replenishment facility [74]. GLS with VNS [43] and GLS with ES [51] hybrids have been proposed to solve large-scale VRPs. A new hybridization between GLS and GA has been devised to effectively tackle the transit network design problem in [61]. The deployment of fiber optics in the telecommunication industry has been successfully optimized using GLS in [19] and [66].

Assignment Problems

The generalized assignment problem is a generic scheduling problem in which the task is to assign agents to jobs. Each job can only be handled by one agent, and each agent has a finite resource capacity that limits the number of jobs that it can be assigned to. Assigning different agents to different jobs bear different utilities. On the other hand, different agents will consume different amounts of resources when

doing the same job. In a set of benchmark problems, GGA found results as good as those produced by a state-of-the-art algorithm (which was also a GA algorithm) by Chu and Beasley [17], with improved robustness [47].

GLS hybrids have been proposed for the related QAP. Zhang et al. [88] proposed the GLS/EDA hybrid meta-heuristic. In addition, the hybrid of GLS with ACO (ACO_GLS) has been applied to a variation of the QAP [35]. This encourages Daoud et al. [21] to couple GLS with ACO, GA, and PSO into three frameworks which are successfully applied to the robotic assembly line balancing problem.

In the *radio link frequency assignment problem* (RLFAP), the task is to assign available frequencies to communication channels satisfying constraints that prevent interference [14]. In some RLFAPs, the goal is to minimize the number of frequencies used. Bouju et al. [14] is an early work that applied GENET to radio length frequency assignment. For the CALMA set of benchmark problems, which has been widely used, GLS+FLS reported the best results compared to all work published previously [80]. In the NATO Symposium on RLFAP in Denmark, 1998, GGA was shown to improve the robustness of GLS [46]. In the same symposium, new and significantly improved results by GLS were reported [81]. At the time, GLS and GGA held some of the best-known results in the CALMA set of benchmark problems.

We have experimented with GLS and FLS on a variety of other problems, including the maximum channel assignment problem, a bandwidth packing problem variant, graph coloring, and the car sequencing problem. Some of their work are available for download over the Internet from Essex University's website [33] but are largely undocumented due to lack of time during the original development phase of the algorithm.

GGA was also successfully applied to the processor configuration problem, with new better results than those found by other previously reported algorithms [45,46].

GLS and FLS have been successfully applied to the 3-D bin packing problem and its variants [26, 44], VLSI design problems [27], and network planning problems [29, 84]. GLS has been applied to the natural language parsing problem [22], graph set T-coloring problem [16], query reformulation [55], and ligand docking problems via drug design [59]. Variations of GLS have been applied to graph coloring [11] and the multidimensional knapsack problem [37, 70]. The multi-objective 0/1 knapsack problem has been tackled using GPLS in [6].

Resource Allocation Problems

In the *workforce scheduling problem* (WSP) [9], the task is to assign technicians from various bases to serve the jobs, which may include customer requests and repairs, at various locations. Customer requirements and working hours restrict the service times at which certain jobs can be served by certain technicians. The objective is to minimize a function that takes into account the traveling cost,

overtime cost, and unserved jobs. In the WSP, GLS + FLS holds the best-published results for the benchmark problem available to the authors [76]. GPLS has also been applied to a multi-objective WSP in [4].

Constrained Optimization Problem

Given a set of propositions in conjunctive normal form, the satisfiability (SAT) problem is to determine whether the propositions can all be satisfied. The MAX-SAT problem is a SAT problem in which each clause is given a weight. The task is to minimize the total weight of the violated clauses. In other words, the weighted MAX-SAT problem is an optimization problem. Many researchers believe that many problems, including scheduling and planning, can be formulated as SAT and MAX-SAT problems; hence, these problems have received significant attention in recent years (e.g., see Gent et al. [31]).

GLSSAT, an extension of GLS, was applied to both the SAT and weighted MAX-SAT problem [53]. On a set of SAT problems from DIMACS, GLSSAT produced more frequently better or comparable solutions than those produced by WalkSAT [64], a variation of GSAT [65], which was specifically designed for the SAT problem.

On a popular set of benchmark weighted MAX-SAT problems, GLSSAT produced better or comparable solutions, more frequently than state-of-the-art algorithms, such as DLM [67], WalkSAT [64], and GRASP [63].

GLS in Commercial Packages

GLS and FLS have been incorporated into commercial software packages, namely, iOpt, which is a software toolkit for heuristic search methods [83], and iSchedule [24], which is an extension of iOpt for planning and scheduling applications (e.g., for solving problems such as the VRP [25]).

Conclusions

For many years, general heuristics for combinatorial optimization problems, with prominent examples such as simulated annealing and genetic algorithms, heavily relied on randomness to generate good approximate solutions to difficult NP-hard problems. The introduction and acceptance of tabu search [32] by the operations research community initiated an important new era for heuristic methods where deterministic algorithms exploiting historical information started to appear and to be used in real-world applications.

Guided local search described in this chapter follows this trend. While tabu search is a class of algorithms (where a lot of freedom is given to the management of the tabu list), GLS is more prescriptive (the procedures are more concretely

defined). GLS heavily exploits information (not only the search history) to distribute the search effort in the various regions of the search space. Important structural properties of solutions are captured by solution features. Solutions features are assigned costs, and local search is biased to spend its efforts according to these costs. Penalties on features are utilized for that purpose.

When local search settles in a local minimum, the penalties are increased for selected features present in the local minimum. By penalizing features appearing in local minima, GLS escapes the local minima visited (exploiting historical information) but also diversifies the choices, with regard to the various structural properties of solutions, as captured by the solution features. Features of high costs are penalized more often than features of low cost: the diversification process is directed and deterministic rather than undirected and random.

In general, several penalty cycles may be required before a move is executed out of a local minimum. This should not be viewed as an undesirable situation. It is caused by the uncertainty in the information as captured by the feature costs which forces the GLS to test its decisions against the landscape of the problem.

The penalization scheme of GLS is ideally combined with FLS which limits the neighborhood search to particular parts of the overall solution leading to the GFLS algorithm. GFLS significantly reduces the computation times required to explore the area around a local minimum to find the best escape route allowing many more penalty modification cycles to be performed in a given amount of running time.

Despite all the years of development, further research is possible to improve GLS and its related techniques further. The use of incentives implemented as negative penalties, which encourage the use of specific solution features, is one promising direction to be explored. Other interesting directions include *fuzzy features* with indicator functions returning real values in the [0, 1] interval, automated tuning of the λ or α parameters, definition of effective termination criteria, alternative utility functions for selecting the features to be penalized, and also studies about the convergence properties of GLS.

It is relatively easy to adapt GLS and GFLS to the different problems examined in this chapter. Although local search is problem dependent, the other structures of GLS and also GFLS are problem independent. Moreover, a mechanical, step-by-step procedure is usually followed when GLS or GFLS is applied to a new problem (i.e., implement a local search procedure, identify features, assign costs, define sub-neighborhoods, etc.) This makes GLS and GFLS easier to use by nonspecialist software engineers.

Cross-References

▸ Iterated Local Search
▸ Tabu Search
▸ Variable Neighborhood Search

References

1. Aardal K, Van Hoesel S, Koster A, Mannino C, Sassano A (2007) Models and solution techniques for frequency assignment problems. Ann Oper Res 153(1):79–129
2. Alhindi A (2015) Multiobjective evolutionary algorithm based on decomposition with advanced local search methods. PhD thesis, Department of computer science, University of Essex, Colchester
3. Alhindi A, Zhang Q (2013) MOEA/D with guided local search: some preliminarily experimental results. In: 5th computer science and electronic engineering conference (CEEC), Colchester, pp 109–114
4. Alsheddy A (2011) Empowerment scheduling: a multi-objective optimization approach using guided local search. PhD thesis, Department of computer science, University of Essex, Colchester
5. Alsheddy A, Tsang EPK (2009) Guided pareto local search and its application to the 0/1 multi-objective knapsack problems. In: Metaheuristics international conference (MIC2009), Hamburg
6. Alsheddy A, Tsang EPk (2010) Guided pareto local search based frameworks for biobjective optimization. In: IEEE congress on evolutionary computation (CEC), Barcelona, pp 1–8
7. Alsheddy A, Tsang EPk (2011) Empowerment scheduling for a field workforce. J Sched 14(6):639–654
8. Anderson CA, Fraughnaugh K, Parker M, Ryan J (1993) Path assignment for call routing: an application of tabu search. Ann Oper Res 41:301–312
9. Azarmi N, Abdul-Hameed W (1995) Workforce scheduling with constraint logic programming. BT Technol J 13(1):81–94
10. Backer BD, Furnon V, Shaw P, Kilby P and Prosser P (2000) Solving vehicle routing problems using constraint programming and metaheuristics. J Heuristics 6(4):501–523
11. Basharu M, Arana I, Ahriz H (2005) Distributed guided local search for solving binary DisCSPs. In: Proceedings of FLAIRS 2005. AAAI Press, Clearwater Beach, pp 660–665
12. Bentley JJ (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Comput 4:387–411
13. Beullens P, Muyldermans L, Cattrysse D, Van Oudheusden D (2003) A guided local search heuristic for the capacitated arc routing problem. Eur J Oper Res 147(3):629–643
14. Bouju A, Boyce JF, Dimitropoulos CHD, vom Scheidt G, Taylor JG (1995) Intelligent search for the radio link frequency assignment problem. In: Proceedings of the international conference on digital signal processing, Limassol
15. Castillo-Salazar A, Landa-Silva D, Qu R (2016) Workforce scheduling and routing problems: literature survey and computational study. Ann Oper Res 239(1):39–67
16. Chiarandini M, Stutzle T (2007) Stochastic local search algorithms for graph set T-colouring and frequency assignment. Constraints 12(3):371–403
17. Chu P, Beasley JE (1997) A genetic algorithm for the generalized assignment problem. Comput Oper Res 24:17–23
18. Congram RK, Potts CN (1999) Dynasearch algorithms for the traveling salesman problem. In: Presentation at the travelling salesman workshop, CORMSIS, University of Southampton, Southampton
19. Cramer S, Kampouridis M (2015) Optimising the deployment of fibre optics using guided local search. In: Proceedings of the 2015 IEEE congress on evolutionary computation (CEC). IEEE Press, Sendai, pp 799–806
20. Croes A (1958) A method for solving traveling-salesman problems. Oper Res 5:791–812
21. Daoud S, Chehade H, Yalaoui F, Amodeo L (2014) Solving a robotic assembly line balancing problem using efficient hybrid methods. J Heuristics 20(3):235–259
22. Daum M, Menzel W (2002) Parsing natural language using guided local search. In: Proceedings of 15th European conference on artificial intelligence (ECAI-2002), Lyon, pp 435–439

23. Davenport A, Tsang EPK, Wang CJ, Zhu K (1994) GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In: Proceedings of 12th national conference for artificial intelligence (AAAI), Seattle, 325–330
24. Dorne R, Voudouris C, Liret A, Ladde C, Lesaint D (2003) iSchedule – an optimisation tool-kit based on heuristic search to solve BT scheduling problems. BT Technol J 21(4):50–58
25. Dorne R, Mills P, Voudouris C (2007) Solving vehicle routing using iOpt. In: Doerner KF et al (eds) Metaheuristics: progress in complex systems optimization. Operations research/computer science interfaces series, vol 39. Springer, New York, pp 389–408
26. Egeblad J, Nielsen B, Odgaard A (2007) Fast neighbourhood search for two- and three-dimensional nesting problems. Eur J Oper Res 183(3):1249–1266
27. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for final placement in VLSI design. J Heuristics 9:269–295
28. Flood MM (1956) The traveling-salesman problem. In: Operations research, vol 4. Columbia University, New York, pp 61–75
29. Flores Lucio G, Reed M, Henning I (2007) Guided local search as a network planning algorithm that incorporates uncertain traffic demands. Comput Netw 51(11):3172–3196
30. Freisleben B, Merz P (1996) A genetic local search algorithm for solving symmetric and asymmetric travelling salesman problems. In: Proceedings of the 1996 IEEE international conference on evolutionary computation. IEEE Press, Piscataway, pp 616–621
31. Gent IP, van Maaren H, Walsh T (2000) SAT2000, highlights of satisfiability research in the year 2000. Frontiers in artificial intelligence and applications. IOS Press, Amsterdam/Washington, DC
32. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Boston
33. GLS Demos (2008). http://cswww.essex.ac.uk/CSP/glsdemo.html
34. Gomes N, Vale Z, Ramos C (2003) Hybrid constraint algorithm for the maintenance scheduling of electric power units. In: Proceeding Of international conference on intelligent systems application to power systems (ISAP 2003), Lemnos
35. Hani Y, Amodeo L, Yalaoui F, Chen H (2007) Ant colony optimization for solving an industrial layout problem. Eur J Oper Res 183(2):633–642
36. Hansen P, Mladenović N, Todosijević R (2016) Variable neighborhood search: basics and variants. EURO J Comput Optim 1–32
37. Hifi M, Michrafy M, Sbihi A (2004) Heuristic algorithms for the multiple-choice multidimensional Knapsack problem. J Oper Res Soc 55:1323–1332
38. Holstein D, Moscato P (1999) Memetic algorithms using guided local search: a case study. In: Corne D, Glover F, Dorigo M (eds) New ideas in optimisation. McGraw-Hill, London, pp 235–244
39. Johnson D (1990) Local optimization and the traveling salesman problem. In: Proceedings of the 17th colloquium on automata languages and programming. Lecture notes in computer science, vol 443. Springer, London, pp 446–461
40. Kilby P, Prosser P, Shaw P (1999) Guided local search for the vehicle routing problem with time windows. In: Voss S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic, Boston, pp 473–486
41. Knox J (1994) Tabu search performance on the symmetric traveling salesman problem. Comput Oper Res 21(8):867–876
42. Koopman BO (1957) The theory of search, part III, the optimum distribution of search effort. Oper Res 5:613–626
43. Kytöjoki J, Nuortio T, Bräysy O, Gendreau M (2007) An efficient variable neighbourhood search heuristic for very large scale vehicle routing problems. Comput Oper Res 34(9): 2743–2757
44. Langer Y, Bay M, Crama Y, Bair F, Caprace JD, Rigo P (2005) Optimization of surface utilization using heuristic approaches. In: Proceedings of the international conference COMPIT'05, Hamburg, pp 419–425
45. Lau TL (1999) Guided genetic algorithm. PhD thesis, Department of computer science, University of Essex, Colchester

46. Lau TL, Tsang EPK (1998) Guided genetic algorithm and its application to the radio link frequency allocation problem. In: Proceedings of NATO symposium on frequency assignment, sharing and conservation in systems (AEROSPACE), Aalborg, AGARD, RTO-MP-13, paper No.14b
47. Lau TL, Tsang EPK (1998) The guided genetic algorithm and its application to the general assignment problem. In: IEEE 10th international conference on tools with artificial intelligence (ICTAI'98), Taiwan, pp 336–343
48. Lin S (1965) Computer solutions of the traveling-salesman problem. Bell Syst Tech J 44: 2245–2269
49. Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
50. Martin O, Otto SW (1966) Combining simulated annealing with local search heuristics. Ann Operat Res 63(1):57–75
51. Mester D, Bräysy O (2005) Active guided evolution strategies for large-scale vehicle routing problems with time windows. Comput Oper Res 32(6):1593–1614
52. Mester DI, Ronin YI, Nevo E, Korol AB (2004) Fast and high precision algorithms for optimization in large-scale genomic problems. Comput Biol Chem 28(4):281–290
53. Mills P, Tsang EPK (2000) Guided local search for solving SAT and weighted MAX-SAT problems. J Autom Reason 24:205–223
54. Mills P, Tsang E, Ford J (2003) Applying an extended guided local search to the quadratic assignment problem. Ann Ope Res 118:1–4/121–135
55. Moghrabi I (2006) Guided local search for query reformulation using weight propagation. Int J Appl Math Comput Sci (AMCS) 16(4):537–549
56. Murphey RA, Pardalos PM, Resende MGC (1999) Frequency assignment problems. In: Du D-Z, Pardalos P (eds) Handbook of combinatorial optimization. vol 4, Kluwer Academic, Boston
57. Padron V, Balaguer C (2000) New methodology to solve the RPP by means of isolated edge. In: Tuson A (ed) Cambridge conference tutorial papers. Young OR, vol 11. Operational Research Society, UK
58. Paquete L, Chiarandini M, Stützle T (2004) Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. In: Gandibleux X (ed) Metaheuristics for multiobjective optimisation, vol 535. Springer, Berlin/New York, pp 177–199
59. Peh S, Hong J (2016) GLSDock – drug design using guided local search. In: Proceedings of the 2016 international conference on computational science and its applications, Beijing, pp 11–21
60. Pesant G, Gendreau M (1999) A constraint programming framework for local search methods. J Heuristics 5(3)255–279
61. Rahman MK, Nayeem MA, Rahman MS (2015) Transit network design by hybrid guided genetic algorithm with elitism. In: Proceedings of the 2015 conference on advanced systems for public transport (CASPT), Rotterdam
62. Reinelt G (1991) A traveling salesman problem library. ORSA J Comput 3:376–384
63. Resende MGC, Feo TA (1996) A GRASP for satisfiability. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. DIMACS series on discrete mathematics and theoretical computer science, vol 26. American Mathematical Society, Providence, pp 499–520
64. Selman B, Kautz H (1993) Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: Proceeding of 13th international joint conference on AI, Chambéry, pp 290–295
65. Selman B, Levesque HJ, Mitchell DG (1992) A new method for solving hard satisfiability problems. In: Proceedings of AAAI-92, San Jose, pp 40–446
66. Shaghaghi A, Glover T, Kampouridis M, Tsang E (2013) Guided local search for optimal GPON/FTTP network design. In: Chaki N et al (eds) Computer networks & communications (NetCom): proceedings of the fourth international conference on networks & communications. Springer, New York, pp 255–263

67. Shang Y, Wah BW (1998) A discrete Lagrangian-based global-search method for solving satisfiability problems. J Glob Optim 12(1):61–99
68. Simon HU (1989) Approximation algorithms for channel assignment in cellular radio networks. In: Proceedings 7th international symposium on fundamentals of computation theory. Lecture notes in computer science, vol 380. Springer, Berlin/New York, pp 405–416
69. Stone LD (1983) The process of search planning: current approaches and continuing problems. Oper Res 31:207–233
70. Tairan N, Algarni A, Varghese J, Jan M (2015) Population-based guided local search for multidimensional knapsack problem. In: Proceedings of the 2015 fourth international conference on future generation communication technology (FGCT), Luton, pp 1–5
71. Tairan N, Zhang Q (2013) P-GLS-II: an enhanced version of the population-based guided local search. In: Proceedings of the 13th annual conference on genetic and evolutionary computation (GECCO), Dublin, pp 537–544
72. Tamura H, Zhang Z, Tang Z, Ishii M (2006) Objective function adjustment algorithm for combinatorial optimization problems. IEICE Trans Fundam Electron Commun Comput Sci E89-A:9:2441–2444
73. Tarantilis CD, Zachariadis EE, Kiranoudis CT (2008) A guided tabu search for the heterogeneous vehicle routeing problem. J Oper Res Soc 59(12):1659–1673
74. Tarantilis CD, Zachariadis EE, Kiranoudis CT (2008) A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. INFORMS J Comput 20(1)154–168
75. Tikhonov AN, Arsenin VY (1977) Solutions of ill-posed problems. Wiley, New York
76. Tsang EPK, Voudouris C (1997) Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Oper Res Lett 20(3):119–127
77. Tsang EPK, Wang CJ, Davenport A, Voudouris C, Lau TL (1999) A family of stochastic methods for constraint satisfaction and optimisation. In: Proceedings of the first international conference on the practical application of constraint technologies and logic programming (PACLP), London, pp 359–383
78. Vansteenwegen P, Souffriau W, Berghe G, Oudheusden D (2009) A guided local search metaheuristic for the team orienteering problem. Eur J Oper Res 196(1):118–127
79. Voudouris C (1997) Guided local search for combinatorial optimisation problems. PhD thesis, Department of computer science, University of Essex, Colchester
80. Voudouris C, Tsang EPK (1996) Partial constraint satisfaction problems and guided local search. In: Proceedings of PACT'96, London, pp 337–356
81. Voudouris C, Tsang E (1998) Solving the radio link frequency assignment problems using guided local search. In: Proceedings of NATO symposium on frequency assignment, sharing and conservation in systems (AEROSPACE), Aalborg, AGARD, RTO-MP-13, paper No. 14a
82. Voudouris C, Tsang EPK (1999) Guided local search and its application to the travelling salesman problem. Eur J Oper Res 113(2):469–499
83. Voudouris C, Dorne R, Lesaint D, Liret A (2001) iOpt: a software toolkit for heuristic search methods. In: Walsh T (ed) Practice of constraint programming – CP 2001, Paphos. Lecture notes in computer science, vol 2239, pp 716–729
84. Xiaohu T, Haubrich H-J (2005) A hybrid metaheuristic method for the planning of medium-voltage distribution networks. In: Proceedings of 15th power systems computation conference (PSCC 2005), Liege
85. Zachariadis E, Tarantilis C, Kiranoudis C (2009) A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. Eur J Oper Res 195(3):729–743
86. Zachariadis E, Tarantilis C, Kiranoudis C (2009) A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. Expert Syst Appl 36(2):1070–1081
87. Zhang Q, Hui L (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11:712–731

88. Zhang Q, Sun J, Tsang EPK, Ford J (2003) Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem. In: Bird of a feather workshops, genetic and evolutionary computation conference, Chicago
89. Zhong Y, Cole MH (2005) A vehicle routing problem with backhauls and time windows: a guided local search solution. Transp Res E Logist Transp Rev 41(2):131–144

Theory of Local Search

11

W. Michiels, E. H. L. Aarts, and J. Korst

Contents

Introduction 300
Solution Quality 302
Exact Neighborhood Functions 302
Exact Neighborhood Function for TSP: An NP-Completeness Proof 303
Performance Ratios of Neighborhood Functions 306
Non-approximability Results 310
From Neighborhood Function to Polynomial-Time Algorithm 311
Time Complexity 318
Computational Complexity of Local Search Problems 318
Proving PLS-Completeness 322
Time Complexity of Iterative Improvement 329
Ad Hoc Strategies 330
General Strategy 333
Relevance of PLS Beyond Local Search 337
Conclusion 338
References 338

W. Michiels (✉)
NXP, Eindhoven, The Netherlands

Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: wil.michiels@nxp.com; w.p.a.j.michiels@tue.nl

E. H. L. Aarts
Tilburg University, Tilburg, The Netherlands
e-mail: e.h.l.aarts@uvt.nl

J. Korst
Philips Research Laboratories, Eindhoven, The Netherlands
e-mail: jan.korst@philips.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_6

Abstract

Local search is a widely used method to solve combinatorial optimization problems. As many relevant combinatorial optimization problems are NP-hard, we often may not expect to find an algorithm that is guaranteed to return an optimal solution in a reasonable amount of time, i.e., in polynomial time. Therefore, one often resorts to heuristic methods that return good, suboptimal solutions in reasonable running times. Local search is a heuristic method that is popular for its ability to trade solution quality against computation time. By spending more time, we will generally get better solutions. Well-known examples of local search approaches are iterative improvement, simulated annealing, and tabu search.

The performance of local search, in terms of quality or running time, may be investigated empirically, probabilistically, and from a worst-case perspective. In this chapter we focus on the last option. That is, we give provable results on the worst-case performance of local search algorithms.

Besides combinatorial optimization problems, the theory discussed in this chapter also finds its application in game theory and computational complexity.

Keywords

Performance ratio · Time complexity · PLS · Iterative improvement

Introduction

Local search is a widely used method to solve combinatorial optimization problems. An instance of a combinatorial optimization problem P can be characterized by a pair (S, f), where S denotes a finite set of solutions and $f : S \to \mathbb{R}$ a cost function that assigns to each solution a real value that represents the quality of this solution. The problem is to find an optimal solution s^* in S, i.e., a solution for which $f(s^*) \leq f(s)$ for each $s \in S$ in case P is a minimization problem or a solution for which $f(s^*) \geq f(s)$ for each $s \in S$ in case P is a maximization problem.

As many combinatorial optimization problems are NP-hard, we often may not expect to find polynomial-time solution approaches that always return provably optimal solutions. In practice, one often has to settle for heuristic methods that return good, suboptimal solutions in reasonable running times. Local search is a heuristic method that is popular for its ability to trade solution quality against computation time. By spending more time, we will generally get better solutions. Well-known examples of local search approaches are iterative improvement, simulated annealing, and tabu search.

Loosely defined, a local search approach starts at an (often) arbitrarily chosen solution $s_{\text{init}} \in S$ and iteratively jumps to a (potentially) better solution in the neighborhood, until it ends at a locally optimal solution s_{end} for which – hopefully – $f(s_{\text{end}})$ is close to $f(s^*)$.

Using local search amounts to defining a neighborhood function $N : S \to P(S)$, where the power set $P(S)$ denotes the set of all subsets of solutions in S. For each solution $s \in S$, $N(s)$ gives the neighbors of s, i.e., the solutions one can jump to from S in one iteration of the local search approach.

Given S and N, the execution of a local search algorithm can be considered as a path in the directed neighborhood graph $G = (V, E)$, where each node $v \in V$ corresponds to a solution in S and $(v, w) \in E$ if and only if w corresponds to a neighbor of v. A local search algorithm specifies how s_{init} is chosen, how iteratively for the current solution s a neighboring solution from $N(s)$ is chosen as the next solution, and how eventually the algorithm will terminate. For example, iterative improvement starts with an initial solution and repeatedly selects for the given current solution s one of its neighbors $s' \in N(s)$ using some pivoting rule. If s' has a better quality than s, then s' is adopted as the new current solution. Otherwise, s is retained as the current solution. This procedure is repeated until the algorithm arrives at a local optimum, i.e., a solution that is better than any of its neighbors. This local optimum is then returned as the final solution. For a given local search approach, the transition graph is defined to be the subgraph of the neighborhood graph containing only the cost-improving edges.

The performance of a local search algorithm is determined by two aspects, namely, (1) the quality of the solutions it finally produces and (2) the time it takes to find these solutions (time complexity). The performance of a local search algorithm can be investigated in three ways:

- *empirical analysis*: investigate the performance by applying it to a representative set of problem instances.
- *probabilistic analysis*: assume a probability distribution over the set of all problem instances and use this to derive probabilistic results on solution quality and running time.
- *worst-case analysis*: investigate the performance by assuming worst-case assumptions.

Empirical and probabilistic analyses have the disadvantage that additional assumptions are required on how representative or how probable problem instances will be for a given application at hand and the information to base these assumptions upon may not be available. In this chapter we restrict ourselves to worst-case analysis.

Many of the examples that we give in this chapter relate to the traveling salesman problem (TSP). An instance of this well-known combinatorial optimization problem is defined by a set of n cities and an $n \times n$ distance matrix d, where $d_{ij} \in \mathbb{N}^+$ defines the distance from city i to city j. The problem is to find a tour visiting all n cities with minimum tour length. The TSP has three variants that will be considered in this chapter, namely, SYMMETRIC TSP, METRIC TSP, and EUCLIDEAN TSP. For the first variant, the distance matrix is required to be symmetric, i.e., $d_{ij} = d_{ji}$ for all i, j. For the second, the distance matrix should satisfy the triangle inequality, i.e.,

$d_{ik} \leq d_{ij} + d_{jk}$ for all i, j, k. And a TSP instance is said to belong to the third variant if the distance matrix d is defined by the Euclidean distance.

This chapter can be considered as a condensed version of two chapters from the monograph *Theoretical Aspects of Local Search* [16]. For a more extensive treatment of the subject, the reader is referred to this work. This chapter is organized in two parts. The first part considers solution quality, while the second part considers time complexity.

Solution Quality

In this section we study the guaranteed solution quality for iterative improvement. This corresponds to determining the quality of local optima. Ideally, we want iterative improvement to be guaranteed to end up in an optimal solution. This means that the neighborhood function N is exact. In the subsection below, we give examples of this. However, in most applications of local search, we do not have the luxury of a practical exact neighborhood function. There, we have to be satisfied with suboptimal solutions. In these cases, it is interesting to know how far the quality of the suboptimal solution can be from that of an optimal one. To this end, we say that a neighborhood function has a performance bound U if all its local optima are guaranteed to have a cost at most U times the optimal cost in case of a minimization problem and a cost at least $1/U$ times the optimal cost in case of a maximization problem. If bound U is tight, then U is called a performance ratio. We devote a subsection to analyzing the performance ratio of the k-change neighborhood function for Euclidean TSP. In the last two subsections, we finally elaborate on the relation between the formal hardness of finding an R-approximate solution and the existence of an efficient neighborhood function with a performance bound of R.

Exact Neighborhood Functions

A neighborhood function is said to be exact if each locally optimal solution is also globally optimal. In this section we present some examples of exact neighborhood functions. First of all, consider sorting. Sorting a multiset of n numbers $a_1, a_2, \ldots, a_n$ corresponds to finding a permutation π of $\{1, 2, \ldots, n\}$ that minimizes the cost function $f(\pi) = \sum_{i=1}^{n} i \cdot a_{\pi(i)}$ if the numbers have to be put in non-increasing order and that maximizes this cost function if they have to be put in non-decreasing order. A well-known algorithm for sorting is bubble sort. This algorithm starts with an arbitrary permutation π and then swaps repeatedly two adjacent numbers that are not in the right order. Bubble sort finds an optimal solution because a sequence of numbers is sorted if and only if any two adjacent numbers in the sequence are sorted correctly. Obviously, swapping two adjacent numbers improves the cost of a permutation if and only if they are not in the right order. This means that bubble sort corresponds to iterative improvement with the neighborhood

function that constructs neighbors by swapping two adjacent numbers. Because bubble sort solves sorting, this neighborhood function is exact.

In the minimum spanning tree (MST) problem, we are asked to find a spanning tree with minimum weight for an edge-weighted graph. This problem can also be solved by applying iterative improvement with the exact neighborhood function proposed in the following theorem, which is taken from Papadimitriou and Steiglitz [21].

Theorem 1. *Consider for MST the neighborhood function that generates a neighbor of a spanning tree T in the following way. First of all, we add an arbitrary edge to T. This produces a single cycle. From this cycle we delete an arbitrary edge, which gives a new spanning tree. This neighborhood function is exact.*

Proof. Suppose we have a spanning tree T that is a local optimum, but not a global optimum. To prove the theorem, we show that this yields a contradiction. Let T^* be an optimal spanning tree that has a maximum number of edges in common with T. Furthermore, let e be an edge with minimum weight from the set of edges that are in T^* but not in T. As any spanning tree has $n-1$ edges, such an edge exists. The removal of e splits T^* into two subtrees T_1^* and T_2^*. We now construct a neighbor T' of T as follows. First, we add edge e to T. This results in a cycle c. Besides e, this cycle must contain at least one other edge e' connecting a node from T_1^* with a node from T_2^*. We now construct T' by removing e' from c.

Because T is a local optimum, we have that the cost of T' is at least the cost of T. Hence, $w(e) \geq w(e')$, where $w(e)$ and $w(e')$ give the weights of edges e and e'. However, this implies that connecting T_1^* and T_2^* by edge e' results in a spanning tree that is at least as good as T^* and that has at least one edge more in common with T. This yields a contradiction. Hence, we conclude that the proposed neighborhood function is exact. □

Exact Neighborhood Function for TSP: An NP-Completeness Proof

Theoretically, all problems admit an exact neighborhood function. If we choose as neighborhood of a solution s the complete solution set, i.e., $N(s) = S$ for all $s \in S$, then this neighborhood function is exact. However, this neighborhood is not very practical since verifying whether a solution is locally optimal corresponds to solving the decision problem underlying the optimization problem. Mostly, we want to use local search for NP-hard problems, in which case this problem is NP-complete. In order to be practically useful, it makes sense to aim for a neighborhood function with the property that a single iteration of iterative improvement can be implemented to run in polynomial time. This is captured by the following definition.

Definition 1. A neighborhood function is said to be *polynomially searchable* if in polynomial time we can decide whether a solution is locally optimal and, if not, construct a better neighboring solution. □

The problem is how do we know whether or not an exact polynomially searchable neighborhood function exists for a problem? One way of settling this issue is to use complexity theory to prove for a particular problem that an exact neighborhood function does not exist. We now present the result from Papadimitriou and Steiglitz [20, 21], who show how this can be done for Metric TSP.

An outline of the proof is as follows. It is well known that Hamiltonian cycle (HC) is NP-complete, where HC is the problem of deciding for a graph whether it contains a Hamiltonian cycle. We first prove that the problem does not become easier if in addition to the graph we are also given a Hamiltonian path. Remember that a Hamiltonian cycle is a Hamiltonian path in which we return to the starting node. Hence, we prove that the following problem is NP-complete.

Definition 2 (Restricted Hamiltonian Cycle (RHC)). Given are a graph G and a Hamiltonian path in G. Is there a Hamiltonian cycle in G? □

Using this result, we next show that Metric TSP suboptimality is also NP-complete, which was our goal. □

Definition 3 (Metric TSP Suboptimality). Given are an instance I of Metric TSP and a tour τ for I. Is τ suboptimal? □

Lemma 1. *RHC is NP-complete.*

Proof. Proving RHC $\in$ NP is trivial. Hence, to prove the lemma, it now suffices to show that RHC is polynomially reducible from HC. Let $G = (V, E)$ be an arbitrary graph defining a problem instance of HC. The construction of a corresponding instance of RHC is based on the special-purpose subgraph depicted in Fig. 1. This subgraph, which consists of eight nodes, is called a diamond.

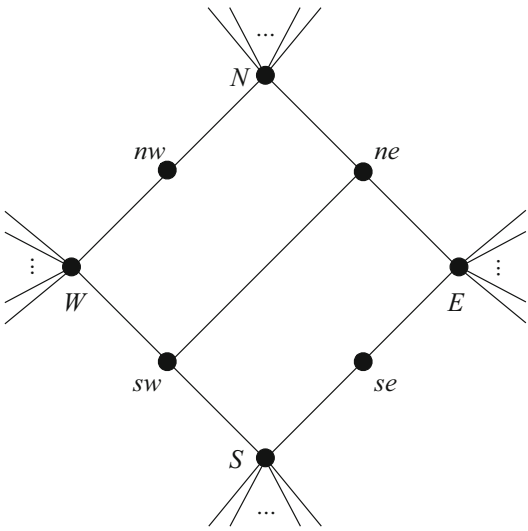


Fig. 1 A diamond subgraph

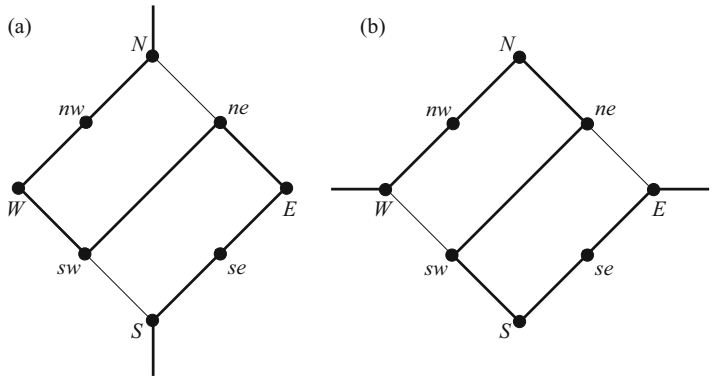


Fig. 2 (**a**) North-south mode. (**b**) East-west mode

In a diamond we identify four special nodes, called N (north), E (east), S (south), and W (west). We will use diamonds in such a way that if a graph G' contains a diamond D, then only these special nodes may be connected to other nodes in the graph. We now claim that if a graph G' contains a Hamiltonian cycle c, then for each diamond D in G' we have that its nodes are visited by c either in the way depicted in Fig. 2a or in the way depicted in Fig. 2b. The former case is referred to as north-south mode and the latter as east-west mode.

We now focus on proving this claim. Suppose that Hamiltonian cycle c enters diamond D in node N. We show that following any other path than the one depicted in Fig. 2a results in a contradiction. If node N is not succeeded by node nw (north-west) but by ne, then the only way for c to visit nw is via node W. However, this is not possible because, being in nw, we can neither go to N nor return to W as this would imply visiting N or W twice. Hence, N is indeed succeeded by nw in c. Obviously, the only way to proceed from nw is to go to W. Once at node W, we cannot leave D because then the only way for c to visit sw and ne is by skipping se. From node sw we have to go to ne because if we chose S this would make it impossible to visit ne as well as se. From ne we can then only go to E, and to prevent skipping node se, we conclude the traversal of D by visiting se and S, successively. By a similar reasoning, it can be verified that if c enters a diamond D via the south, then D is also traversed as depicted in Fig. 2a, and if c enters D via the east or the west, then D is traversed as depicted in Fig. 2b. This proves our claim.

We now construct for our arbitrary graph $G = (V, E)$ specifying the arbitrary HC instance the following corresponding graph $G' = (V', E')$. Let $V = \{1, 2, \ldots, n\}$. For each node $i \in V$, we introduce a diamond D_i with special nodes N_i, E_i, S_i, and W_i. Except for the last diamond D_n, we connect the south node of each diamond D_i with the north node of the next diamond D_{i+1}, i.e., we add the edges $\{S_i, N_{i+1}\}$ for all $1 \leq i < n$ to E'. Finally, we introduce for each edge $\{i, j\} \in E$ the two edges $\{W_i, E_j\}$ and $\{W_j, E_i\}$.

Obviously, the path that visits the diamonds in increasing order and traverses each diamond from north to south in the way depicted in Fig. 2a gives us a Hamiltonian path. Hence, to prove that we presented a valid polynomial-time reduction, it now suffices to show that G has a Hamiltonian cycle if and only if G' has one. We start with the "only if" part. Hence, suppose that G has a Hamiltonian cycle c. Then visiting the diamonds in G' in the order specified by c and the nodes inside a diamond in east-west mode yields a Hamiltonian cycle for G'. Next, assume that G' has a Hamiltonian cycle c. If c traverses one diamond in north-south mode, then it must traverse all diamonds in north-south mode as north and south nodes are not connected to east and west nodes. However, as only $n - 1$ north-south edges exist, no Hamiltonian cycle can exist that traverses each diamond in north-south mode. Hence, we can conclude that c must traverse each diamond in east-west mode. The order in which these diamonds are visited defines a Hamiltonian cycle for G. □

Theorem 2. *Metric TSP suboptimality is NP-complete.*

Proof. Again, proving membership in NP is a trivial task. We now show that RHC is polynomially reducible to Metric TSP suboptimality. Let $G = (V, E)$ with Hamiltonian path $p = (v_1, v_2, \ldots, v_n)$ define an arbitrary problem instance I of RHC. If $\{v_1, v_n\} \in E$, then we can directly conclude that I is a yes-instance of RHC. Hence, without loss of generality, we assume that $\{v_1, v_n\} \notin E$. We transform problem instance I into the following problem instance of Metric TSP suboptimality. The set of cities is given by $C = V = \{1, 2, \ldots, n\}$, and the distance d_{ij} between cities i to j is defined by 1 if $\{i, j\} \in E$ and by 2 if $\{i, j\} \notin E$. Hamiltonian path p gives us a tour τ with cost $n + 1$ because $\{v_i, v_{i+1}\} \in E$ with $1 \leq i < n$ and $\{v_1, v_n\} \notin E$. Now, G has a Hamiltonian cycle if and only if tour τ is suboptimal. To see this, suppose that τ is suboptimal. A tour τ' then exists with length n. This tour defines a Hamiltonian cycle in G. Conversely, if G contains a Hamiltonian cycle, then this cycle determines a tour with length n, which implies that τ is suboptimal. This proves the theorem. □

Corollary 1. *Provided that $P \neq NP$, no polynomially searchable exact neighborhood function exists for Metric TSP.* □

Performance Ratios of Neighborhood Functions

In case a problem does not admit a polynomially searchable exact neighborhood function, we have to resort to a neighborhood function for which local optima can be suboptimal. To know how far we can be off from an optimal solution if we use such a neighborhood function, we can try to derive bounds on the performance ratio of the neighborhood function. Observe that this performance ratio equals the performance ratio of iterative improvement where we select the initial solution randomly. If we select the initial solution more carefully, the performance ratio of iterative improvement may improve.

Consider Symmetric TSP. As the neighborhood size of k-change is polynomially bounded, a single iteration of k-Opt, which corresponds to a single step in the transition graph, can be implemented to run in polynomial time. Furthermore, finding a tour with length at most $2^{p(n)}$ times the optimal length is NP-hard for any fixed polynomial p [22]. Hence, if we were to prove that k-Opt is guaranteed to reach a locally optimal tour within a polynomial number of iterations, then we would obtain that, unless P = NP, k-Opt does not admit a performance ratio at most $2^{p(n)}$ for any fixed polynomial p. However, as we show in the next section, k-Opt may require an exponential number of iterations, which implies that the given reasoning is not applicable. The following theorem states that the lower bound of $2^{p(n)}$ on the performance ratio of k-Opt nevertheless holds. It even claims a stronger result: no function in n exists that gives an upper bound on the performance ratio of k-change. We note that this does not mean that it is impossible to bind the performance ratio. It can, for instance, be bounded trivially by the largest distance in the distance matrix d divided by the smallest distance in d.

Theorem 3. *For any fixed $k \geq 2$ and $n \geq 2k + 4$, the performance ratio of the k-change neighborhood function cannot be bounded by a constant for Symmetric TSP on n cities. This means that no function in n exists that gives a performance bound for k-change.*

Proof. For any $k \geq 2$ and $\varepsilon > 0$, we define a problem instance of Symmetric TSP containing $n = 2k + 4$ cities. For this problem instance, we construct a locally optimal tour $\hat{\tau}$, which has a performance ratio of $1 + \frac{1}{n\varepsilon}$. It is easy to extend this problem instance and to modify the locally optimal tour, such that the problem instance contains any $n > 2k + 4$ cities, the tour $\hat{\tau}$ remains locally optimal, and the performance ratio of $\hat{\tau}$ is still $1 + \frac{1}{n\varepsilon}$. Since for $\varepsilon \to 0$ this performance ratio approaches ∞, this proves the theorem.

We define the symmetric distance matrix d, such that all entries are given by $1 + \varepsilon$ except for the entries $d_{i,i+1}$ with $1 < i < n$, $d_{1,n}$, $d_{1,k+2+l}$ with $l = k \bmod 2$, and $d_{i,n-i+1}$ with $2 \leq i \leq k + 2$, which are all given by ε. In this definition we restrict ourselves without loss of generality to the entries d_{ij} with $i < j$.

Let $G = (V, E)$ be the graph obtained from the complete graph of n nodes by maintaining an edge $\{i, j\}$ if $d_{ij} = \varepsilon$ and by removing it if $d_{ij} = 1 + \varepsilon$; see Fig. 3. Obviously, a tour has length $n\varepsilon$ if it defines a Hamiltonian cycle in graph G and it has length at least $1 + n\varepsilon$ otherwise. Moreover, it can be proved that the tour $\tau^* = (1, n, n-1, 2, 3, n-2, n-3, \dots)$ defines the only Hamiltonian cycle in G, where tours that only differ in their starting position or direction are considered to be equivalent. Hence, tour τ^* is the only optimal tour, and the length of any other tour τ is at least $1 + n\varepsilon$. From this it follows that if a tour $\hat{\tau}$ has length $1 + n\varepsilon$ and if it has $k + 1$ edges that are absent in τ^*, then $\hat{\tau}$ is a locally optimal tour with performance ratio $\frac{1+n\varepsilon}{n\varepsilon} = 1 + \frac{1}{n\varepsilon}$. It can be verified that such a tour is given by $\hat{\tau} = (1, 2, \dots, n)$. This proves the theorem. □

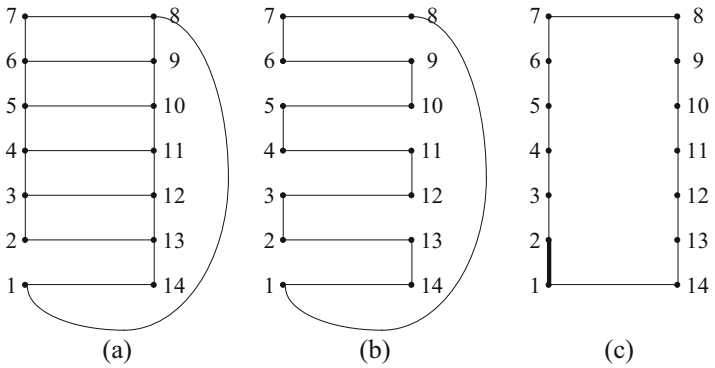


Fig. 3 For $k = 5$, the figure shows (**a**) graph G, (**b**) optimal tour τ^* of length $n\varepsilon$, and (**c**) locally optimal tour $\hat{\tau}$ of length $1 + n\varepsilon$

In the remainder of this section, we focus on Metric TSP. Although solving this problem is as difficult as solving Symmetric TSP (they are both strongly NP-hard), it is easier to find an approximate solution for Metric TSP than for Symmetric TSP. Currently, Christofides' algorithm [3] is the polynomial-time algorithm with the best known worst-case performance. This constructive algorithm works as follows. First of all, it computes a minimum spanning tree T. Next, it converts this tree into an Eulerian graph, which is a graph in which each node has an even degree. This is done by deriving a perfect matching on the nodes with an odd degree. These edges are added to T. For an Eulerian graph, we can easily determine an Eulerian tour, which is a tour that traverses each edge exactly once. By introducing shortcuts, we can eliminate multiple visits to the same city in the tour. This results in a tour for which it can be proved that its length is at most $\frac{3}{2}$ times the optimal length.

For k-Opt Chandra, Karloff, and Tovey [2] have proved that, for infinitely many n, a lower bound on the performance ratio is given by $\frac{1}{4}\sqrt{n}$ for $k = 2$ and by $\frac{1}{4}n^{\frac{1}{2k}}$ for $k \geq 3$. Hence, from a worst-case perspective, the local search algorithm k-Opt cannot compete against Christofides' constructive algorithm. Nevertheless, k-Opt may still be preferred over Christofides' algorithm as in practice we may be more interested in a good average-case performance than in a good worst-case performance. Hence, it is still interesting to analyze the worst-case performance of k-Opt or, equivalently, k-change, not only to obtain a worst-case performance guarantee but also to enhance our understanding of the algorithm.

Chandra, Karloff, and Tovey [2] also proved that k-Opt has a performance bound, i.e., an upper bound on the performance ratio, of $4\sqrt{n}$ for any $k \geq 2$. By a tighter analysis, Levin and Yovel [13] improved this bound to $2\sqrt{2n}$. Below, we prove this result. Note that this performance bound differs by a constant factor from the presented lower bound for $k = 2$ and by a factor that increases with n for any $k \geq 3$. In order to prove the bound, we need the following auxiliary result that states that a

locally optimal tour cannot contain too many long edges as expressed as a fraction of the optimal tour length.

Lemma 2. *For an arbitrary problem instance of Metric TSP, let l^* be the optimal tour length and $\hat{\tau}$ a locally optimal tour with respect to the 2-change neighborhood function. For any i with $1 \leq i \leq n$, we have $|E_i| < i/2$, where set E_i contains the edges $\{j, j'\}$ from $\hat{\tau}$ with $d_{jj'} > \frac{2l^*}{\sqrt{i}}$.*

Proof. We prove the lemma by contradiction. So, suppose we have $|E_i| \geq i/2$ for some i. Although the direction of $\hat{\tau}$ is irrelevant, we assume some orientation of the tour. As a result, the $r = |E_i|$ edges in E_i transform into r arcs (t_j, h_j) with $1 \leq j \leq r$. The cities t_j and h_j are called the tail and head of arc (t_j, h_j).

We first show that a tail t_u with $1 \leq u \leq r$ cannot be surrounded by too many other tails of arcs from E_i. Let V_u be the set of all tails of arcs from E_i that have a distance of at most $\frac{l^*}{\sqrt{i}}$ from t_u. Assume that $|V_u| \geq \sqrt{i}/2$. We prove that $|V_u| < \sqrt{i}/2$ by showing that this assumption yields a contradiction.

Let $t_j, t_{j'} \in V_u$. Because the triangle inequality holds and because, by definition, the distance from t_u to any of these two tails is at most $\frac{l^*}{\sqrt{i}}$, we obtain that the distance between t_j and $t_{j'}$ is at most $\frac{2l^*}{\sqrt{i}}$. This implies that the distance between the heads h_j and $h_{j'}$ is strictly larger than $\frac{2l^*}{\sqrt{i}}$. If this would not be the case, then replacing the edges $\{t_j, h_j\}$ and $\{t_{j'}, h_{j'}\}$ in $\hat{\tau}$ via a 2-change into $\{t_j, t_{j'}\}$ and $\{h_j, h_{j'}\}$ would result in a shorter tour because the length of each of the former two edges is strictly larger than $\frac{2l^*}{\sqrt{i}}$ by the definition of E_i. This would contradict the local optimality of $\hat{\tau}$.

Hence, $|V_u| \geq \sqrt{i}/2$ heads can be identified in E_i that all have a mutual distance of strictly more than $\frac{2l^*}{\sqrt{i}}$. This implies that the optimal tour on these heads is strictly more than l^*, which gives a contradiction because due to the triangle inequality it is not possible that an optimal tour over all cities is shorter than the optimal tour over a subset of all cities. This proves our claim $|V_u| < \sqrt{i}/2$. We use this result to derive a contradiction from the assumption $r \geq i/2$, which settles the proof of the lemma.

Consider the following labeling algorithm. Select an unlabeled tail t_u in E_i, and label all tails in E_i that have a distance of at most $\frac{l^*}{\sqrt{i}}$ from t_u (including t_u). Repeat this labeling step until all tails in E_i are labeled. As proved above, strictly fewer than $\sqrt{i}/2$ tails are labeled in each iteration. As $r \geq i/2$, this implies that the labeling procedure takes strictly more than $\sqrt{i}$ iterations. Obviously, the tails that are selected in these iterations all have a mutual distance strictly larger than $\frac{l^*}{\sqrt{i}}$. This implies that the optimal tour has length strictly larger than $\sqrt{i} \cdot \frac{l^*}{\sqrt{i}} = l^*$, which results in a contradiction. This proves the lemma. □

Using Lemma 2, we can now prove the claimed result.

Theorem 4. *The k-change neighborhood function has a performance bound of $2\sqrt{2n}$ for Metric TSP.*

Proof. To prove the theorem, it suffices to show that a tour $\hat{\tau}$ has a performance ratio of at most $2\sqrt{2n}$ if it is a locally optimal tour with respect to the 2-change neighborhood function. Lemma 2 yields that the ith largest edge in $\hat{\tau}$ has a length of at most $\frac{\sqrt{2}l^*}{\sqrt{i}}$, where l^* is the optimal tour length. Hence, the length of $\hat{\tau}$ is at most

$$\sum_{i=1}^{n} \frac{\sqrt{2}l^*}{\sqrt{i}} = \sqrt{2}l^* \sum_{i=1}^{n} \frac{1}{\sqrt{i}}. \tag{1}$$

As $\frac{1}{\sqrt{x}}$ is a decreasing function in $x \in \mathbb{R}$, we have $\frac{1}{\sqrt{i}} \leq \int_{i-1}^{i} \frac{1}{\sqrt{x}}\,\mathrm{d}x$ for any positive integer i. Hence, the right-hand side of (1) is at most $\sqrt{2}l^* \int_0^n \frac{1}{\sqrt{x}}\,\mathrm{d}x$, which equals $2\sqrt{2n}l^*$. □

We note that, to get better performance guarantees, it sometimes helps to use an objective function that differs from the one defined in the optimization problem. An example of this is given by Angel et al. [1]. They refer to a result from [9] who show that the flip neighborhood function has a performance bound of $\frac{3}{2}$ for the non-weighted Max k-SAT problem. Khanna et al. [12] prove that this is the best you can achieve using any normal local search implementation. However, they also show that when using to a different objective function, the flip neighborhood function can achieve a performance ratio of $\frac{4}{3}$ for this problem.

Non-approximability Results

Above, we gave examples of performance ratio analyses for a particular neighborhood function. In this section, we show that we can also prove results on the performance ratio for a complete class of neighborhood functions for a problem.

Suppose that finding an R-approximate solution for Π is NP-hard for a given $R \geq 1$. Provided that P $\neq$ NP, this means that iterative improvement using neighborhood function N does not find a solution with performance ratio R in polynomial time. As it may take iterative improvement an exponential number of steps to reach a local optimum, this does not directly imply that N has a performance ratio larger than R. The following theorem from Yannakakis [24] states that the claim is nevertheless true. Observe that, by taking $R = 1$, the theorem yields that an NP-hard problem will not admit a polynomially searchable exact neighborhood function.

Theorem 5. *Let $\Pi \in$ NPO, and let N be a neighborhood function for Π that is polynomially searchable. If the problem of finding an R-approximate solution for Π is NP-hard for a given $R \geq 1$, then N does not have a performance bound of R, provided that NP $\neq$ co-NP.*

Proof. We prove the result by contradiction. Suppose that approximating Π within a factor R is NP-hard and that N has a performance bound of R. We show that for any decision problem $\Pi_D \in$ NP the complementary problem Π_D^c obtained from Π_D by reversing the answers is also in NP. This yields a contradiction as it implies NP = co-NP.

To prove $\Pi_D^c \in$ NP, we need to show that each yes-instance of Π_D^c has a certificate that can be checked in polynomial time for validity, where the size of the certificate is polynomially bounded. As approximating Π within a factor R is NP-hard, a polynomial-time algorithm $\mathcal{A}$ exists that decides Π_D, where $\mathcal{A}$ may use an oracle that returns an R-approximate solution for any problem instance of Π. Hence, if for a problem instance I of Π_D we are given a sequence consisting of solutions that may be returned successively by the oracle during the execution of $\mathcal{A}$ on I, then we can determine in polynomial time whether I is a yes-instance or a no-instance of Π_D. This implies that as a certificate for yes-instances of Π_D^c we can use a sequence consisting of solutions that are locally optimal for the problem instances that are successively given to the oracle, i.e., the ith solution in the sequence is a local optimum for the ith problem instance given to the oracle. The certificate, which is of polynomial size, can be checked for validity in polynomial time by substituting each call of the oracle in $\mathcal{A}$ by a polynomial-time procedure that checks whether the corresponding solution in the certificate is a local optimum for the problem instance given to the oracle. By assumption this polynomial-time procedure exists. □

From the above theorem and the complexity result of Sahni and Gonzalez [22], it follows that if NP $\neq$ co-NP, then the performance ratio of the k-change neighborhood function cannot be bounded by $2^{p(n)}$ for some fixed polynomial p. Note that this result is weaker than the result of Theorem 3. Theorem 3 states that the performance ratio cannot even be bounded by 2^{2^n}, for instance, and that it also holds in the unlikely case that NP = co-NP.

From Neighborhood Function to Polynomial-Time Algorithm

Rewriting Theorem 5 gives that if a polynomially searchable neighborhood function N with performance bound R exists for some given combinatorial optimization problem $\Pi \in$ NPO, then the problem of finding an R-approximate solution for Π is not NP-hard, provided that NP $\neq$ co-NP. One question that arises is whether this result can be strengthened to the extent that the existence of such an N implies that the problem of finding an R-approximate solution for Π is easy, i.e., polynomially solvable. In this section we prove for a special type of combinatorial optimization problem that the answer is affirmative for $R = 1$ and that it is affirmative up to some arbitrary small error $\varepsilon > 0$ for $R > 1$. In other words, we show for a special type of combinatorial optimization problem how we can derive from a polynomially searchable exact neighborhood function a polynomial-time algorithm that solves the problem and how we can derive from a polynomially searchable neighborhood function with performance bound R a polynomial-time

approximation algorithm with a performance bound of $R + \varepsilon$ for any precision $\varepsilon > 0$. Besides being polynomial in the input size, the $(R + \varepsilon)$-approximation algorithm is also polynomial in $\frac{1}{\varepsilon}$. Before proving these results, we introduce some terminology used in this section. The results are due to Schulz, Weismantel, and Ziegler [23] and Orlin, Punnen, and Schultz [18], respectively.

For many combinatorial optimization problems, it holds that a solution can be viewed as a subset of a ground set E and that the cost of a solution $s \subseteq E$ can be written as $\sum_{e \in s} c(e)$, where $c(e)$ is the cost of element $e \in E$. This is, for instance, the case for TSP: define E as the set of all pairs (i, j) of cities and $c(i, j)$ as the distance d_{ij} from city i to city j. We call such problems linear combinatorial optimization problems.

Definition 4. An *instance* of a linear combinatorial optimization problem is a pair (S, c), where each solution in the solution space $S \subseteq 2^E$ is a subset of a finite ground set $E = \{1, 2, \ldots, n\}$ and where $c : E \to \mathbb{Q}^+$ assigns a cost to each element $e \in E$. The cost of a solution $s \in S$ is defined by $f(s) = \sum_{e \in s} c(e)$. □

Definition 5. A *linear combinatorial optimization problem* Π is specified by a set of problem instances as defined in Definition 4, and the goal is to minimize for a problem instance the associated cost function f. Hence, for a problem instance (S, c), we have to find a solution s^* that satisfies $f(s^*) \leq f(s)$ for all $s \in S$. □

Definition 6. A linear combinatorial optimization problem Π is called *closed under scaling* whenever for each problem instance (S, c) of Π it holds that if we change function $c : E \to \mathbb{Q}^+$ to some other cost function $c' : E \to \mathbb{Q}^+$, then the problem instance remains in Π. □

In this section the focus is on linear combinatorial optimization problems Π that satisfy the following two properties.

- Π is closed under scaling.
- For each problem instance of Π, we can derive in polynomial time a feasible solution.

It can easily be verified that the linear combinatorial optimization problem TSP (the generic formulation) satisfies these two properties. However, this is not true, for instance, for Metric TSP because this problem is not closed under scaling.

If a problem is closed under scaling, then this need not necessarily mean that we can change the cost function of a problem instance in polynomial time. The reason for this is that an encoding scheme may be applied that does not give $c(e)$ explicitly for each $e \in E$. To avoid this problem, we focus on linear combinatorial optimization problems that are encoded by a cost-explicit encoding scheme, where a cost-explicit encoding scheme is defined as follows.

Definition 7. An encoding scheme of a linear combinatorial optimization problem is called *cost explicit* if the encoding $e_I \in \{0,1\}^*$ of a problem instance $I = (S, c)$ with underlying ground set E is given by $e_I = e_I^{(1)} e_I^{(2)}$, where

- $e_I^{(1)}$ is an explicit encoding of the function $c : E \to \mathbb{Q}^+$, which means that it contains the encoding of $c(e)$ for each $e \in E$, and
- $e_I^{(2)}$ is an encoding, either implicit or explicit, of the solution space S.

Furthermore, $e_I^{(1)}$ only depends on S via its ground set E, and $e_I^{(2)}$ is independent of c. □

Exact Neighborhood Functions

Let Π be a linear combinatorial optimization problem from NPO that satisfies the two properties given above, and let a cost-explicit encoding scheme be used for Π. Furthermore, let N be an exact polynomially searchable neighborhood function for the problem. We will now present a polynomial-time algorithm, called $\mathcal{A}_{\text{OPT}}(N)$, that solves Π.

Let $I = (S, c)$ be a problem instance of Π, where, without loss of generality, we assume that c is a function to the natural numbers instead of the rational numbers. $\mathcal{A}_{\text{OPT}}(N)$ solves I in $K = \lceil \log_2(c_{\max} + 1) \rceil$ phases, where $c_{\max}$ is the maximum cost $c(e)$ of any element $e \in E$. In phase k an optimal solution is derived for the problem instance $I_k = (S, c_k)$ that is obtained from I by defining the cost $c_k(e)$ of an element $e \in E$ as the number represented by the k leading bits in the binary representation of $c(e)$. By adding leading zeros, the binary representation is made equally long for all $e \in E$, i.e., each binary representation consists of exactly K bits. We implement a phase by applying iterative improvement with the exact neighborhood function N. In the first phase, an arbitrary solution may be chosen as the starting solution, and in phase k with $k \geq 2$, the solution derived in phase $k-1$ is chosen as the starting solution. Figure 4 gives the algorithm in pseudo-code, where f_k denotes the cost function related to problem instance I_k.

Theorem 6. *Let $\Pi \in$ NPO be a linear combinatorial optimization problem that is closed under scaling and for which we can derive a feasible solution in polynomial time. Furthermore, let N be a polynomially searchable exact neighborhood function for Π. Then the algorithm $\mathcal{A}_{\text{OPT}}(N)$ given in Fig. 4 solves Π in polynomial time in the case that we use a cost-explicit encoding scheme for Π.*

Proof. By construction, the number of phases is bounded by $K = \lceil \log_2(c_{\max} + 1) \rceil$. Furthermore, because N is polynomially searchable, the individual iterations of the iterative improvement algorithm that makes up a single phase of $\mathcal{A}_{\text{OPT}}(N)$ can be implemented to run in polynomial time. Hence, to prove the theorem, we only have to show that in an arbitrary phase k the iterative improvement algorithm executed terminates within a polynomial number of iterations.

algorithm $\mathcal{A}_{\mathrm{OPT}}(N)$

begin

$s :=$ some initial solution;

$K := \lceil \log_2(c_{\max} + 1) \rceil$;

for $k := 1$ **to** K **do**

begin

for all $e \in E$ **do** $c_k(e) := \lfloor c(e) \cdot 2^{k-K} \rfloor$;

repeat

generate an $s' \in N(s)$;

if $f_k(s') < f_k(s)$ **then** $s := s'$;

until $f_k(s') \geq f_k(s)$ for all $s' \in N(s)$;

end

end;

Fig. 4 Algorithm $\mathcal{A}_{\mathrm{OPT}}(N)$ for a linear combinatorial optimization problem Π and an accompanying neighborhood function N. Π is closed under scaling, and (S, c) denotes an arbitrary problem instance of Π that is given as input to the algorithm. Cost function f_k relates to problem instance (S, c_k)

Let s_i be the solution derived in phase i of $\mathcal{A}_{\mathrm{OPT}}(N)$, and let s_0 be the starting solution of the first phase. Hence, in phase k, iterative improvement starts with s_{k-1} and ends with s_k. Because the cost of the solution derived by iterative improvement decreases in each iteration, it suffices to prove that $f_k(s_{k-1}) - f_k(s_k)$ is polynomially bounded to show that in phase k iterative improvement terminates within a polynomial number of iterations. By defining $c^{(i)}(e)$ with $e \in E$ as the ith bit in the binary representation of $c(e)$, we obtain that for any solution s the cost $f_k(s)$ satisfies

$$f_k(s) = \begin{cases} \sum_{e \in s} c^{(1)}(e) & \text{if} \quad k = 1 \\ 2 f_{k-1}(s) + \sum_{e \in s} c^{(k)}(e) & \text{if} \quad k \geq 2. \end{cases}$$

Furthermore, we have $\sum_{e \in s} c^{(k)}(e) \leq n$, where n denotes the number of elements in E. Using these two observations and the optimality of s_{k-1} with respect to cost function f_{k-1}, we can derive for $k \geq 2$

$$f_k(s_{k-1}) - f_k(s_k) = 2\,(f_{k-1}(s_{k-1}) - f_{k-1}(s_k)) + \sum_{e \in s_{k-1}} c^{(k)}(e) - \sum_{e \in s_k} c^{(k)}(e) \leq n$$

and for $k = 1$

$$f_k(s_{k-1}) - f_k(s_k) = \sum_{e \in s_{k-1}} c^{(k)}(e) - \sum_{e \in s_k} c^{(k)}(e) \leq n.$$

This proves the theorem. □

Neighborhood Functions with Performance Bound

Theorem 6 yields that if a polynomially searchable neighborhood function with performance bound $R=1$ exists for a given linear combinatorial optimization problem Π that satisfies some properties, then the problem of finding an R-approximate solution for Π is polynomially solvable. We now show that for $R > 1$ this result holds up to some arbitrary small error $\varepsilon > 0$. More precisely, we proceed as follows.

Let Π be a linear combinatorial optimization problem from NPO that satisfies the same two properties as those given in Theorem 6 and for which we use a cost-explicit encoding scheme. Furthermore, let N be a polynomially searchable neighborhood function for Π with a performance bound of R. We now present an algorithm for which we will prove that for any precision $\varepsilon > 0$ it (i) has a running time that is polynomially bounded in both the input size and $\frac{1}{\varepsilon}$ and (ii) has a performance bound of $R + \varepsilon$.

Let $I = (S, c)$ be a problem instance of Π, and suppose that we perform iterative improvement on I, where we take an arbitrary $s \in S$ as a starting solution. By assumption, s can be derived in polynomial time. We turn iterative improvement into a polynomial-time algorithm in the following way: we replace the function c in I by c', such that in each iteration the cost of the solution derived by iterative improvement decreases by at least q and $f'(s)/q$ is polynomially bounded, where $f'(s)$ is the cost of solution s with respect to the new function $c' : E \to \mathbb{Q}^+$. The former property is achieved by defining for any $e \in E$

$$c'(e) = \left\lceil \frac{c(e)}{q} \right\rceil q.$$

The latter property is achieved by defining q as

$$q = \frac{f(s)\varepsilon}{2nR(R+\varepsilon)}$$

since we then have

$$\frac{f'(s)}{q} \leq \frac{f(s) + nq}{q} = 2nR\left(\frac{R}{\varepsilon} + 1\right) + n,$$

which is polynomially bounded in $|I|$ and $\frac{1}{\varepsilon}$.

Using the polynomial-time algorithm described, we can now construct a polynomial-time $(R+\varepsilon)$-approximation algorithm $\mathcal{A}_\varepsilon(N)$ for Π. Algorithm $\mathcal{A}_\varepsilon(N)$ proceeds in phases. In each phase the algorithm described is executed, where the

final solution of the previous phase is taken as a starting solution. If the algorithm arrives at a solution for which the cost is at most half the cost of the solution with which the phase started, then the algorithm is interrupted, and the next phase is initiated. If the algorithm terminates, i.e., if it arrives at a local optimum, then the overall algorithm $\mathcal{A}_\varepsilon(N)$ also terminates. In Fig. 5 algorithm $\mathcal{A}_\varepsilon(N)$ is given in pseudo-code.

Lemma 3. *Let $\Pi \in NPO$ be a linear combinatorial optimization problem that is closed under scaling and for which we can derive a feasible solution in polynomial time. Furthermore, let N be a polynomially searchable neighborhood function for Π. Then the algorithm $\mathcal{A}_\varepsilon(N)$ given in Fig. 5 runs in polynomial time in the case that we use a cost-explicit encoding scheme for Π.*

Proof. The number of phases executed by $\mathcal{A}_\varepsilon(N)$ is bounded by $\log(nc_{\max})$, where $c_{\max}$ is the maximum cost $c(e)$ of any element $e \in E$. As a cost-explicit encoding scheme is assumed, this implies that the number of executed phases is polynomially bounded. Combined with the above-derived polynomial running time of each individual phase, this proves the polynomial-time running time of $\mathcal{A}_\varepsilon(N)$. □

The next lemma focuses on the quality of the solutions returned by $\mathcal{A}_\varepsilon(N)$.

Lemma 4. *Let Π be a linear combinatorial optimization problem that is closed under scaling, and let N be a neighborhood function for Π with performance bound R. Then the algorithm $\mathcal{A}_\varepsilon(N)$ given in Fig. 5 is an $(R+\varepsilon)$-approximation algorithm.*

Proof. Let s_{init}, q, c', and f' denote the corresponding values at the moment $\mathcal{A}_\varepsilon(N)$ terminates, let s be the solution returned by the algorithm, and let s^* be an optimal solution to the problem instance $I = (S, c)$, which is the problem instance that is given as input to the algorithm.

According to the definition of function c', we have $f(s) \leq f'(s)$. Furthermore, as s is locally optimal for problem instance $I' = (S, c')$ (but not necessarily for I) and as the neighborhood function has a performance bound of R, by assumption, we have $f'(s) \leq R \cdot f'(s^*)$. Using this, we can derive

$$f(s) \leq R \cdot f'(s^*) = R \sum_{e \in s^*} \left\lceil \frac{c(e)}{q} \right\rceil q \leq R \sum_{e \in s^*} (c(e) + q) \leq R \cdot f(s^*) + Rnq. \tag{2}$$

Moreover, $f(s_{\text{init}})/2 < f(s)$ holds because otherwise the last phase of the algorithm would have been interrupted just before setting the variable $stop$ to true. From this and the definition of q, it now follows that $q < f(s)\frac{\varepsilon}{nR(R+\varepsilon)}$. Combined with (2) this gives $f(s) \leq R \cdot f(s^*) + f(s)\frac{\varepsilon}{R+\varepsilon}$, which can be rewritten as $f(s) \leq (R + \varepsilon) f(s^*)$. □

algorithm $\mathcal{A}_\varepsilon(N)$

begin

$s :=$ some initial solution;

$stop :=$ FALSE;

while $\neg stop$ **do**

begin

$s_{\text{init}} := s$;

$q := f(s_{\text{init}})\varepsilon/(2nR(R+\varepsilon))$;

for all $e \in E$ **do** $c'(e) := \left\lceil \frac{c(e)}{q} \right\rceil q$;

repeat

if $f'(s') \geq f'(s)$ for all $s' \in N(s)$ **then** $stop :=$ TRUE;

else

begin

generate an $s' \in N(s)$;

if $f'(s') < f'(s)$ **then** $s := s'$;

end

until $stop =$ TRUE or $f(s) \leq f(s_{\text{init}})/2$;

end

end;

Fig. 5 Algorithm $\mathcal{A}_\varepsilon(N)$ for a linear combinatorial optimization problem Π that is closed under scaling, an accompanying neighborhood function N, and an arbitrary precision $\varepsilon > 0$. (S, c) denotes an arbitrary problem instance of Π that is given as input to the algorithm. Cost function f relates to problem instance (S, c), while cost function f' relates to problem instance (S, c')

Theorem 7. *Let $\Pi \in$ NPO be a linear combinatorial optimization problem that is closed under scaling and for which we can derive a feasible solution in polynomial time. Furthermore, let N be a polynomially searchable neighborhood function for Π with performance bound R. Then the algorithm $\mathcal{A}_\varepsilon(N)$ given in Fig. 5 is a*

polynomial-time $(R + \varepsilon)$*-approximation algorithm in the case that we use a cost-explicit encoding scheme for* Π.

Proof. The theorem follows directly from Lemmas 3 and 4. □

Time Complexity

Since optimal algorithms for NP-hard problems have exponential running time, we presented local search as an approach that trades off quality against performance. More precisely, it tries to reduce execution time by not solving the real optimization problem, but an associated local search problem, as defined below.

Definition 8. An *instance* of a local search problem is a triple (S, f, N), where (S, f) is an instance of a combinatorial optimization problem and $N : S \rightarrow 2^S$ is a neighborhood function on S. □

Definition 9. A *local search problem* Π_{LS} is specified by a set of problem instances, and it is either a minimization or a maximization problem. The problem is to find for a given instance (S, f, N) a locally optimal solution $\hat{s} \in S$. For a minimization problem, this means that $f(\hat{s}) \leq f(s)$ for all $s \in N(s)$, and for a maximization problem, it means that $f(\hat{s}) \geq f(s)$ for all $s \in N(s)$. □

For the time complexity of a local search algorithm, we can both be interested in the average case time complexity or in a guaranteed, i.e., worst-case time complexity. In this section we focus on the latter. First we study the computational complexity of local search problems. This means that we try to find an answer to the question of how hard the problem is that a local search algorithm tries to solve. This section is based on the paper of Yannakakis [24], who gives an excellent presentation of the complexity theory for local search problems introduced by Johnson, Papadimitriou, and Yannakakis [11]. Second, we look at the time complexity of the basic local search algorithm: iterative improvement.

Computational Complexity of Local Search Problems

To distinguish easy *decision* problems from hard ones, the theory of NP-completeness has been developed. In a nutshell, the idea of this theory is as follows. A class NP of decision problems is defined that contains the decision variants of many apparently hard and interesting combinatorial optimization problems. To relate the complexity of two problems in NP, polynomial-time reductions are introduced. If a problem Π is polynomially reducible to problem Π', then Π' is at least as hard as Π, which means that a polynomial-time algorithm for Π' can only exist if it exists for Π. The problems in NP to which all problems in this class are polynomially reducible are called NP-complete. These problems are the hardest

problems in NP since they only allow an efficient polynomial-time algorithm if such an algorithm exists for all problems in NP. This is very unlikely as no polynomial-time algorithm has been found for an NP-complete problem despite the enormous amount of effort that has been spent by many experts on finding one.

A similar strategy is used to identify hard local search problems for which it is very unlikely that they can be solved by a polynomial-time algorithm. First of all, we define a complexity class, called PLS, which contains local search problems that are specified by a combinatorial optimization problem from NPO and a "reasonable" corresponding neighborhood function. Next, we present a reduction to express that a local search problem is at least as hard as another given local search problem. Using this reduction, we can identify the hardest problems in PLS. Because no polynomial-time algorithm is known for any of these PLS-complete problems, as they are called, it is commonly believed that no such algorithm exists for these problems.

We now further elaborate on this theory. Let Π be a combinatorial optimization problem from NPO, and let N be a neighborhood function for it. A successful application of local search for this problem depends on whether it is easy to generate an initial solution and to check whether a solution is locally optimal and, if not, to derive a better neighbor. In terms of iterative improvement, this means that both the initialization and each single iteration of the algorithm can be implemented to run in polynomial time. The number of iterations, however, may be exponential. The class PLS contains all local search problems in which the properties described are satisfied.

Definition 10. Let Π_{LS} be a local search problem, and let Π be the underlying combinatorial optimization problem. Local search problem Π_{LS} is in the class *PLS (Polynomial-time Local Search)* if $\Pi \in$ NPO and if two polynomial-time algorithms A and B exist that satisfy the following properties.

- For a problem instance (S, f, N) of Π_{LS}, algorithm A returns a solution $s \in S$.
- For a problem instance (S, f, N) of Π_{LS} and a solution $s \in S$, algorithm B decides whether s is a local optimum and, if this is not the case, it returns a neighboring solution with better cost.

□

The following definition specifies a PLS-reduction by which we can relate the complexity of two local search problems. From the definition it follows directly that if local search problem Π_{LS} is PLS-reducible to local search problem Π'_{LS}, then the existence of a polynomial-time algorithm for Π'_{LS} implies the existence of such an algorithm for Π_{LS}. In other words, Π'_{LS} is at least as hard as Π_{LS}. Furthermore, it can be verified that PLS-reductions are transitive, which means that if Π_{LS} is PLS-reducible to Π'_{LS} and Π'_{LS} is PLS-reducible to Π''_{LS}, then Π_{LS} is also PLS-reducible to Π''_{LS}.

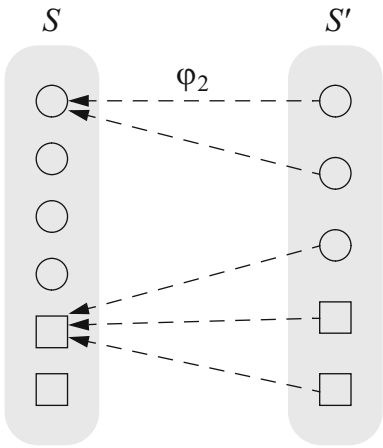


Fig. 6 Example of algorithm φ_2 for mapping solutions from S' to solutions from S. The squares indicate the local optima

Definition 11. Local search problem Π_{LS} is *PLS-reducible* to local search problem Π'_{LS}, denoted by $\Pi_{\text{LS}} \propto \Pi'_{\text{LS}}$, if two polynomial-time algorithms φ_1 and φ_2 exist that satisfy the following properties; see also Fig. 6.

- Algorithm φ_1 transforms a problem instance I of Π_{LS} into a problem instance $\varphi_1(I)$ of Π'_{LS}.
- Algorithm φ_2 maps a problem instance $I = (S, f, N)$ of Π_{LS} and a solution $s' \in S'$ with $\varphi_1(I) = (S', f', N')$ to a solution $s \in S$.
- For a problem instance I of Π_{LS}, we have that if $s' \in S'$ is a local optimum for $\varphi_1(I) = (S', f', N')$, then $\varphi_2(I, s')$ is a local optimum for I.

The pair (φ_1, φ_2) is called a *PLS-reduction*. □

We note that the definition of φ_2 can be weakened. For φ_2 it suffices that instead of all solutions in S', only the local optima are mapped to a solution in S. However, if we have $\Pi_{\text{LS}} \in$ PLS, then we can easily transform such a φ_2 into an algorithm that satisfies Definition 11 by mapping all solutions that are not in the domain of the restricted φ_2 to the solution returned by the polynomial-time algorithm A defined in Definition 10. Analogously to NP-completeness, we define the notion of PLS-completeness to indicate the hardest problems in PLS.

Definition 12. A local search problem in PLS is *PLS-complete* if each problem $\Pi_{\text{LS}} \in$ PLS is PLS-reducible to it. □

Let NPCO contain all NP-hard problems from NPO. When determining the complexity of solving some combinatorial optimization problem from NPO to optimality, one generally aims to decide whether the problem is in PO or in NPCO. It is therefore interesting to relate the class PLS and the subclass of PLS-complete

problems to the classes PO, NPO, and NPCO. This is the focus of the remainder of this section.

First we show that PLS $\subseteq$ NPO. Let Π_{LS} be an arbitrary local search problem in PLS. Problem Π_{LS} can be formulated as a minimization problem $\Pi \in$ NPO by defining an instance $(S, f') \in \Pi$ for any instance $(S, f, N) \in \Pi_{\mathrm{LS}}$, such that cost function $f' : S \to \{0, 1\}$ satisfies $f'(s) = 0$ if s is locally optimal and $f'(s) = 1$ otherwise. The existence of algorithm B as defined in Definition 10 for Π_{LS} implies that f' is computable in polynomial time. From this it follows that Π is indeed a problem from NPO. Moreover, Π_{LS} and Π are clearly equivalent. As a result, we get PLS $\subseteq$ NPO.

We now show that PO $\subseteq$ PLS. Let $\Pi \in$ PO and let $\mathcal{A}$ be an optimal polynomial-time algorithm for Π. Without loss of generality, we assume that Π is a minimization problem. Then we define minimization problem $\Pi' \in$ PO such that any instance $I = (S, f)$ of Π corresponds to an instance (S, f') of Π', where $f'(s) = 0$ if solution s is optimal with respect to cost function f and $f'(s) = 1$ otherwise. If we define a neighborhood function for Π' in such a way that each solution has only one neighbor, namely, the solution returned by $\mathcal{A}$, then we obtain a local search problem that is equivalent to Π. Furthermore, the problem is in PLS. Algorithm A in Definition 10 can be defined as $\mathcal{A}$, and algorithm B in this definition can be realized by first comparing the cost of the solution given by $\mathcal{A}$ with the cost of a given solution s and by then returning the optimal solution derived by $\mathcal{A}$ if the latter cost is larger. Hence, we now obtain the following result.

Theorem 8. *The complexity classes PO, PLS, and NPO satisfy*

$$PO \subseteq PLS \subseteq NPO.$$

□

From the relation PLS $\subseteq$ NPO, it follows that a PLS-complete problem is not harder than any NP-hard problem in NPO, as will now be shown. Suppose that a polynomial-time algorithm exists for an NP-hard problem in NPCO. This means that all problems in NPO are polynomially solvable, including those in PLS. To substantiate the intractability of PLS-complete problems, a more interesting question is whether PLS-complete problems are also at least as hard as the problems in NPCO. If the answer were to be affirmative, then the existence of a polynomial-time algorithm for a PLS-complete problem would be as unlikely as the existence of such an algorithm for a problem in NPCO. However, this is not the case because the following theorem states that if NP $\neq$ co-NP, then a PLS-complete problem is not NP-hard. Hence, in the unlikely case that PLS-complete problems turn out to be polynomially solvable, this still does not imply that the problems in NPCO are also solvable in polynomial time. Figure 7 depicts the relations between the different complexity classes.

Theorem 9. *If NP $\neq$ co-NP, then none of the problems in PLS are NP-hard.*

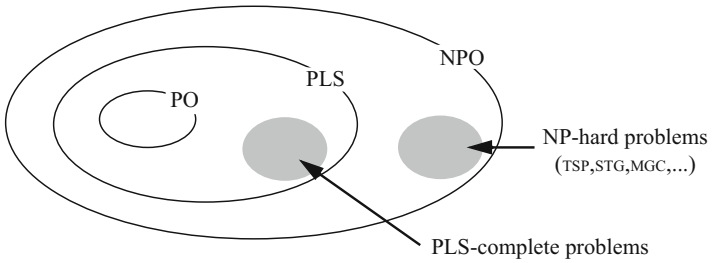


Fig. 7 Positioning of the classes PO, PLS, and NPO in the case that P $\neq$ NP, NP $\neq$ co-NP, and PO $\neq$ PLS

Proof. The result can be proved in a similar way as Theorem 5. Note that Theorem 5 easily follows from this theorem. Suppose that a problem $\Pi_{\text{LS}} \in$ PLS is NP-hard. We show that this contradicts NP $\neq$ co-NP by showing that it implies that, for any decision problem Π_D in NP, the complementary problem Π_D^c obtained by reversing the answers in Π_D is also in NP.

As Π_{LS} is NP-hard, a polynomial-time algorithm $\mathcal{A}$ exists that decides Π_D, where $\mathcal{A}$ may use an oracle that, for any problem instance of Π_{LS}, returns a local optimum. Hence, if for a problem instance I of Π_D we are given a sequence consisting of local optima that may be returned successively by the oracle during the execution of $\mathcal{A}$ on I, then we can determine in polynomial time whether I is a yes-instance or a no-instance of Π_D. As a result, we can use this sequence of locally optimal solutions as a certificate for yes-instances of Π_D^c. Furthermore, the certificate, which is of polynomial size, can be checked in polynomial time for validity by substituting each call of the oracle in $\mathcal{A}$ by a polynomial-time procedure that checks whether the corresponding solution in the certificate is a solution to the problem instance of Π_{LS} given to the oracle. The existence of polynomial-time algorithm B in Definition 10 shows that this polynomial-time procedure exists. This proves $\Pi_D^c \in$ NP, which completes the proof of the theorem. □

Proving PLS-Completeness

As for the theory of NP-completeness, the theory of PLS-completeness only becomes useful once we have a first problem for which we can prove that it is among the hardest of the class PLS. After that, we can prove PLS-completeness of a problem by simply proving that a known PLS-complete problem is PLS-reducible to it. The first problem for which PLS-completeness has been proved is circuit/flip as defined below.

Suppose we want to apply local search to the problem of finding an integer s with $0 \leq s < 2^n$ for which a given function $f : \{0, 1, \ldots, 2^n - 1\} \rightarrow \{0, 1, \ldots, 2^m - 1\}$ is

minimized. We can use the neighborhood function in which $s' \in \{0, 1, \ldots, 2^n - 1\}$ is a neighbor of s if s' can be obtained from s by flipping exactly one of the n bits in the binary representation of s. The corresponding local search problem, formulated in terms of Boolean circuits, is the first problem that has been proved to be PLS-complete.

Boolean circuits are theoretical counterparts of the digital circuits from which computers are made. They compute Boolean functions $f : \{0,1\}^n \to \{0,1\}^m$, and, conversely, each Boolean function is computed by a circuit.

Definition 13. A *Boolean circuit* is a directed acyclic graph $D = (V, A)$. The node set V consists of n input nodes and $|V| - n$ gates. The input nodes have indegree zero and are labeled by the binary variables $x_1, x_2, \ldots, x_n$. The labels of the gates are taken from the set $\{\wedge, \vee, \neg\}$ of Boolean functions. The gates with outdegree zero are called the output nodes, and they are additionally labeled by the binary variables $y_1, y_2, \ldots, y_m$. Boolean circuit D computes a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ by deriving for given values of the input variables $x_1, x_2, \ldots, x_n$ corresponding values for the output variables $y_1, y_2, \ldots, y_m$ in the following way.

Let l be the label of a gate g. If l is given by $\neg$, then the value of g is one minus the value of the node from which the only incoming edge of g is incident. Next, suppose that $l \in \{\wedge, \vee\}$. Then g has exactly two incoming edges. Let o_1, o_2 be the values of the nodes from which these two edges are incident. If g is labeled $\wedge$, then it is assigned the value 1 if $o_1 = o_2 = 1$, and it is assigned the value 0, otherwise. If g is labeled $\vee$, then it is assigned the value 0 if $o_1 = o_2 = 0$, and it is assigned the value 1 otherwise. The *size* of Boolean circuit $D = (V, A)$ is given by the number of nodes in V. □

By interpreting the input and output vectors as two numbers written in binary notation, i.e., by interpreting the input as $\sum_{i=1}^{n} 2^{i-1} x_i$ and the output as $\sum_{i=1}^{m} 2^{i-1} y_i$, a Boolean circuit can compute an integer function. Hence, the local search problem mentioned above can be formalized by the following definition. We note that the Hamming distance of two strings s and s' of equal size, denoted by $H(s, s')$, is given by the number of positions in which s and s' differ.

Definition 14 (Circuit/flip). Given is a Boolean circuit D with n input nodes $x_1, x_2, \ldots, x_n$ and m output nodes $y_1, y_2, \ldots, y_m$. The neighborhood function, called flip, is based on the solution space $\{0,1\}^n$ containing all possible input vectors $(x_1, x_2, \ldots, x_n)$ of the circuit. A solution s is a neighbor of solution s' if s and s' have a Hamming distance $H(s, s')$ of one. Find a solution $\hat{s}$ that is locally optimal with respect to the cost function

$$f(\hat{s}) = \sum_{i=1}^{m} 2^{i-1} y_i,$$

which gives the integer interpretation of the output of Boolean circuit D on input $\hat{s}$. In min-circuit/flip, it is our goal to minimize the cost function, while in max-circuit/flip it is our goal to maximize the cost function. □

Theorem 10. *Min-circuit/flip and max-circuit/flip are both PLS-complete.* □

Although very interesting, the proof of the above theorem is rather complex and therefore left out of this chapter. For details, we refer to Yannakakis [24], Johnson, Papadimitriou, and Yannakakis [11] or Michiels, Aarts, and Korst [16]. To focus on how the theory can be used, we now give an example of how it can be used to show that the uniform graph partitioning problem with the Kernighan-Lin neighborhood function is PLS-complete.

Definition 15 (Uniform Graph Partitioning (UGP)). Given is an edge-weighted graph $G = (V, E)$ with $|V| = 2n$ and a weight $w(e) \in \mathbb{N}$ for each edge $e \in E$. Find a partition of V into two subsets V_1 and V_2 with $|V_1| = |V_2| = n$, such that the sum of the weights of the edges that have one endpoint in V_1 and one endpoint in V_2 is minimal, i.e., such that

$$c(V_1, V_2) = \sum_{e \in E \cap (V_1 \times V_2)} w(e)$$

is minimal. □

An obvious neighborhood function for UGP is the swap neighborhood function in which the neighbors of a partition (V_1, V_2) are obtained by interchanging a node from V_1 with a node from V_2. Variable-depth search gives an effective approach for making the swap neighborhood function more powerful. The main idea is that we allow multiple swaps to be performed simultaneously. We call the resulting neighborhood function Kernighan-Lin, after its two inventors. For each i with $1 \leq i \leq n$, the Kernighan-Lin neighborhood of a partition (V_1, V_2) contains exactly one partition that is obtained from (V_1, V_2) by exchanging i nodes from V_1 with i nodes from V_2.

The n Kernighan-Lin neighbors of a partition (V_1, V_2), written as $(V_1^{(i)}, V_2^{(i)})$ for $1 \leq i \leq n$, can be determined in n steps as follows. In Step i we derive partition $(V_1^{(i)}, V_2^{(i)})$ by performing the best possible swap on (V_1, V_2) if $i = 1$ and on the partition $(V_1^{(i-1)}, V_2^{(i-1)})$ derived in the previous step if $2 \leq i \leq n$. We are not allowed to select a node that has already been selected before for a swap. By the best possible swap, we mean the swap of the nodes from $V_1^{(i-1)}$ and $V_2^{(i-1)}$ that produces the maximum decrease in cost or, if such a swap does not exist, a minimum increase in cost. In the case of a tie, a tie-breaking rule is used to choose a unique pair of nodes that is swapped. The neighborhood consists of n partitions because, after n swaps, all nodes of the graph have been selected for an exchange and, by definition, each node may only be selected once. In other words, the neighbors are obtained by

performing iterative improvement using the best-improvement pivoting rule and the swap neighborhood function. However, unlike the standard iterative improvement algorithm, we have the additional constraint that each node may only be selected once for a swap and that we accept a best cost-increasing neighbor if we are in a local optimum.

We prove that UGP with the Kernighan-Lin neighborhood function is PLS-complete by a reduction from another local search problem. Before defining this problem, we introduce some terminology. Let U be a set of binary variables. A truth assignment for U assigns to each variable in U either the value 0 (false) or 1 (true). A variable $u \in U$ is called a positive literal, and its negation $\bar{u} = 1 - u$ is called a negative literal. A clause is a set of literals and/or constants, where a constant has the value either 0 or 1.

Definition 16 (Positive Not-All-Equal Max-3Sat (POS NAE MAX-3SAT)). Given is a set U of binary variables, a set C of clauses over U, and a positive integer weight $w(c)$ for each clause $c \in C$. A clause does not contain negative literals, and the number of positive literals plus the number of constants in each clause is at most three. A clause is satisfied by some truth assignment if and only if it contains at least one literal/constant with value 1 and at least one literal/constant with value 0. The problem is to find a truth assignment that maximizes the sum of the weights of clauses that are satisfied. □

A possible neighborhood function for POS NAE MAX-3SAT is the flip neighborhood function in which truth assignment t' is a neighbor of truth assignment t if t' can be obtained from t by changing (flipping) the value of one variable. In much the same way as the Kernighan-Lin neighborhood function for UGP is obtained by applying variable-depth search to the swap neighborhood function, the Kernighan-Lin neighborhood function for POS NAE MAX-3SAT is obtained by applying variable-depth search to the flip neighborhood function. This means that a truth assignment t' is a neighbor of truth assignment t with respect to the Kernighan-Lin neighborhood function if it can be derived from t by performing a sequence of flips, where in each step the most profitable (least unprofitable) flip is chosen from the flips that change the value of a variable that has not been flipped before in this sequence. An arbitrary rule may be used for breaking ties. We now state the following result without proof. For a proof, we again refer to Yannakakis [24], Johnson, Papadimitriou, and Yannakakis [11] or Michiels, Aarts, and Korst [16].

Theorem 11. *POS NAE MAX-3SAT with the Kernighan-Lin neighborhood function is PLS-complete.* □

Knowing that POS NAE MAX-3SAT/Kernighan-Lin is PLS-complete, we are now able to prove that UGP/Kernighan-Lin is PLS-complete.

Theorem 12. *UGP with the Kernighan-Lin neighborhood function is PLS-complete.*

Proof. Let the cut of a partition (V_1, V_2) be defined as the set of edges that have one endpoint in V_1 and one in V_2.

UGP is equivalent to its maximization variant in which we are asked to partition the nodes of a graph into two sets V_1, V_2 of equal size, such that the weight $c(V_1, V_2)$ of the resulting cut is maximized instead of minimized. This can be seen as follows. Let $G = (V, E)$ be a weighted graph with maximum weight $W = \max_{e \in E} w(e)$, and let K be the complete graph on the same node set V, in which the weight of an edge e is given by $W - w(e)$ if $e \in E$ and by W otherwise. The cost of a partition (V_1, V_2) in one variant of the problem is then equivalent to $n^2 W$ minus the cost of this partition in the other variant. This implies that both problems are indeed equivalent.

When we inspect the given relation between the two variants, it follows easily that the corresponding local search problems under the Kernighan-Lin neighborhood function are also equivalent. As a result, we can prove the theorem by giving a PLS-reduction (φ_1, φ_2) from POS NAE MAX-3SAT/Kernighan-Lin to the maximization version of UGP with the Kernighan-Lin neighborhood function, where we use Theorem 11, which states that the former problem is PLS-complete.

Definition PLS-reduction. Let I be an arbitrary problem instance of POS NAE MAX-3SAT/Kernighan-Lin, and let $x_1, x_2, \ldots, x_m$ be the binary variables of I. The problem instance $\varphi_1(I)$ is defined by the following graph $G = (V, E)$. Node set V contains two nodes x_i and x_i' for each variable x_i in I. The nodes x_i and x_i' represent the value of the variable x_i and its negation, respectively. To V we also add the nodes y and z, which represent the constant 1, and the nodes y' and z', which represent the constant 0. This concludes the definition of V.

The edge set E is partitioned into the two subsets E_1 and E_2. Set E_1 contains all edges that connect an unprimed node x_i with its primed variant. Furthermore, it contains an edge from each of the two 1-nodes y and z to each of the two 0-nodes y' and z'. All these $m + 4$ edges are assigned the same large weight of $W + 1$, where W is defined as the total weight of all clauses in I. The following two conditions are necessary and sufficient to guarantee that a cut contains all edges from E_1.

- Nodes y and z that represent the constant 1 are both assigned to the same subset, and nodes y' and z' that represent the constant 0 are both assigned to the other subset.
- The nodes x_i and x_i' are assigned to different subsets for all $1 \leq i \leq m$.

If these two conditions are satisfied by a partition, we say that the partition is feasible; see Fig. 8 for an example. We can assume the following one-to-one correspondence between a feasible partition and a truth assignment without violating the interpretation of the nodes x_i and x_i' as being the value of variable x_i and its negation. A variable x_i or its negation is assigned the value 1 if and only if the corresponding node is contained in the same subset as the variables y, z that represent the constant 1. This means that if a variable or its negation is assigned the value 0, then the corresponding node is contained in the same subset as the two

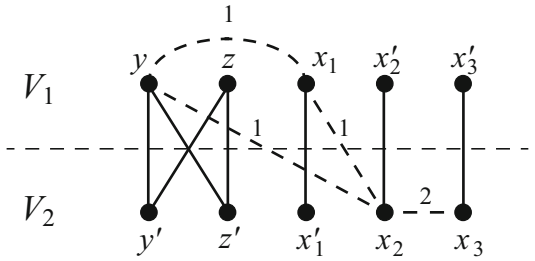


Fig. 8 Feasible partition $P_t = (V_1, V_2)$ for the graph $\varphi_1(I)$, where I is the problem instance of POS NAE MAX-3SAT/Kernighan-Lin consisting of the two clauses $\{x_1, x_2, 1\}$ and $\{x_2, x_3\}$. Both clauses have a weight of two. The undashed edges are from E_1 and the dashed edges are from E_2. Each edge from E_1 has weight $W + 1 = 5$, and the weights of the edges from E_2 are denoted in the figure. Feasible partition P_t corresponds to truth assignment t with $x_1 = 1$ and $x_2 = x_3 = 0$. This truth assignment satisfies the clause $\{x_1, x_2, 1\}$ but not the clause $\{x_2, x_3\}$

variables y' and z' that represent the constant 0. We define P_t as the feasible partition corresponding to truth assignment t and R as the set of all feasible partitions, i.e., $R = \{P_t \mid t \text{ truth assignment}\}$.

To be able to derive the cost of a truth assignment t from the feasible partition P_t, we add additional edges to E via E_2. For each clause $c = \{a, b\}$, we add the edge $\{a, b\}$ to E_2 with the same weight as clause c, where for the constants 1 and 0 we use the nodes y and y', respectively. If such an edge contributes its weight to the cost of a feasible partition P_t, then a and b are assigned to different subsets, which imply that the clause $\{a, b\}$ is satisfied by truth assignment t.

For a clause $c = \{a, b, c\}$ of weight w, we include the three edges $\{a, b\}$, $\{b, c\}$, and $\{a, c\}$ in E_2, each with weight $w/2$. It can be verified that the cut of a potentially infeasible partition contains either two out of the three edges or none of the three edges. Moreover, for a feasible partition P_t, the former case implies that clause c is satisfied by t, and the latter case implies that c is not satisfied by t. As a result, the edges contribute w to the cost of a feasible partition P_t if c is satisfied by t and they do not contribute otherwise. We note that if a pair of nodes a, b occurs in several clauses, we assign to edge $\{a, b\}$ the weight obtained by summing up the weights that arise from all these clauses. This concludes the definition of G and therefore of problem instance $\varphi_1(I)$. To complete the definition of our PLS-reduction, we define algorithm φ_2 in such a way that it returns for a potentially infeasible partition P the truth assignment that assigns to a variable x_i the value 1 if and only if it is contained in the same subset as variable y. Note that this algorithm complies with the given relation between a truth assignment t and the feasible partition P_t.

Proof strategy. As the total weight W of the edges from E_2 that can be in the cut of a partition is strictly smaller than the weight $W + 1$ of a single edge in E_1 and as the cut of a feasible partition contains all edges from E_1, we can make the following observation.

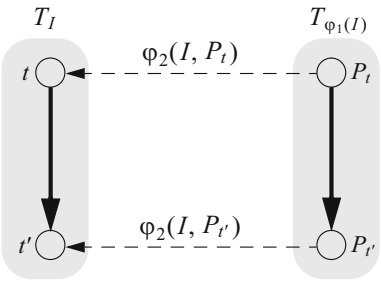


Fig. 9 Correspondence between arc in transition graph T_I and arc in transition graph $T_{\varphi_1(I)}$

Observation. The minimum cost over all feasible partitions is strictly larger than the maximum cost over all infeasible partitions.
This implies that, for any feasible partition P_t, the transition graph $T_{\varphi_1(I)}$ of $\varphi_1(I)$ only contains outgoing arcs to other feasible solutions. Consequently, to prove that (φ_1, φ_2) correctly defines a PLS-reduction, it suffices to prove that the transition graph satisfies the following two conditions.

- *Condition 1.* $T_{\varphi_1(I)}$ contains an arc $(P_t, P_{t'})$ if and only if T_I contains the arc (t, t'). See Fig. 9.
- *Condition 2.* For any partition $P \notin R$, transition graph $T_{\varphi_1(I)}$ contains a path to a feasible partition.

In the proof we assume that, to determine the Kernighan-Lin neighborhood of a partition, an arbitrary rule is used for breaking ties in the case that several pairs of nodes qualify for being selected for a swap. Given this rule, we may choose the tie-breaking rule used to determine the Kernighan-Lin neighborhood of a truth assignment as we like because the PLS-completeness result for POS NAE MAX-3SAT/Kernighan-Lin holds regardless of this rule.

Proof of Condition 1. As the cost of a feasible partition P_t is given by $|E_1|(W + 1)$ plus the cost of truth assignment t, the first condition holds if we can prove that for any truth assignment t, partition P_t and t have the same neighborhood. More precisely, it suffices to show that $P_{t'}$ is a neighbor of P_t for $t' \neq t$ if and only if t' is a neighbor of t. Until all pairs of nodes x_i, x_i' have been swapped and thus blocked, we only swap pairs x_i, x_i' of nodes when deriving the neighbors of P_t. This is true since swapping such a pair in a feasible partition results again in a high-quality, feasible partition, while swapping any other pair of nodes results in a low-quality, infeasible partition. After all pairs of nodes x_i, x_i' have been swapped, the nodes y, z and y', z' are swapped in two steps, where the intermediate partition is infeasible. This brings us back to partition P_t.

Swapping x_i and x_i' has the same effect on the cost of a feasible partition as flipping the value of variable x_i on the cost of the corresponding truth assignment. As a result, we obtain that, if the same rule is used for breaking ties in the derivation of the Kernighan-Lin neighborhood of t as is used in the derivation of

the Kernighan-Lin neighborhood of P_t, then the feasible neighbors of P_t, excluding P_t, correspond to the truth assignments in the neighborhood of t. This proves the first claimed condition given above.

Proof of Condition 2. To prove the second condition, suppose that it does not hold, which means that an infeasible partition $\hat{P}$ exists that is locally optimal. We show that this gives rise to a contradiction. As $\hat{P} = (V_1, V_2)$ is infeasible, either V_1 contains a pair u, u' of corresponding nodes or the two 1-nodes y, z are split. First of all, consider the former case. Obviously, V_2 also contains a pair v, v' of corresponding nodes as $|V_1| = |V_2|$. Swapping u and v leads to a weight gain of at least $W + 1$ via the edges from E_1, and it leads to a weight reduction of at most W via the edges from E_2. This contradicts the assumption that $\hat{P}$ is locally optimal. Next, assume that the two 1-nodes are split but that V_1 does not contain a pair u, u' of corresponding nodes. The two 0-nodes then also have to be in different subsets. Swapping a 0-node from one subset with a 1-node in the other subset yields a cost improvement of $2(W + 1)$. Again, this contradicts the local optimality of $\hat{P}$. □

It follows from the definition of PLS that an improving solution can be found in polynomial time if it exists. This is true for k-Opt, where an improving tour can be found in $\mathcal{O}(n^k)$ time. However, this is only practical for very small k. An interesting question is whether we can do better. This is studied by Marx [15] and Guo et al. [8]. They prove for a number of TSP neighborhood functions that the answer to this question is negative. However, the complexity may be exponential in k.

Time Complexity of Iterative Improvement

While the focus of the previous section was on the computational complexity of local search problems, we now study the time complexity of the iterative improvement algorithm when it is used to solve such a problem. A local search problem can be formulated as finding a node in a transition graph with outdegree zero. Iterative improvement is the obvious algorithm for tackling this problem. It traverses a transition graph, which is directed and acyclic, until it reaches a node without outgoing arcs. The precise path that the algorithm follows is determined by the pivoting rule. This rule specifies the neighboring solution that is selected if a solution has more than one neighbor with better cost. In terms of the transition graph, the pivoting rule prescribes which arc is to be selected in a given node if it has multiple outgoing arcs. Ideally, we would like to use a pivoting rule that takes the shortest path to a local optimum. However, computing this rule may be NP-hard, as, for instance, is the case for TSP with the 2-change neighborhood function [6].

As most of the neighborhood functions used in practice induce relatively small neighborhoods, which is reflected in our definition of the class PLS, it is generally easy to derive efficient implementations of commonly used (heuristic) pivoting rules as first improvement and best improvement. Hence, with regard to the time complexity of iterative improvement, we are particularly interested in the number of iterations required by iterative improvement. Obviously, the maximum number of

iterations required by iterative improvement is bounded from above by the longest path that occurs in the transition graph. This upper bound holds regardless of the pivoting rule used and the solution with which the algorithm starts. Whether or not this bound is tight depends on the pivoting rule applied and on the algorithm we use for generating the initial solution. With regard to the starting solution of iterative improvement, we generally do not want to make any assumptions. A reason for this is that because the quality of a starting solution generally has a major effect on the quality of the final solution, we want to have the opportunity to tune the algorithm that derives a starting solution to the application we are considering. Another reason is that, in order to make multiple runs of iterative improvement successful, we like to initialize each run with a solution that is quite different from the starting solutions used in the other runs.

Suppose that we apply iterative improvement to solve a problem Π_{LS} in PLS and let c be the cost of the solution with which we initialize the algorithm. Because the cost function is assumed to be integral and because in each iteration of iterative improvement the cost of the solution improves, the number of iterations the algorithm needs for reaching a local optimum is bounded by $|c - f^*|$, where f^* is the cost of an optimal solution. From this, it follows that if in the combinatorial optimization problem underlying Π_{LS} the cost of a solution is polynomially bounded in the input size, then iterative improvement can be implemented to run in polynomial time for Π_{LS}.

Although not satisfying this strong constraint, the cost functions of many combinatorial optimization problems satisfy the weaker property that the cost of a solution is polynomially bounded in the input size and the magnitude of the largest number that occurs in the problem instance. In these cases iterative improvement has a pseudo-polynomial running time. Examples include the already discussed problems TSP and UGP. For TSP the cost of a solution is, for instance, bounded by n times the longest distance in the distance matrix. In the combinatorial optimization problem underlying circuit/flip, the cost function does not even satisfy this weaker property.

In the next two subsections, we describe two different strategies for analyzing the time complexity of iterative improvement. In the first subsection, we give an example of the strategy where we analyze a specific implementation of iterative improvement. That is, the analysis assumes a certain pivoting rule. In the next subsection, we take a more generic strategy where we further exploit the theory of PLS-completeness to derive negative results that hold regardless of the pivoting rule used.

Ad Hoc Strategies

In the literature, a number of results have appeared on the maximum number of iterations that are required by an implementation of iterative improvement. Proving a negative result, i.e., that the number of iterations can be exponential, is easiest if we assume the pivoting rule that selects an improving neighbor randomly. In

that case, we only need to show that there exists a path in the transition graph that has exponential length. An example of this is given in the result below from Luecker [14]. We note, however, that the result has been improved by Englert, Röglin and Vöcking [4] who prove that also for Euclidean TSP, the 2-change neighborhood function may result in an exponential number of iterations.

Theorem 13. *For Metric TSP with the 2-change neighborhood function, iterative improvement may require an exponential number of iterations to reach a local optimum if the pivoting rule is used that selects an improving move randomly.*

Proof. Consider a problem instance I of Symmetric TSP instead of Metric TSP. By adding to each entry d_{ij} of the distance matrix d the maximum distance $d_{\max}$ occurring in d, we obtain a distance matrix that satisfies the triangle inequality. This modification does not affect a run of iterative improvement as for any tour it results in a length increase of exactly $n \cdot d_{\max}$. Hence, to prove the theorem, it suffices to show that it holds for Symmetric TSP.

We now proceed as follows. For any even n, we construct a problem instance I of Symmetric TSP consisting of n cities. For this problem we then derive a sequence of exponentially many tours that can be visited successively by the iterative improvement algorithm. Note that because iterative improvement runs in pseudo-polynomial time for Symmetric TSP, the distance matrix of I will have to contain distances of exponential length.

Because we are considering Symmetric TSP, we have $d_{ij} = d_{ji}$. Therefore, it suffices to define the distance d_{ij} for the case where $i \geq 2$ and either $j > i$ or $j = 1$. The definition of the distance matrix d is visualized in Fig. 10. Formally, for even $i \geq 2$, the distance d_{ij} is given by 2^{2i+3} if j is odd and either $j = 1$ or $j > i$, and d_{ij} is given by $d_{ij} = 2^{2i+4}$ if j is even and $j > i$. For odd $i \geq 2$, we apply the

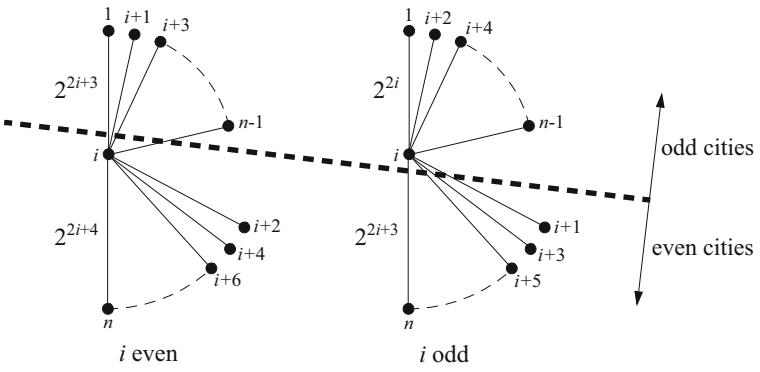


Fig. 10 Definition of the distance matrix d in the problem instance of Symmetric TSP introduced in the proof of Theorem 13, where $i \geq 2$ is an arbitrary city that is either even (left) or odd (right)

same case distinction: d_{ij} is given by 2^{2i} if j is odd and either $j = 1$ or $j > i$, and it is given by 2^{2i+3} if j is even and $j > i$.

We prove that iterative improvement can take an exponential number of iterations to transform the tour $(1, 2, \ldots, n)$ into the tour $(1, n-1, n-2, \ldots, 2, n)$ for the problem instance described. We prove this result by showing by induction on q that the following assertion holds. Let tour τ contain a subtour $x, 2, 3, \ldots, 2q+1, y$ for a $q \leq \frac{n-2}{2}$, where x and y are cities, possibly 1 and $2q+2$, such that for all $2 \leq i \leq 2q+1$ we have $d_{x,i} = d_{1,i}$ and $d_{i,y} = d_{i,2q+2}$. Then a sequence of at least 2^{q-1} improving 2-changes exists that transforms tour τ into τ_q, where tour τ_q is equivalent to τ, except that the subtour $x, 2, 3, \ldots, 2q+1, y$ is replaced by the subtour $x, 2q+1, 2q, \ldots, 2, y$, in which the order of the $2q$ middle cities has been reversed.

We first prove the basis case $q = 1$. Tour $\tau_q = \tau_1$ that contains $x, 3, 2, y$ is obtained from tour τ that contains $x, 2, 3, y$ by the single 2-change that replaces the edges $\{x, 2\}$ and $\{3, y\}$ by $\{x, 3\}$ and $\{2, y\}$. As $2^{q-1} = 1$, the basis case now follows if τ_1 is shorter than τ. This is true because for the cost improvement $\Delta(\tau, \tau_1)$ of τ_1 in comparison with τ, we can derive

$$\begin{aligned}\Delta(\tau, \tau_1) &= d_{1,2} + d_{3,4} - d_{1,3} - d_{2,4} \\ &= 2^7 + 2^9 - 2^6 - 2^8 \\ &> 0.\end{aligned}$$

Next, consider the case $q \geq 2$. We show how τ can be transformed into τ_q in at least 2^{q-1} improving 2-changes, where we distinguish five phases in the transformation. The final tour in phase i is denoted by $\tau^{(i)}$. Hence, $\tau^{(5)} = \tau_q$.

In the first phase, $\tau^{(1)} = \tau_{q-1}$ is derived from τ. By the induction hypothesis, this can be done in 2^{q-2} improving 2-changes. In Phase 2 we next change the order of the cities $2q$ and $2q+1$ in subtour $x, 2q-1, 2q-2, \ldots, 2, 2q, 2q+1, y$ of $\tau^{(1)}$ by performing a 2-change on the edges $\{2, 2q\}$ and $\{2q+1, y\}$. This results in a tour $\tau^{(2)}$ with shorter length than $\tau^{(1)}$ since

$$\begin{aligned}\Delta(\tau^{(1)}, \tau^{(2)}) &= d_{2,2q} + d_{2q+1,2q+2} - d_{2,2q+1} - d_{2q,2q+2} \\ &= 2^8 + 2^{4q+5} - 2^7 - 2^{4q+4} \\ &> 0.\end{aligned}$$

In Phase 3 the first and last edges $\{x, 2q-1\}$ and $\{2q, y\}$ occurring in subtour $x, 2q-1, 2q-2 \ldots, 2, 2q+1, 2q, y$ of $\tau^{(2)}$ are replaced by the two edges $\{x, 2q\}$ and $\{2q-1, y\}$. This yields a cost improvement of

$$\begin{aligned}\Delta(\tau^{(2)}, \tau^{(3)}) &= d_{1,2q-1} + d_{2q,2q+2} - d_{1,2q} - d_{2q-1,2q+2} \\ &= 2^{4q-2} + 2^{4q+4} - 2^{4q+3} - 2^{4q+1} \\ &> 0.\end{aligned}$$

The resulting tour $\tau^{(3)}$ contains $x, 2q, 2q+1, 2, 3, \ldots, 2q-1, y$. By the definition of the distance matrix, we have $d_{2q+1,i} = d_{1,i}$ and $d_{i,2q+2} = d_{i,2q}$ for all $2 \leq i \leq 2q$. As a result, we can apply the induction hypothesis to the subtour $x', 2, 3, \ldots, 2q-1, y'$ of $\tau^{(3)}$ with $x' = 2q+1$ and $y' = y$. This yields that $\tau^{(3)}$ can be transformed into $\tau^{(4)}$ by at least 2^{q-2} improving 2-changes, where $\tau^{(4)}$ is given by $\tau^{(3)}$ with the order of the cities $2, 3, \ldots, 2q-1$ reversed, i.e., $\tau^{(4)}$ contains $x, 2q, 2q+1, 2q-1, 2q-2, \ldots, 2, y$. In the final phase, we reverse the order of the cities $2q$ and $2q+1$ by performing a 2-change on $\{x, 2q\}$ and $\{2q+1, 2q-1\}$. This again results in an improvement of the tour as we have

$$\begin{aligned}\Delta(\tau^{(4)}, \tau^{(5)}) &= d_{1,2q} + d_{2q-1,2q+1} - d_{1,2q+1} - d_{2q-1,2q} \\ &= 2^{4q+3} + 2^{4q-2} - 2^{4q+2} - 2^{4q+1} \\ &> 0.\end{aligned}$$

In conclusion, we have transformed τ into $\tau^{(5)} = \tau_q$ by only performing improving 2-changes. The total number of 2-changes is at least 2^{q-2} in Phase 1, one in Phases 2 and 3, at least 2^{q-2} in Phase 4, and one in Phase 5. This makes a total of more than 2^{q-1} improving 2-changes. This proves the induction hypothesis and consequently it proves the theorem. □

The above result does not match the subquadratic number of iterations that one observes in practice [10]. To better understand the average case behavior, Chandra, Karloff, and Tovey [2] analyzed the expected length of the longest path in the transition graph in case the n cities are placed uniformly at random in the unit square. They proved an expected length of $\mathcal{O}(n^{10}\log(n))$.

General Strategy

Suppose that, either empirically or via an analysis as given in the above subsection, we find out that iterative improvement takes an exponential number of steps. A question that arises is whether this bad worst-case behavior can be avoided by choosing a less naive pivoting rule or whether it is unavoidable due to the structure of the transition graph. This means that we are interested in the question of whether iterative improvement would still run in worst-case exponential time if for the pivoting rule we were to have an oracle at our disposal that gives us a neighbor that leads to a nearest local optimum. This problem, called the transition graph complexity problem, is formalized as follows.

Definition 17. Let T be a transition graph and let $\hat{V}$ be the nodes with outdegree zero (they represent the local optima). The potential of a node v is the minimum distance in T between v and a node in $\hat{V}$. The potential of T is the maximum potential over all its nodes.

Definition 18. For a given local search problem Π_{LS}, the *transition graph complexity problem* corresponds to deciding whether Π_{LS} can induce transition graphs with an exponentially large potential. □

In this section we study the transition graph complexity problem for PLS-complete problems. For PLS-complete problems, we already know that, unless PO = PLS, they cannot be solved in polynomial time. Solving the transition graph complexity problem gives us additional information on whether the unavoidable exponential running time is caused by the structure of the transition graph or is only caused by the intractability of computing an optimal pivoting rule.

The reason we introduced local search problems is that they are the problems that are actually solved by iterative improvement. Solving a local search problem is generally not a goal in itself. Therefore, considerably less attention has been paid by scientists to solving these problems efficiently than to finding efficient algorithms for NP-hard combinatorial optimization problems and NP-complete decision problems. As a result, we have stronger evidence for the inequality P $\neq$ NP, which is equivalent to PO $\neq$ NPO, than for PO $\neq$ PLS. This brings us to the second relevant implication of knowing that the potential of a transition graph can be exponentially large for a given PLS-complete problem. It proves an exponential worst-case running time for iterative improvement even in the case that PO = PLS. We nevertheless emphasize that it is commonly believed that PO $\neq$ PLS holds.

To solve the transition graph complexity problem, we proceed as follows. We defined PLS-reductions such that if $\Pi_{LS} \propto \Pi'_{LS}$ holds and if Π_{LS} cannot be solved in polynomial time, then Π'_{LS} cannot be solved in polynomial time either. Below we refine the definition of a PLS-reduction such that, in addition to this behavior, we have that if $\Pi_{LS} \propto \Pi'_{LS}$ holds and if a transition graph for Π_{LS} can have an exponentially large potential, then a transition graph of Π'_{LS} can also have an exponentially large potential. This more powerful PLS-reduction is called a tight PLS-reduction. We show that the class PLS contains a local search problem for which the transition graph can have an exponentially large potential. Furthermore, most known PLS-reductions, including the ones for circuit/flip and Theorem 11, are tight [11, 16, 24]. We now elaborate a bit more on the details.

Let (φ_1, φ_2) define a PLS-reduction between two local search problems Π_{LS} and Π'_{LS}, and let $I = (S, f, N)$ be an arbitrary problem instance of Π_{LS} with image $\varphi_1(I) = (S', f', N')$. By definition, algorithm φ_2 maps any solution in S' to a solution in S, such that a local optimum in S' is also a local optimum in S. Figure 11a depicts possible transition graphs T_I and $T_{\varphi_1(I)}$ of I and $\varphi_1(I)$, respectively, and a possible choice for φ_2. For (φ_1, φ_2) to be a tight PLS-reduction, a subset R of S' has to exist that satisfies the following three conditions. In the first place, φ_2 has to be surjective for I when restricting the solution set S' of $\varphi_1(I)$ to R. This means that for each solution $s \in S$, a solution $s' \in R$ must exist that is mapped to s by algorithm φ_2. Furthermore, solution s' must be computable in polynomial time. Note that for (φ_1, φ_2) to be a standard PLS-reduction it is not necessary that φ_2 is surjective for a given problem instance, as follows from Fig. 6. The second condition on R is that it contains all local optima of $\varphi_1(I)$. Figure 11b,

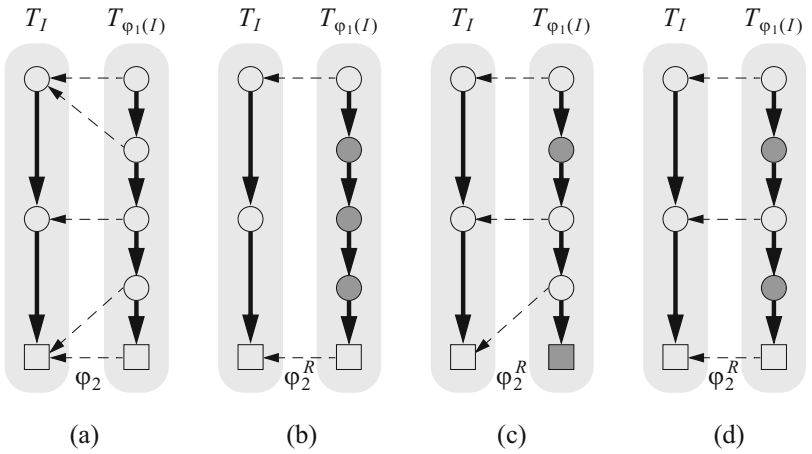


Fig. 11 (**a**) Transition graphs T_I of $I = (S, f, N)$ and $T_{\varphi_1(I)}$ of $\varphi_1(I) = (S', f', N')$ and algorithm φ_2 for some local search problem instance I and PLS-reduction (φ_1, φ_2). A dashed arc originates from an $s' \in S'$ and points to $s = \varphi_2(I, s')$. (**b**)–(**c**) Invalid choices of R, where R is given by S' minus the shaded solutions in transition graph $T_{\varphi_1(I)}$. φ_2^R denotes algorithm φ_2 for I with the solution set S' of $\varphi_1(I)$ restricted to R. (**d**) Valid choice of R

c give examples of R that do not satisfy the first and second condition, respectively, and Fig. 11d gives an example of R that satisfies both conditions. The third and last condition that has to be satisfied is that if in $T_{\varphi_1(I)}$ a solution $s'_2 \in R$ can be reached from a solution $s'_1 \in R$ without visiting any other solution from R, then the distance between the corresponding solutions in S is not too large, i.e., it is polynomially bounded.

Definition 19. Local search problem Π_{LS} is *tightly PLS-reducible* to local search problem Π'_{LS}, denoted by $\Pi_{\text{LS}} \propto_{\text{tight}} \Pi'_{\text{LS}}$, if a PLS-reduction (φ_1, φ_2) exists, such that for a polynomial p the following holds. For any problem instance $I = (S, f, N)$ of Π_{LS} with image instance $\varphi_1(I) = (S', f', N')$ of Π'_{LS}, we can choose a subset $R \subseteq S'$ that satisfies the following properties.

- For any $s \in S$ we can construct in polynomial time a solution $s' \in R$ with $\varphi_2(I, s') = s$.
- R contains all local optima in S'.
- If transition graph $T_{\varphi_1(I)}$ of $\varphi_1(I)$ contains a direct path from $s'_1 \in R$ to $s'_2 \in R$, then the distance from $s_1 = \varphi_2(I, s'_1)$ to $s_2 = \varphi_2(I, s'_2)$ in transition graph T_I of I is bounded from above by $p(|I|)$, where a path in $T_{\varphi_1(I)}$ is said to be direct if, except for the first and last solution, it only contains solutions outside R.

The pair (φ_1, φ_2) is called a *tight PLS-reduction*. □

Definition 20. A local search problem in PLS is *tightly PLS-complete* if each problem $\Pi_{\text{LS}} \in$ PLS is tightly PLS-reducible to it. □

This definition of a tight PLS-reduction preserves the property that a transition graph induced by a problem can be exponential, as shown in the following. By the last condition in Definition 19, we have that if the distance between any two solutions $s_1' \in R$ and $s_2' \in R$ in $T_{\varphi_1(I)}$ is $d_{s_1',s_2'}$, then $d_{s_1',s_2'} \cdot p(|I|)$ gives an upper bound on the distance between the corresponding solutions $s_1 = \varphi_2(I, s_1')$ and $s_2 = \varphi_2(I, s_2')$ in T_I. As R contains all local optima, this implies that if $s' \in R$ is within k steps of a local optimum $\hat{s}'$ in $T_{\varphi_1(I)}$, then solution $s = \varphi_2(I, s')$ is within $k \cdot p(|I|)$ steps of $\hat{s} = \varphi_2(I, \hat{s}')$, which by the definition of a PLS-reduction is a local optimum in S. Hence, by the second condition in Definition 19, the potential of T_I is at most $p(|I|)$ larger than the potential of $T_{\varphi_1(I)}$. Because the input size of $\varphi_1(I)$ is bounded from above by a polynomial in the input size of I, this implies that if T_I has an exponentially large potential, then $T_{\varphi_1(I)}$ must have one as well. We thus arrive at the following result.

Lemma 5. *If (φ_1, φ_2) defines a tight PLS-reduction from local search problem Π_{LS} to local search problem Π'_{LS} and if the potential of Π_{LS} cannot be bounded by a polynomial, then neither can the potential of Π'_{LS}.* □

By Lemma 5, it now suffices to indicate one local search problem in PLS for which the transition graph can have an exponentially large potential to conclude that this result holds for any tightly PLS-complete problem.

Lemma 6. *PLS contains a local search problem for which the potential of the transition graph cannot be bounded by a polynomial.*

Proof. Consider the trivial combinatorial optimization problem that, given an integer n, asks for the smallest integer i between 1 and n. This means that we want to minimize the cost function $f(i) = i$ with $1 \leq i \leq n$. For this problem, we consider the solution space consisting of the integers $1, 2, \ldots, n$, and we assume a neighborhood function in which integer 1 has no neighbors and integer $i \geq 2$ has only one neighbor, namely, $i - 1$. This yields a local search problem that is obviously in PLS. Furthermore, the transition graph is given by a single directed path that successively visits the solutions $n, n-1, \ldots, 1$. The potential of this graph equals the distance from n to 1, which is $n - 1$. This is exponentially large as the input size is only $\log n$. □

Theorem 14. *For a tightly PLS-complete problem, iterative improvement requires an exponential number of iterations in the worst case to reach a local optimum, regardless of the pivoting rule used.* □

It can be verified from the definition that the PLS-reduction given in Theorem 12 is tight. Furthermore, as shown in [11, 16, 24], the problem POS NAE MAX-3SAT/Kernighan-Lin from which the reduction is made is tightly PLS-complete. This brings us at the following result.

Theorem 15. *The local search problem UGP/Kernighan-Lin is tightly PLS-complete.* □

Suppose that for a combinatorial optimization problem Π we have two neighborhood functions N and N', where N' is a generalization of N, i.e., $N(s) \subseteq N'(s)$ for any solution s. For TSP, the 2-change and 3-change neighborhood functions are examples of N and N', respectively. Then the local search problem Π_{LS} specified by Π and N is PLS-complete if this is the case for the local search problem Π'_{LS} specified by Π and N'. More generally, this means that to prove PLS-completeness results for one combinatorial optimization problem and multiple neighborhood functions it suffices to prove that the weakest neighborhood function induces a PLS-complete problem. However, if we want to know whether the induced local search problems are *tightly* PLS-complete, we can no longer restrict ourselves to the weakest neighborhood function. It may be the case that Π_{LS} is tightly PLS-complete, while this is not the case for Π'_{LS}. The reason for this is that the generalization can add arcs to the transition graph, which may decrease its potential to a polynomially bounded value.

We refer you to Michiels, Aarts, and Korst [16] and to Monien, Dumrauf, and Tscheuschner [17] for a set of local search problems that have been proven to be (tightly) PLS-complete.

Relevance of PLS Beyond Local Search

Although introduced to enhance the understanding of applying local search algorithms to combinatorial optimization problems, the class PLS has also found its way to other disciplines. First of all, the theory of PLS-completeness is used in game theory to prove the hardness of computing pure Nash equilibria. An overview of such results until 2010 is given in [17]. Furthermore, PLS is an interesting class of problem in between P and NP-hard. We motivate this using the argumentation given by the excellent paper of Papadimitriou [19]. Many search problems have proven either to be NP-complete or to belong to P. The complexity of NP-hard problems typically results from the difficulty of deriving whether a solution exists. As a result of this, most of the problems that have not been categorized as being either NP-complete or in P belong to the class of total search problems. For problems in this class, a solution is known to exist. The most well-known example of this is factoring. It is known that any integer number can be factored into primes, but finding the factorization is considered to be difficult. This problem has not proven to be NP-complete, but as for NP-complete problems, many have tried to solve it in polynomial time but without success. Therefore, it is commonly assumed that this problem is also hard to solve. The security of the widely used cryptographic cipher RSA even relies on this assumption.

Different classes of total search problems are used to capture the possible different reasons why a total search problem can be hard to solve. PLS is such

a class. Others are Polynomial Parity Argument (PPA), Polynomial Pigeonhole Principle (PPP), and Polynomial Parity Argument for Directed graphs (PPAD). Interesting papers in this area are [5, 7, 19].

Conclusion

Local search owes its popularity mainly to its good average-case performance. Its worst-case performance is typically less good compared to alternative solution approaches, like constructive algorithms. Nevertheless, studying the worst-case performance is still valuable since it both provides performance guarantees and enhances our understanding of the algorithm. In this chapter we discussed several worst-case local search results that have been scattered over the literature. We presented results on the worst-case performance ratio of local search implementations and on the time complexity of reaching a locally optimal solution. The latter results also find their applications outside the scope of local search for combinatorial optimization problems, viz., game theory and general computational complexity theory.

References

1. Angel E, Christopoulos P, Zissimopoulos V (2014) Local search: complexity and approximation. In: Paschos VT (ed) Paradigms of combinatorial optimization: problems and new approaches, vol 2. Wiley, Hoboken, pp 435–471
2. Chandra B, Karloff H, Tovey C (1999) New results on the old k-opt algorithm for the traveling salesman problem. SIAM J Comput 28:1998–2029
3. Christofides N (1976) Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, 388. Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh
4. Englert M, Röglin H, Vöcking B (2007) Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. In: Proceedings of the 18^{th} ACM-SIAM symposium on discrete algorithm, New Orleans, pp 1295–1304
5. Fearnley J, Savani R (2015) The complexity of the simplex method. In: Proceedings of the forty-seventh annual ACM symposium on theory of computing, pp 201–208
6. Fischer S (1995) A note on the complexity of local search problems. Inf Process Lett 53:69–75
7. Gordon S (2017) The complexity of continuous local search. Master thesis, University of Illinois
8. Guo J, Hartung S, Niedermeier R, Suchý O (2013) The parameterized complexity of local search for TSP, more refined. Algorithmica 67:89–110
9. Hansen P, Jaumard B (1990) Algorithms for the maximum satisfiability problem. Computing 44:279–303
10. Johnson D, McGeoch L (1997) The traveling salesman problem: a case study. In: Aarts E, Lenstra J (eds) Local search in combinatorial optimization. Wiley, New York, pp 215–310
11. Johnson D, Papadimitriou C, Yannakakis M (1988) How easy is local search? J Comput Syst Sci 37:79–100
12. Khanna S, Motwani R, Sudan M, Vazirani U (1998) On syntactic versus computational views of approximability. SIAM J Comput 28:164–191

13. Levin A, Yovel U (2013) Nonoblivious 2-Opt heuristics for the traveling salesman problem. Networks 62(3):201–219
14. Luecker G (1976) unpublished manuscript. Department of Computer Science, Princeton University, Princeton
15. Marx D (2008) Searching the k-change neighborhood for TSP is w[1]-hard. Oper Res Lett 36:31–36
16. Michiels W, Aarts E, Korst J (2007) Theoretical aspects of local search. Springer, Berlin
17. Monien B, Dumrauf D, Tscheuschner T (2010) Local search: simple, successful, but sometimes sluggish. In: Automata, languages and programming, pp 1–17
18. Orlin J, Punnen A, Schulz A (2004) Approximate local search in combinatorial optimization. SIAM J Comput 33:1201–1214
19. Papadimitriou C (2014) Algorithms, complexity, and the sciences. Proc Natl Acad Sci 111:15881–15887
20. Papadimitriou C, Steiglitz K (1977) On the complexity of local search for the traveling salesman problem. SIAM J Comput 6:76–83
21. Papadimitriou C, Steiglitz K (1982) Combinatorial optimization. Prentice-Hall, Englewood Cliffs
22. Sahni S, Gonzalez T (1976) P-complete approximation problems. J Assoc Comput Mach 23:555–565
23. Schulz A, Weismantel R, Ziegler G (1995) 0/1-integer programming: optimization and augmentation are equivalent. In: Proceedings of the 3rd annual European symposium on algorithms, Corfu, pp 473–483
24. Yannakakis M (1997) Computational complexity. In: Aarts E, Lenstra J (eds) Local search in combinatorial optimization. Wiley, New York, pp 19–55

Variable Neighborhood Descent

12

Abraham Duarte, Nenad Mladenović, Jesús Sánchez-Oro, and Raca Todosijević

Contents

Introduction . . . 342
Neighborhoods . . . 344
Neighborhoods for Continuous Optimization Problems . . . 345
Neighborhoods for Binary Problems . . . 346
Neighborhoods for Integer Problems . . . 348
Neighborhoods for Permutation Problems . . . 350
Local Search Methods . . . 352
VND Variants . . . 355
Sequential Variable Neighborhood Descent Procedures . . . 356
Nested Variable Neighborhood Descent . . . 359
Mixed Variable Neighborhood Descent . . . 362
Conclusions . . . 365
Cross-References . . . 365
References . . . 366

A. Duarte (✉)
Department of Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: abraham.duarte@urjc.es

J. Sánchez-Oro
Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: jesus.sanchezoro@urjc.es

N. Mladenović
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

LAMIH, University of Valenciennes, Famars, France

LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia
e-mail: nenadmladenovic12@gmail.com; Nenad.Mladenovic@univ-valenciennes.fr

R. Todosijević
LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia
e-mail: racatodosijevic@gmail.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_9

Abstract

Local search heuristic that explores several neighborhood structures in a deterministic way is called variable neighborhood descent (VND). Its success is based on the simple fact that different neighborhood structures do not usually have the same local minimum. Thus, the local optima trap problem may be resolved by deterministic change of neighborhoods. VND may be seen as a local search routine and therefore could be used within other metaheuristics. In this chapter, we discuss typical problems that arise in developing VND heuristic: what neighborhood structures could be used, what would be their order, what rule of their change during the search would be used, etc. Comparative analysis of usual sequential VND variants is performed in solving traveling salesman problem.

Keywords

Variable neighborhood descent · Local search · Intensification · Deterministic exploration

Introduction

Optimization is a key discipline in fields such as computer science, artificial intelligence, and operations research. Outside these scientific communities, the meaning of optimization becomes quite vague, going to mean simply "do it as better as you can." In the context of this chapter, the concept of optimization is conceived as the process of trying to find the best possible solution to an optimization problem, usually in a limited time horizon.

In the simplest case, an optimization problem may be defined by a 2-tuple (X, f), where X represents the set of feasible solutions and f is an objective function that assigns a real number to each solution $x \in X$, which represents its quality or fitness. Then, the main objective of an optimization problem is to find a solution $x^\star \in X$ with the best objective function value among all solutions in the search space. Therefore, in a minimization problem, $x^\star \in X$ is a minimum point if $f(x^\star) \leq f(x)$, $\forall x \in X$. Notice that minimization of f is equivalent to maximization of $-f$.

In optimization problems, there is usually either finite but a huge number or infinity number of solutions and a clear criterion for the comparison among them. Some well-known examples of optimization problems are the traveling salesman problem (TSP), the vehicle routing problem (VRP), the quadratic assignment problems (QAP), or scheduling problems, among others. A detailed description of these problems can be found in [3, 36]. The difficulty of solving these problems has been studied since the late 1970s [12]. These studies concluded that there is a subset of problems where it is possible to design an algorithm, which presents a polynomial computational complexity, i.e., the execution time of these algorithms polynomially grows with the problem size. Such problems belong to the class $\mathcal{P}$, and they are considered "easy to solve." Examples of these problems are the shortest

path problem (Dijkstra algorithm), the minimum spanning tree (Prim or Kruskal algorithms), or flows in networks (Ford-Fulkerson algorithm). However, computing optimal solutions is intractable for many optimization problems of industrial and scientific importance (i.e., there is no known algorithm with polynomial complexity to solve it optimally). This type of problems belongs to a class known as $\mathcal{NP}$, and they are considered "hard to solve."

In practice, we are usually satisfied with "good" solutions, which are obtained by heuristic algorithms. In particular, metaheuristics (MHs) represent a family of approximate [42] optimization techniques that gained a lot of popularity in the past two decades, becoming the most promising and successful techniques for solving hard problems. Unlike exact optimization algorithms, metaheuristics do not guarantee the optimality of the obtained solutions. Additionally, metaheuristics do not define how close the obtained solutions are from the optimal ones, in contrast with approximation algorithms. MHs provide acceptable solutions in a reasonable computing time for solving hard and complex problems in science and engineering.

The term metaheuristic was coined in 1986 [13] as a way of defining *a master process that guides and modifies other subordinate heuristics to explore solutions beyond simple local optimality*. MHs are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial and numerical optimization problems arising in human activities, such as economics, industry, or engineering. MHs can be seen as general algorithmic frameworks that require relatively few modifications to be adapted to tackle a specific problem. They constitute a very diverse family of optimization algorithms including methods such as simulated annealing (SA), Tabu search (TS), genetic algorithms (GA), ant colony optimization (ACO), or variable neighborhood search (VNS).

Metaheuristics are high-level strategies for exploring the search space using different methods. The search strategies are highly dependent on the philosophy of the metaheuristic itself. In particular, *trajectory-based metaheuristics* can be seen as intelligent extensions of traditional local search methods. The goal of this kind of MH is to escape from a local optimum in order to proceed in the exploration of the search space and move on to find other hopefully better local optimum. Examples of these MHs are Tabu search [13], simulated annealing [23], or variable neighborhood search [15], among others. *Population-based metaheuristics* deal with a set of solutions instead of dealing with only one solution. These techniques proved a natural and intrinsic way for the exploration of the search space. The final performance of these methods strongly depends on how the population is managed. Examples of population-based metaheuristics are genetic algorithms [19], scatter search [14], or memetic algorithms [33], among others. Some authors consider a third kind of metaheuristics called *constructive-based metaheuristics*, where the main effort is put in the intelligent construction of the solution. In other words, instead of starting the search from a random solution, these methods try to construct a high-quality initial solution. Examples of constructive-based metaheuristics are GRASP [10], ant colony optimization [5], or iterated greedy [39], among others.

The number of new proposed metaheuristics has amazingly increased in the last 25 years. Nowadays, the portfolio of MHs contains more than 50 variants, only

considering the most stabilized ones (MHs successfully applied to a relatively large set of optimization problems). However, at the end, when designing a metaheuristic for an optimization problem, we face with two contradictory criteria: intensification (exploitation) and diversification (exploration). In fact, the performance of a metaheuristic basically relies on how it balances both criteria. The intensification of an algorithm describes its ability to thoroughly explore the promising regions in the hope to find better solutions. On the other hand, diversification describes the ability of the metaheuristic to explore non-visited regions of the search space in order to assure the evenly exploration of the search space and to avoid the confinement to the procedure to a reduced number of regions. Therefore, when tackling an optimization problem, it is necessary to search for the equilibrium between both criteria.

Variable neighborhood search (VNS) is a metaheuristic which was proposed in [32] as a general framework to solve hard optimization problems. This methodology is based on performing systematic changes of neighborhoods during the search space exploration. VNS has evolved in recent years, resulting in a large variety of strategies. Some of the most relevant variants are reduced VNS (RVNS), variable neighborhood descent (VND), basic VNS (BVNS), skewed VNS (SVNS), general VNS (GVNS), or variable neighborhood decomposition search (VNDS), among others (see [18] for a survey on VNS). We refer the reader to [6, 8, 37, 40, 41] to some recent and successful applications of VNS to hard optimization problems.

In this chapter, we focus on the deterministic variant of VNS, namely, variable neighborhood descent (VND). VND deserves separate attention since it is usually used in context of other metaheuristics as a local search routine. Once the set of neighborhood structures is selected, to be used in a deterministic manner, the VND-based local searches may be designed in three possible ways: (i) sequential VND, where neighborhoods are placed in the list with a given order and always explored in that order; (ii) nested or composite VND, where neighborhood operators are composed, i.e., $N_1(N_2(N_3(\ldots(x))))$ (neighborhood one of neighborhood two of neighborhood three, etc. of x; and (iii) mixed nested VND, where the two previous strategies are combined.

In this chapter, we first give some possible classification of neighborhood structures that are usually used in solving continuous and discrete optimization problems. Then we provide pseudocodes of sequential, composite, and mixed nested variants of VND. Variants of sequential VND are compared on traveling salesman problem.

Neighborhoods

The representation (encoding) of a solution in an optimization problem plays a relevant topic in the design of an algorithm. In fact, as it is well documented, this representation determines the difficulty of solving a problem [12] and also the complexity of some routines within algorithm. In addition, this encoding strongly influences the way in which the neighborhoods of a given solution are defined. Therefore, it is not possible to separate the neighborhood of a solution from its corresponding representation in the computer memory.

Let us assume without loss of generality that each solution $x \in X$ is represented by a vector $x = (x_1, \ldots, x_n)$, being n the size of the problem. Depending on the values of each x_i, we can distinguish among different types of problems: continuous ($x_i \in \mathbb{R}$), binary ($x_i \in \{0, 1\}$), integer ($x_i \in \mathbb{N}$), or permutations ($x_i \in \mathbb{N}$, $1 \leq x_i \leq n$ and $x_i \neq x_j$ if $i \neq j$). We could also identify a fifth class of problems which encompasses mixed variables. In this section, we describe the most common neighborhoods defined in the related literature as well as some basic properties.

Neighborhoods for Continuous Optimization Problems

The continuous constrained nonlinear global optimization problem (GOP) in general form is given as follows:

$$(GOP) \quad \begin{cases} \min & f(x) \\ \text{s.t.} & g_i(x) \leq 0 \qquad \forall i \in \{1, 2, \ldots, m\} \\ & h_i(x) = 0 \qquad \forall i \in \{1, 2, \ldots, r\} \\ & a_j \leq x_j \leq b_j \ \forall j \in \{1, 2, \ldots, n\} \end{cases}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, \ldots, m$, and $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, \ldots, r$ are possibly nonlinear continuous functions and $a, b \in \mathbb{R}^n$ are the variable bounds.

GOP naturally arises in many applications, e.g., in advanced engineering design, data analysis, financial planning, risk management, scientific modeling, etc. Most cases of practical interest are characterized by multiple local optima, and, therefore, in order to find the globally optimal solution, a global scope search effort is needed.

If the feasible set X is convex and objective function f is convex, then GOP is relatively easy to solve, i.e., the Karush-Kuhn-Tucker conditions may be applied. However, if X is not a convex set or f is not a convex function, we could have many local minima, and thus, the problem may not be solved by using classical techniques.

For solving GOP, neighborhood structures $\mathcal{N}_k(x)$ are usually induced from the ℓ_p metric:

$$\rho(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty) \tag{1}$$

or

$$\rho(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p \to \infty). \tag{2}$$

The neighborhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k–th neighborhood of x, and using the metric ρ, it is defined as:

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x, y) \leq \rho_k\}, \tag{3}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} \leq \rho(x, y) \leq \rho_k\}, \tag{4}$$

where ρ_k, known as the radius of $\mathcal{N}_k(x)$, is monotonically increasing with k.

Note that in some papers neighborhoods structures in $\mathbb{R}^n$ are not induced from the ℓ_p metric (see e.g., [1]).

Neighborhoods for Binary Problems

There are a large family of optimization problems, whose solution is usually represented as a binary array, where the presence or absence of an element is described by means of a binary variable. The knapsack problem [28], the max-cut problem (MCP) [29], or the maximum diversity problem [7] are examples of these problems. For solving binary problems, neighborhood structures $\mathcal{N}_k(x)$ are usually induced from the Hamming metric:

$$d_H(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{5}$$

More precisely, the k–th neighborhood of a solution x, i.e., $\mathcal{N}_k(x)$, relatively to Hamming metric, is defined as

$$\mathcal{N}_k^{Bin}(x) = \{y \in X \mid d_H(x, y) = k\} \tag{6}$$

We will use the max-cut problem to illustrate the most relevant characteristics of binary problems. Consider a graph $G = (V, E)$ with vertex set V and edge set E. Let w_{ij} be the weight associated with edge $(i, j) \in E$. A $cut(W, W')$ is a partition of V into two sets W, $W' = V \setminus W$, and its value is given by the expression:

$$cut(W, W') = \sum_{i \in W \wedge j \in W'} w_{ij}$$

The max-cut problem (MCP) consists of finding a cut in G with maximum value. Karp [22] showed that MCP is an $\mathcal{NP}$-hard problem. Figure 1a shows an example graph with five vertices and seven edges where the number close to each edge represents the corresponding weight. Figure 1b shows a possible solution, $x = (W, W')$, where $W = \{1, 2\}$ and $W' = \{3, 4, 5\}$. The value of this solution is $cut(W, W') = 9 + 14 + 10 + 5 = 38$, computed as the sum of the edges whose endpoints are in different sets (dashed edges). This solution can be represented as a binary vector $x = \{1, 1, 0, 0, 0\}$ where $x_i = 1$ indicates that the corresponding vertex is in W, while $x_i = 0$ means that the vertex is in W'.

For a given solution x of a MCP, we define $\mathcal{N}_{drop}^{Bin}(x)$ neighborhood using the $drop(x, i)$ move operator. This operator is responsible of changing the value of a variable x_i from 1 to 0, producing a new solution x'. The associated neighborhood, $\mathcal{N}_{drop}^{Bin}(x)$, has size n (in the worst case) and is a subset of $\mathcal{N}_1^{Bin}(x)$. Formally, $\mathcal{N}_{drop}^{Bin}(x)$ is defined as:

$$\mathcal{N}_{drop}^{Bin}(x) = \{x' \leftarrow drop(x, i) : x_i = 1 \wedge 1 \leq i \leq n\}$$

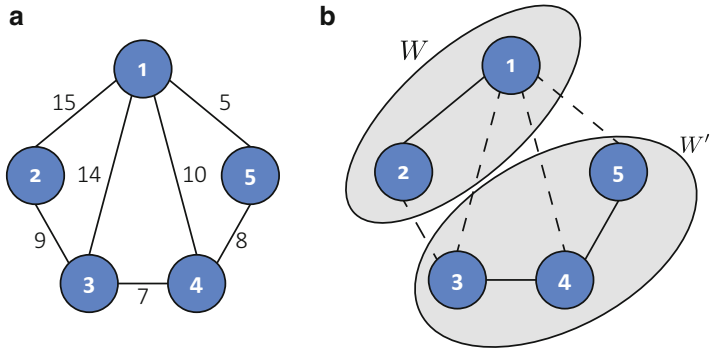


Fig. 1 Example of graph with five vertices and seven edges and a possible solution for the MCP. (**a**) Example of a graph. (**b**) Solution $x = \{1, 1, 0, 0, 0\}$

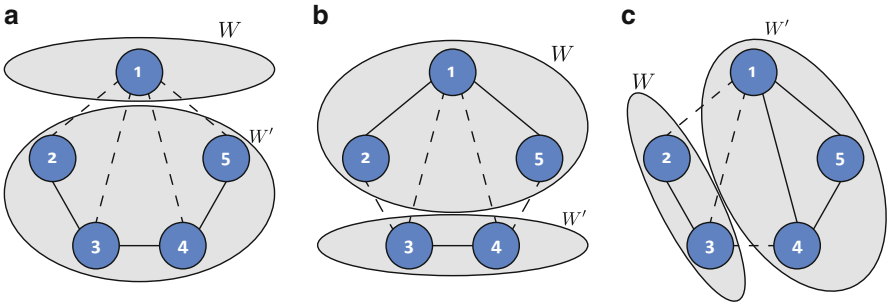


Fig. 2 Solutions generated when performing the different moves to the one depicted in Fig. 1b. (**a**) $x' \leftarrow drop(x, 2)$. (**b**) $x'' \leftarrow add(x, 5)$. (**c**) $x''' \leftarrow swap(x, 1, 3)$

Figure 2a shows an example of a this type of move. In particular, $drop(x, 2)$ considers the solution x (depicted in Fig. 1b), removes the vertex 2 from W, and includes it in W', producing a new solution $x^{drop} = \{1, 0, 0, 0, 0\}$. The Hamming distance between x and x' is 1 since there is only one vertex located in a different group.

Symmetrically, we define the $add(x, i)$ as the move operator responsible of changing the value of a variable x_i from 0 to 1, producing a new solution x^{add}. Considering the MCP, this move operator removes a vertex from W' and includes it in W. This move is represented as $x'' \leftarrow add(x, i)$, and the neighborhood, $\mathcal{N}^{Bin}_{add}(x)$, with size n in the worst case, is:

$$\mathcal{N}^{Bin}_{add}(x) = \{x'' \leftarrow add(x, i) : x_i = 0 \wedge 1 \leq i \leq n\}$$

Figure 2b shows the move $add(x, 1)$. It considers again the solution x (depicted in Fig. 1b), removes the vertex 1 from W' including it in W. The Hamming distance between the new produced solution, $x'' = \{1, 1, 0, 0, 1\}$, and x is again 1.

Finally, we define $\mathcal{N}_{swap}^{Bin}(x)$ neighborhood as a subset of $\mathcal{N}_2^{Bin}(x)$ defined by *swap* move. *Swap* move is defined as an operation that changes the value of one variable x_i from 1 to 0, and simultaneously a different variable x_j change the value from 0 to 1. This move interchanges a vertex from W to W' and, simultaneously a different vertex from W' to W. This move is represented as $x''' \leftarrow swap(x, i, j)$, and the neighborhood, $\mathcal{N}_{swap}^{Bin}(x)$ (with size n^2 in the worst case), is:

$$\mathcal{N}_{swap}^{Bin}(x) = \{x''' \leftarrow swap(x, i, j) : x_i \neq x_j \wedge 1 \leq i, j \leq n\}.$$

Figure 2c shows the move $swap(x, 1, 3)$. It considers again the solution x (depicted in Fig. 1b) and interchanges the vertex 1 and 3 between W and W', respectively. The Hamming distance between the new produced solution, $x''' = \{0, 1, 1, 0, 0\}$, and x is in this case 2.

Neighborhoods for Integer Problems

There are some optimization problems where the solution x is represented as a vector of n variables, $x = (x_1, x_2, \ldots, x_n)$, such that $l \leq x_j \leq u$ for all j (i.e., assuming that all variables are integer and bounded within the interval $[l, u]$). For example, the solutions to the set covering problem [9], the capacitated task allocation problem [24], or the maximally diverse grouping problems [11] can be represented as a vector of integer values. For solving these problems, neighborhood structures $\mathcal{N}_k^{Int}(x)$ are usually induced from the metric defined as:

$$\rho_{int}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{7}$$

or

$$\rho_{int}(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \tag{8}$$

The neighborhood $\mathcal{N}_k^{Int}(x)$ denotes the set of solutions in the k–th neighborhood of x, and it is defined as

$$\mathcal{N}_k^{Int}(x) = \{y \in X \mid \rho_{int}(x, y) \leq k\} \tag{9}$$

or

$$\mathcal{N}_k^{Int}(x) = \{y \in X \mid k - 1 \leq \rho_{int}(x, y) \leq k\} \tag{10}$$

We use the maximally diverse grouping problem to illustrate neighborhoods for integer problems. It consists of grouping a set of M elements into G mutually

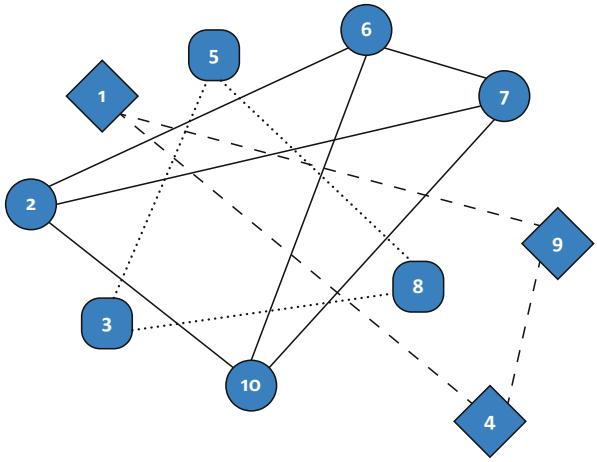


Fig. 3 Example of the maximally diverse grouping problem

disjoint groups in such a way that the diversity among the elements in each group is maximized. The diversity among the elements in a group is calculated as the sum of the individual distances between each pair of elements, where the notion of distance depends on the specific application context. The objective of the problem is to maximize the overall diversity, that is, the sum of the diversity of all groups. Figure 3 shows an example of a graph with ten vertices where we must select three groups of elements. For the sake of clarity, we only represent the distances among selected elements. In addition, we use different shapes to differentiate the vertices on each group. In particular, group 1 (diamond shape) is $G_1 = \{1, 4, 9\}$ whose distances among edges are represented with dashed lines; group 2 (rectangle shape) is $G_2 = \{3, 5, 8\}$ represented with dotted lines; and group 3 (circle shape) is $G_3 = \{2, 6, 7, 10\}$ represented with solid lines. Therefore, this solution can be represented as a vector $x = \{1, 3, 2, 1, 2, 3, 3, 2, 1, 3\}$ (i.e., each vertex i is located in group x_i).

With this representation, we define the following moves:

Exchange of values (Swap move): given a solution x, the exchange of the values of the variables x_i and x_j is denoted by $swap(x, i, j)$ and generates a new solution x' such that $x'_k = x_k, \forall k \neq i, j$, and $x'_i = x_j, x'_j = x_i$. Note that resulting solution x' belongs to the neighborhood

$$\mathcal{N}_0^{Int}(x) = \{y \in X \mid \max_{1 \leq i \leq n} |x_i - y_i| = 0\}.$$

Replacement of a single value (Replacement move): given a solution x, the replacement move denoted by $replace(x, j, i)$ creates a solution x' for which

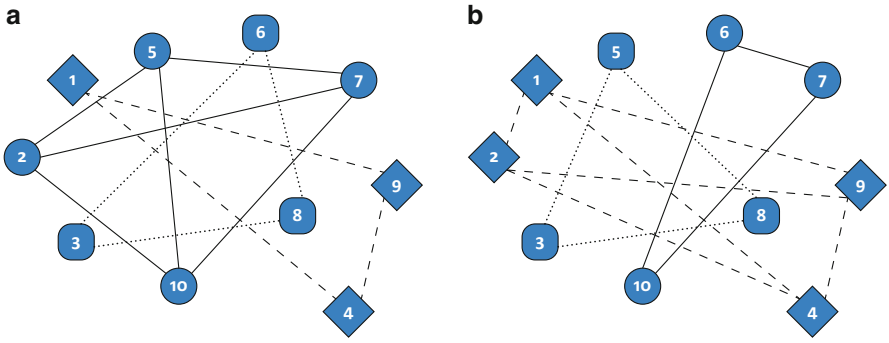


Fig. 4 Example of moves applied to a solution for the maximally diverse grouping problem. (**a**) $x' \leftarrow swap(x, 6, 5)$. (**b**) $x'' \leftarrow replace(x, 2, 1)$

$x'_k = x_k$, $\forall k \neq j$, and $x'_j = i$. The move is such that $x_j \neq i$. In this case, the resulting solution x' belongs to the neighborhood

$$\mathcal{N}_m^{Int}(x) = \{y \in X \mid \sum_{1 \leq i \leq n} |x_i - y_i| \leq m\},$$

where $m = |x_j - i|$.

Given the solution $x = \{1, 3, 2, 1, 2, 3, 3, 2, 1, 3\}$, the move $swap(x, 6, 5)$ interchanges the corresponding group of vertices 6 and 5, producing the solution $x' = \{1, 3, 2, 1, 3, 2, 3, 2, 1, 3\}$ (see Fig. 4a). Similarly, the move $replace(x, 2, 1)$ includes the vertex 2 in group 1 (removing it from its original group). The final solution in this case is $x'' = \{1, 1, 2, 1, 2, 3, 3, 2, 1, 3\}$ (see Fig. 4b).

Neighborhoods for Permutation Problems

In permutation-based representations, a solution is typically expressed as a labeling (permutation), where each element receives a unique label from 1 to n, being n the size of the problem. The traveling salesman problem [35], the linear ordering problem [30], or the cutwidth minimization problem [34] are examples of this kind of problems. For solving these problems, neighborhood structures $\mathcal{N}_k^{Int}(x)$ may be induced using several metrics (see e.g., [4] for possible metric). However, neighborhoods are usually induced from the Cayley distance:

$$\rho_{perm}(x, y) := \text{the minimum number of transpositions needed to obtain y from x} \tag{11}$$

So, the neighborhood $\mathcal{N}_k^{Perm}(x)$ denotes the set of solutions in the k–th neighborhood of x, and it is defined as:

$$\mathcal{N}_k^{Perm}(x) = \{y \in X \mid \rho_{perm}(x, y) = k\} \tag{12}$$

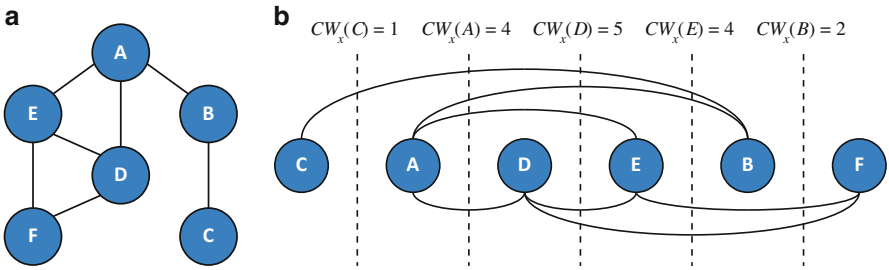


Fig. 5 Graph with six vertices and seven edges and an example solution for the CWP. (**a**) Example graph. (**b**) Cutwidth of G for a labeling x

We use the cutwidth minimization problem (CMP) to illustrate some neighborhoods for permutation-based problem. It can be easily described in mathematical terms. Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, a labeling or linear arrangement x of G assigns the integers $\{1, 2, \ldots, n\}$ to the vertices in V, where each vertex receives a different label. The cutwidth of a vertex v with respect to x, $CW_x(v)$, is the number of edges $(u, w) \in E$ satisfying $x(u) \leq x(v) < x(w)$. In mathematical terms:

$$CW_x(v) = |\{(u, w) \in E : x(u) \leq x(v) < x(w)\}|$$

The cutwidth of G with respect to x is defined as the maximum value of all $CW_x(v)$ for $v \in V$. More formally:

$$CW_x(G) = \max_{v \in V} CW_x(v)$$

The optimum cutwidth of G is then defined as the minimum $CW_x(G)$ over all possible layouts of G. This optimization problem is $\mathcal{NP}$-hard even for graphs with a maximum degree of three [27]. Figure 5a shows an example of an undirected graph with six vertices and seven edges. Figure 5b shows a labeling, x, of the graph in Fig. 5a, setting the vertices in a line with the order of the labeling, as commonly represented in the CMP. We represent x with the ordering (C, A, D, E, B, F) meaning that vertex C is located in the first position (label 1), vertex A is located in the second position (label 2), and so on. In Fig. 5c, the cutwidth of each vertex is represented as a dashed line with its corresponding value. For example, the cutwidth of vertex C is $CW_x(C) = 1$, because the edge (C, B) has an endpoint in C labeled with 1 and the other endpoint in a vertex labeled with a value larger than 1. In a similar way, we can compute the cutwidth of vertex A, $CW_x(A) = 4$, by counting the appropriate number of edges: (C, B), (A, B), (A, E), and (A, D). Then, since the cutwidth of the graph G, $CW_x(G)$, is the maximum of the cutwidth of all vertices in V, in this particular example, we obtain $CW_x(G) = CW_x(D) = 5$.

The associated neighborhoods for this optimization problem are typically based on two different move operators. The first one is referred to as *exchange*. Given a solution $x = (v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n)$, we define $exchange(x, i, j)$ as exchanging in x the vertex in position i (i.e., vertex v_i) with the vertex in position j (i.e., vertex v_j), producing a new solution $x' = (v_1, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_n)$. So, *exchange* move represents a transposition. For the sake of simplicity, we denote $x' = exchange(x, i, j)$. The associated neighborhood $\mathcal{N}^{Perm}_{exchange}$, obtained applying all possible *exchange* moves, has size $n(n-1)/2$, and it is formally defined as:

$$\mathcal{N}^{Perm}_{exchange}(x) = \{x' \leftarrow exchange(x, i, j) : i \neq j \wedge 1 \leq i, j \leq n\}$$

This neighborhood is actually $\mathcal{N}^{Perm}_{1}$ neighborhood.

The second move operator for CMP is known as *insert*. Specifically, given a solution x, we define $insert(x, j, v_i)$ as the move consisting of deleting v_i from its current position i and inserting it in position j. This operation results in the new solution x' as follows:

- If $i \geq j$, then $x = (\ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots)$, and the vertex v_i is inserted just before the vertex v_j, obtaining $x' = (\ldots, v_{j-1}, v_i, v_j, v_{j+1}, \ldots, v_{i-1}, v_{i+1}, \ldots)$.
- If $i < j$, then $x = (\ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots)$, and the vertex v_i is inserted just after the vertex v_j, obtaining $x' = (\ldots, v_{i-1}, v_{i+1}, \ldots, v_{j-1}, v_j, v_i, v_{j+1}, \ldots)$.

Thus, a move $insert(x, j, v_i)$ generates a solution belonging to the neighborhood $\mathcal{N}^{Perm}_{|j-i|}(x)$. The associated neighborhood, $\mathcal{N}^{Perm}_{insert}$, obtained applying all possible *insert* moves has size $n(n-1)/2$, and it is formally defined as:

$$\mathcal{N}^{Perm}_{insert}(x) = \{x' \leftarrow insert(x, j, v_i) : i \neq j \wedge 1 \leq i, j \leq n\}$$

The following example illustrates how the *insert* is implemented. Let $x = (C, A, D, E, B, F)$ be a solution of the cutwidth problem. Suppose that we perform $insert(x, 2, B)$, obtaining solution $x' = (C, B, A, D, E, F)$. Figure 6 graphically shows the solutions before and after the move.

Local Search Methods

Local search methods are likely the oldest and simplest heuristic methods. Starting from a given feasible solution, these procedures explore a determined neighborhood in each iteration, replacing the current solution if a neighbor improves the objective function. The search ends when all neighbor solutions are worse (i.e., larger objective function value in a minimization problem) meaning that a local optimum is found.

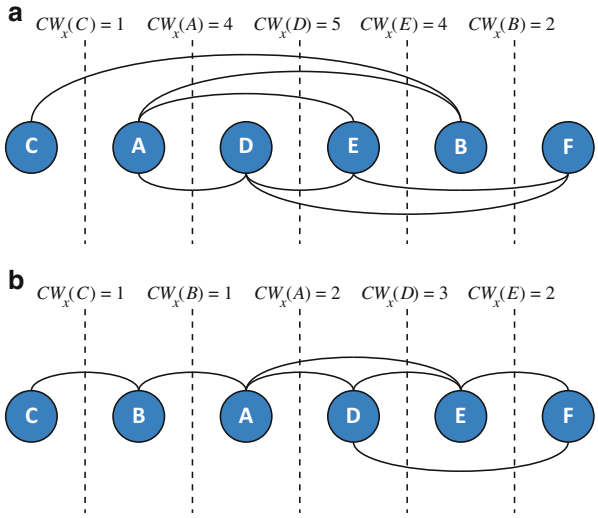


Fig. 6 Example of performing an insert move in a permutation solution. (**a**) Solution x before the move. (**b**) Resulting solution after $insert(x, 2, B)$

Algorithm 1: Best improvement(x)

```
1:  improve ← true
2:  while (improve) do
3:      improve ← false
4:      x' ← arg min_{x∈N(x)} f(x)
5:      if f(x') < f(x) then
6:          x ← x'
7:          improve ← true
8:      end if
9:  end while
10: return x
```

There exist two typical strategies to explore the corresponding neighborhood: **best improvement** and **first improvement**. In the former (also known as **steepest descent**), the associated neighborhood is completely explored by a fully deterministic procedure, performing the best associated move. Therefore, no matter how the neighborhood is scanned, since all neighbor solutions are visited. Algorithm 1 shows the typical pseudocode of this local search method for a minimization problem. The algorithm starts by initializing the control variable *improve* (step 1). Then, the best improvement strategy performs iterations until it finds a local optimum with respect to neighborhood $N(x)$ (steps 2–9). Given a solution x, the best neighbor solution x' is determined in step 4. This instruction has a computational complexity of $|N(x)|$. In steps 5–8, it is decided whether to perform a new iteration (by updating x to x') or not (abandoning the search).

The first improvement strategy tries to avoid the time complexity of exploring the whole neighborhood by performing the first improving move encountered during the exploration of the corresponding neighborhood. In this kind of exploration, the order in which the neighbors are inspected can have a significant influence of the efficiency of the search. Instead of using a fixed ordering for scanning the neighbors, random orders are usually suggested, since the first scanning strategy always drives to the same local optimum, while the second one can reach different local optima.

The pseudocode of this strategy is equivalent to the one presented in Algorithm 1. The only difference is in step 4, where, instead of selecting the best neighbor, it selected the first neighbor which improves the incumbent solution (in terms of objective function value). Additionally, if we follow a predefined order (e.g., a lexicographic order), the first positions in that order (e.g., $1, 2, \ldots$, in the lexicographic order) are favored, producing a kind of bias in the search.

Figure 7 shows the performance of two iterations of both strategies starting from the same solution, where the numbers over the arrows indicate the order of exploration of the solutions. Specifically, considering the MCP described in section "Neighborhoods for Binary Problems", the initial solution is $x_1 = [1, 1, 0, 0, 0]$, representing $W = [1, 2]$ and $W' = [3, 4, 5]$, with an objective function value of 38. The neighborhood selected to be explored is $\mathcal{N}_{add}^{Bin}(x_1)$, where each neighbor is generated by adding a new vertex to W (removing it from W'). In the first iteration, the best improvement strategy (Fig. 7a) generates all possible neighbor solutions for x_1 (i.e., x_2, x_3, x_4). The exploring order, as stated before, is irrelevant, since it is going to explore the whole neighborhood. Then, the method selects the best neighbor solution, which is x_3 in Fig. 7a, with a higher (better) objective function value of 43. Finally, in the next iteration, the method explores the entire x_3 neighborhood, stopping after the exploration, since there is no better neighbor solution.

Regarding the first improvement method (Fig. 7b), the exploration order (which is relevant in this case) is selected at random, starting with solution x_2. As the objective function value of x_2 is lower (worse) than x_1, the method explores another neighbor, which is x_4. The objective function value for x_4, 41, is better than for x_1, 38, so in this case, the first improvement strategy stops the iteration, starting the next one from this new best solution x_4. Notice that although solution x_3 is better than x_4, it is not explored in this strategy, since the order selected has lead the method to find a better solution before reaching x_3. Finally, in the next iteration, the method starts from x_4 and then explores, in this order, x_6 and x_5, stopping the search since there is no improvement in any neighbor.

In the context of large neighborhoods, there is a compromise between the number of iterations required to find a local optimum and the associated computing time. In general, iterations performed in the first improvement strategy are more efficient than those in the best improvement one, since the former only evaluates part of the neighborhood, while the latter explores it completely. On the other hand, the improvement obtained in the first improvement strategy is typically smaller than the one achieved by the best improvement strategy, requiring in general more iterations to obtain the local optimum. Additionally, the best improvement strategy

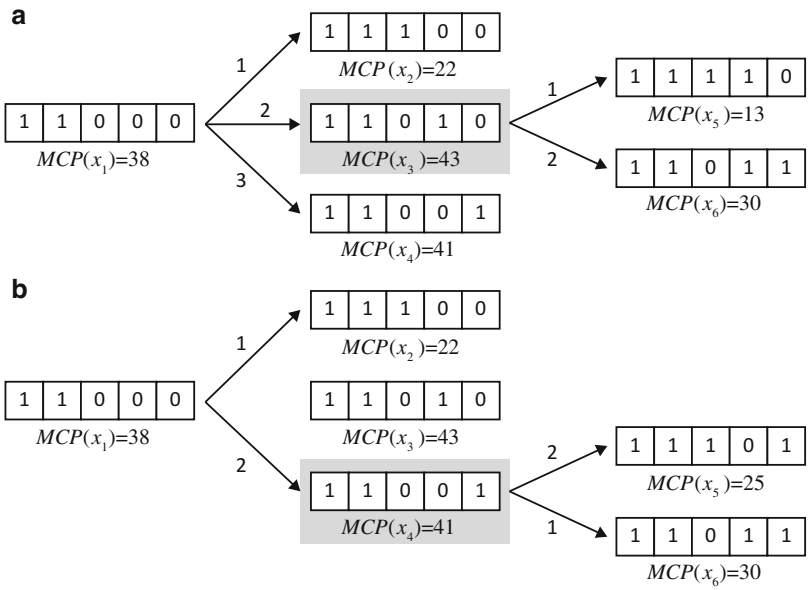


Fig. 7 Comparison of best and first improvement strategies when starting from the same solution for the MCP. (**a**) Best improvement. (**b**) First improvement

is usually more adequate to perform efficient catching and updating mechanisms, which allows the search to efficiently explores the neighborhood [20].

In [17], an empirical study on traveling salesman problem was conducted in order to compare the first and the best improvement strategy within 2-opt neighborhood structure. It appeared that the quality of the final solution depends on the quality of the initial solution: (i) if random initial solution is chosen, the better and faster is the first improvement strategy; (ii) the opposite holds if the greedy solution is taken as initial one.

VND Variants

The Variable Neighborhood Search (VNS) is a metaheuristic proposed in [32] as a general framework to solve hard problems. It is based on a simple idea: systematical changes of neighborhood structures within the search procedure. Let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$ be set of operators such that each operator $\mathcal{N}_k$, $1 \leq k \leq k_{\max}$ maps a given solution x to a neighborhood structure $\mathcal{N}_k(x)$. Note that the order of operators taken from the set $\mathcal{N}$ defines also the order of neighborhood structures of a given solution x examined. When solving an optimization problem by using different neighborhood structures, VNS methodology proposes to explore them in three different ways: (i) at random, (ii) deterministically, or (iii) mixed (both, in deterministic and random fashion).

Variable neighborhood descent (VND) is a variant of VNS that explores neighborhoods in a deterministic way. In general, VND explores small neighborhoods until a local optimum is encountered. At that point, the search process switches to a different (typically larger) neighborhood that might allow further progress. This approach is based on the fact that a local optimum is defined with respect to a neighborhood relation, such that if a candidate solution x is locally optimal in a neighborhood $\mathcal{N}_i(x)$, it is not necessarily a local optimum for another neighborhood $\mathcal{N}_j(x)$. Thus, VND explores the solution space using several neighborhood structures either in a (i) sequential, (ii) a nested (or composite), or (iii) mixed nested way [21, 43].

Sequential Variable Neighborhood Descent Procedures

Most typical VND variants traverse the list of neighborhood structures in a sequential way. Within this variants, the basic VNS, the pipe VND, the cyclic VND, and the union VND emerge as the most representative search procedures. These variants differ in how they implement the neighborhood change procedures. Specifically, if an improvement has been detected in some neighborhood, this is how the search (after updating the incumbent solution) is continued:

- **Basic VND (B-VND)** – returns to the first neighborhood from the list
- **Pipe VND (P-VND)** – continues the search in the same neighborhood
- **Cyclic VND (C-VND)** – resumes the search in the next neighborhood from the list
- **Union VND (U-VND)** (sometimes called multiple neighborhood search [44], where the single neighborhood is obtained as the union of all predefined neighborhoods) – continues the search in the same large neighborhood. U-VND is recently proposed in [2, 26, 44] and used within Tabu search.

All these VND procedures follow the steps given in Algorithm 2 and start from a given solution x. In each VND iteration, a local search procedure through a given neighborhood structure is applied, followed by a neighborhood change function (Step 7). The neighborhood change function defines the neighborhood structure that will be examined in the next iteration. Each VND variant stops when there is no improvement with respect to any of the considered neighborhood structures. Thus, the solution obtained by any sequential VND is a local optimum with respect to all neighborhood structures.

We show the performance of the three aforementioned variants considering the example introduced in [31]. In particular, it shows an empirical study on different VND variants used to solve traveling salesman problem. For testing purposes, 15,200 random test instances were generated in the way described in [17]. Within VND variants, three classical TSP neighborhood structures are considered: 2-opt (Fig. 8), Insertion-1 (Fig. 9), and Insertion-2 (Fig. 10). On each instance from this data set, each VND variant, except U-VND, is tested under 24 different settings.

Algorithm 2: Sequential variable neighborhood descent

Function `SeqVND` $(x, k_{\max}, \mathcal{N})$
1 $x'' \leftarrow x$
2 `Stop= False`
while `Stop= False` **do**
3 $x \leftarrow x''$
4 `Stop= True`
5 $k \leftarrow 1$
while $k \leq k_{\max}$ **do**
6 $x' \leftarrow \arg\min_{y \in \mathcal{N}_k(x)} f(y)$
7 `Change Neighborhood` (x, x', k)
8 **if** $f(x') < f(x'')$ **then**
9 $x'' \leftarrow x'$
10 `Stop= False`
end
end
end
11 **return** x'';

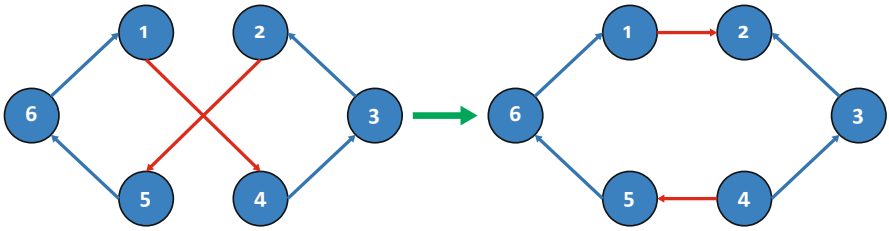


Fig. 8 Example of a 2-opt move involving vertices 1, 2, 4, and 5

Each setting corresponds to choosing the following: (i) one out of two common ways for getting an initial solution, at *random* (solution generated as a random permutation of nodes) or *greedy*; (ii) one out of six possible neighborhood orders; and (iii) the *best* or the *first improvement search strategy*. This gives $2 \times 6 \times 2 = 24$ different search methods that use 2-opt, Insertion-1, and Insertion-2 neighborhoods. On the other hand, U-VND is tested under only two different settings as: U-VND that uses the best improvement search strategy and the greedy initial solution and U-VND that uses the best improvement search strategy and the random initial solution. Note that if the first improvement search strategy is used within U-VND, then U-VND is equivalent to basic VND.

Table 1 summarizes results considering the entire set of 15,200 instances as a test case. Namely, the average solution values (Column '`av. best value`') and

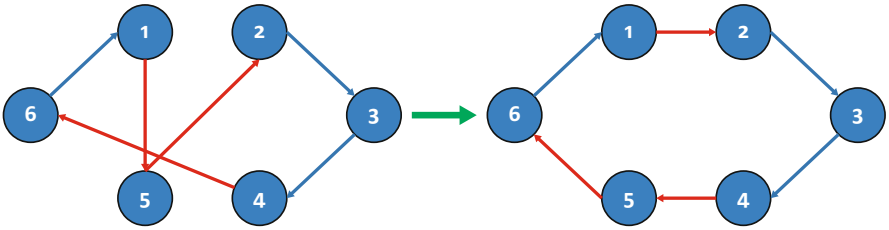


Fig. 9 Example of an Insertion-1 move where vertex 5 is inserted between vertices 1 and 2 and removed from its corresponding former position

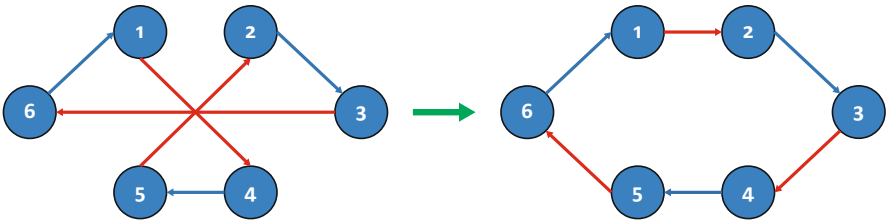


Fig. 10 Example of an Insertion-2 move where vertices 4 and 5 are inserted between vertices 1 and 2, removing them from its corresponding former position

Table 1 Comparison of VND variants

VND variant	av. best value	av. time (s)
Basic VND	1198.24	0.16
Pipe VND	1198.52	0.12
Cyclic VND	1198.76	0.46
Union VND	1197.65	1.06

average CPU times (Column '`av. time`') over all test instances in the data set, attained by VND variants under the best settings, are reported.

From the results presented in Table 1, the following conclusions may be drawn: (i) U-VND is slightly better than the others VNDs, regarding the best average values attained, but much slower than the others. This is explained by the fact that U-VND in each iteration performs exploration of large part of the solution space before deciding to re-center the search. Obviously, such principle is suitable for reaching good final solution, but requires a large CPU time; (ii) comparing VNDs that re-center the search in the inner loop (i.e., B-VND, P-VND, and C-VND), it follows that B-VND is able to provide the best solution values. Regarding average CPU times consumed by B-VND, P-VND, and C-VND to find the best-reported average solution value, their ranking is as follows: the fastest is P-VND, B-VND is ranked as the second, while C-VND is the slowest one. However, since, B-VND consumed negligible more CPU time to find the best reported average solution value comparing to CPU time that P-VND consumed to do so, we may conclude that B-VND is the most appropriate sequential VND version.

Nested Variable Neighborhood Descent

A nested (composite) variable neighborhood descent procedure [21] explores a large neighborhood structure obtained as a composition of several neighborhoods. More precisely, let $\mathcal{N} = \{N_1, \ldots N_{k_{\max}}\}$ again be set of move operators such that each one N_k, $1 \leq k \leq k_{\max}$ maps a given solution x to a predefined neighborhood structure $N_k(x)$. Then, the neighborhood explored within a nested variable neighborhood procedure is defined with operator $N_{nested} = N_1 \circ N_2 \circ \cdots \circ N_{k_{\max}}$. More precisely, the composite neighborhood of solution x is formed by first applying the move operator N_1, obtaining $N_1(x)$. Then, the move operator N_2 is applied to all solutions in $N_1(x)$, forming the set $N_1(N_2(x))$ and so on. Obviously, the cardinality of a neighborhood structure $N_{nested}(x) = N_1(N_2(\ldots(N_{k_{\max}}(x))))$ of some solution x is less than or equal to the product of cardinalities of nested (composed) neighborhoods, i.e.,

$$|N(x)| \leq \prod_{k=1}^{k_{\max}} |N_k(x)|.$$

Such cardinality obviously increases chances to find an improvement in the neighborhood. The nested VND is illustrated in Algorithm 3. The neighborhood $N(x)$ may be explored by using either the first or the best improvement search strategy. However, since its cardinality is usually very large, the first improvement is used more often [21,43].

Algorithm 3: Steps of best improvement nested VND

Function `Nested_VND`$(x, k_{\max}, \mathcal{N})$
$N_{nested} = N_1 \circ N_2 \circ \cdots \circ N_{k_{\max}}$
repeat
  $x' \leftarrow x$;
  $x \leftarrow \text{argmin}_{y \in N(x')} f(y)$;
until $f(x') \leq f(x)$;
return x';

We use the uncapacitated r-allocation p-hub median problem (r-p-HMP) to illustrate how this VND strategy works. Specifically, the r-p-HMP may be stated as follows. Given n nodes, this problem considers for each pair of nodes i and j, the distance d_{ij} and the amount of flow $t_{ij} \geq 0$ that needs to be transferred from i to j. It is generally assumed that the transportation between non-hub nodes i and j is only possible via hub nodes h_i and h_j, to which nodes i and j are assigned, respectively. Transferring t_{ij} units of flow through path $i \rightarrow h_i \rightarrow h_j \rightarrow j$ induces a cost $c_{ij}(h_i, h_j)$, which is computed as

$$c_{ij}(h_i, h_j) = t_{ij}(\gamma d_{ih_i} + \alpha d_{h_i h_j} + \delta d_{h_j j}).$$

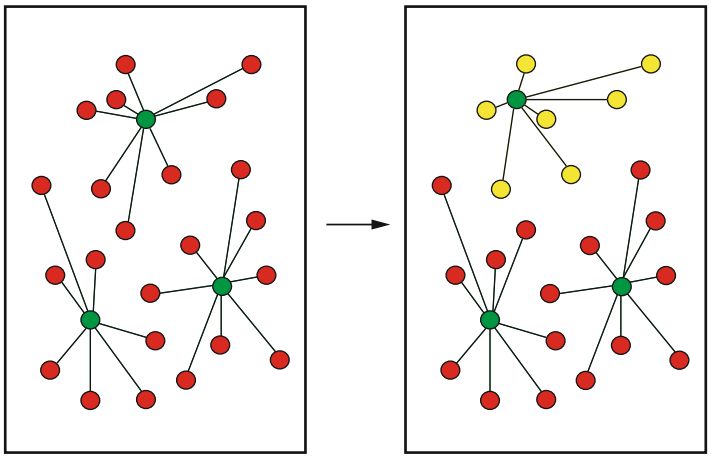


Fig. 11 Interchange neighborhood $\mathcal{N}_H$. *Yellow dots* are the possible new hub location for the one represented in *green*

Parameters γ, α, and δ represent unit rates for collection (origin-hub), transfer (hub-hub), and distribution (hub-destination), respectively. Note that the hub nodes h_i and h_j may be equal.

We represent a solution of r-p-HMP by a set H containing p hubs and a matrix A, where each row i contains r hubs assigned to node i (i.e., i-th row coincides with the set H_i). Thus, our solution is represented as $x = (H, A)$. The initial solution is obtained using the greedy heuristic described in [38]. We consider two neighborhood structures of a given solution $x = (H, A)$. The first neighborhood structure, denoted by $\mathcal{N}_H$, is obtained by replacing one hub node from H by another non-hub node from $N \setminus H$ (see Fig. 11). More formally,

$$\mathcal{N}_H(x) = \{x' \mid x' = (H', A'), |H \cap H'| = p - 1\}.$$

The second neighborhood, denoted by $\mathcal{N}_A$, is obtained by replacing one hub assigned to some node with another hub, while the set H remains unchanged (see Fig. 12):

$$\mathcal{N}_A(x) = \{x' | x' = (H, A'), |A \backslash A'| = 1\}.$$

Unfortunately, evaluating the objective function value of a solution from $\mathcal{N}_H$ requires to solve an allocation problem, which is $\mathcal{NP}$ hard by itself [25]. Therefore, solving exactly the associated allocation problem will be quite time consuming. In order to deal with this drawback, we find near-optimal allocation

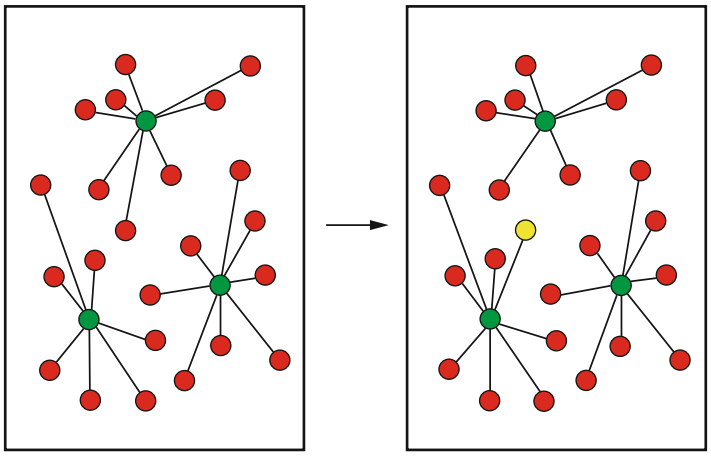


Fig. 12 Allocate neighborhood $\mathcal{N}_A$. *Yellow dot* represents the non-hub node which is reallocated to the another hub

A' of some solution $x'' \in \mathcal{N}_H$ as follows. For each node, we firstly determine node-to-hub allocation using the greedy procedure (see Algorithm 4).

Algorithm 4: Greedy allocation

Function `GreedyAllocation(H, A)`

for $i \in N$ **do**
  for $j = 1$ **to** p **do** $value(j) = d_{ih_j} \sum_{k \in N} t_{ik} + \sum_{k \in N} t_{ik} d_{h_j k}$;
  Sort array $value$ in nondecreasing order i.e.,
  $value(\pi(1)) \leq value(\pi(2)) \leq \cdots \leq value(\pi(p))$;
  for $j = 1$ **to** r **do** $A[i][j] = h_{\pi(j)}$;
end

The solution $x'' = (H'', A'')$ obtained in this way is then improved by exploring the second neighborhood $\mathcal{N}_A(x'')$. In that way, the so-called nested variable neighborhood descent (Nest-VND) is defined (see Algorithm 5).

Algorithm 5: Nested VND for r-p-HMP

Function `NestVND(x);`

for each $x'' \in N_H(x)$ **do**
  `GreedyAllocation(H", A");`
  Select x' as the best solution in $N_A(x'')$;
  if x' is better than x **then** $S \leftarrow x'$
end
return x

Table 2 Comparison of GRASP and GVNS on AP instances

	GRASP		GVNS_RP		
n	Aver. value	Aver. time	Aver. value	Aver. time	% impr.
60	122348.90	4.59	121829.27	3.73	0.42
65	123001.53	6.66	122689.74	5.87	0.25
70	123931.76	10.51	123604.38	5.75	0.26
75	124776.42	11.11	124650.73	5.93	0.10
80	125148.22	14.40	124844.76	9.36	0.24
85	125566.58	19.48	125378.23	13.10	0.15
90	124934.99	22.95	124734.55	12.32	0.16
95	125121.18	24.27	124926.55	25.45	0.16
100	125805.04	4.81	125588.19	10.39	0.17
150	126728.85	21.42	126307.10	24.70	0.33
200	129144.44	58.86	128788.66	98.67	0.28
Avg.	125137.08	18.10	124849.29	19.57	0.23

In [43], the above nested VND was used as a local search within a general variable neighborhood search (GVNS)-based heuristic. The performance of this heuristic, named `GVNS_RP`, was compared with the GRASP heuristic proposed in [38]. In Table 2, we provide a summarized results of comparison of `GVNS_RP` and GRASP on instances from AP data set (see [43]). The average value of best found solutions and average CPU times needed for finding these solutions over all instances with the same number of nodes are reported. The column headings are defined as follows. In the first column of Table 2, we report the number of nodes in the considered instances, whereas in the columns "GRASP" and "`GVNS_RP`," we provide the average of best solution values found by GRASP and `GVNS_RP`, respectively. In columns "`time`," the average time needed to reach best found solutions for instances with n nodes is given, while in column "`impr. (%)`," we report the percentage improvement obtained by `GVNS_RP` compared with the current best known values. From the reported results, it follows that within each set of instances with the same number of nodes, there is at least one instance where the best known solution is improved by `GVNS_RP`. Moreover, the average improvement on AP data set achieved by GVNS variants is around 0.25 %.

Mixed Variable Neighborhood Descent

Mixed variable neighborhood descent [21] combines ideas of sequential and nested variable neighborhood descent. Namely, it uses a set of move operators $\mathcal{N} = \{N_1, \ldots N_b\}$ to define a nested neighborhood, and, after that, on each element in this nested neighborhood, it applies a sequential variable neighborhood descent variant defined by a set of move operators $\mathcal{N}' = \{N_{b+1}, \ldots N_{k_{\max}}\}$. The cardinality of the set explored in one iteration of a mixed VND is bounded by:

$$|N_{mixed}(x)| \leq \prod_{\ell=1}^{b} |N_\ell(x)| \times \sum_{\ell=b+1}^{k_{\max}} |N_\ell(x)|, x \in S.$$

In Algorithm 6, we show the pseudocode of a mixed VND. Note that if the set $\mathcal{N}$ is the empty set (i.e., $b = 0$), we get pure sequential VND. If $b = k_{\max}$, we get pure nested VND. Since nested VND intensifies the search in a deterministic way, boost parameter b may be seen as a way of balancing intensification and diversification in deterministic local search with several neighborhoods.

Algorithm 6: Steps of mixed VND

Function `Mixed_VND`$(x, b, k_{\max}, \mathcal{N}, \mathcal{N}')$
$N = N_1 \circ N_2 \circ \cdots \circ N_b$ $x' \leftarrow x$;
repeat
  $stop = true$;
  $x \leftarrow x'$;
  for *each* $y \in N(x)$ **do**
    $x'' \leftarrow$ `SeqVND`$(y, k_{\max} - b, \mathcal{N}')$;
    if $f(x'') < f(x')$ **then**
      $stop = false$ $x' \leftarrow x''$;
    end
  end
until $stop = true$;
return x';

In [21], two mixed VND heuristics along with one basic sequential VND heuristic were proposed for solving the incapacitated single allocation p-hub median problem (USApHMP). Neighborhood structures examined within these VND variants are cluster based. A cluster represents one hub with all locations assigned to it. In particular, the following neighborhood structures are distinguished: (i) *Allocate*- change membership of a non-hub node by reallocating it to the another hub without changing the location of hubs; (ii) *Alternate*-change the location of the hub in one cluster; (iii) *Locate* - select one cluster C with hub h and a location node that is not in this cluster. The selected node becomes a hub, and all locations from the cluster C are assigned to the closest hub (including the new one).

The proposed basic sequential VND heuristic, named `Seq-VND`, examines neighborhood structures Allocate, Alternate, and Locate in that order. On the other hand, the first mixed VND heuristic, named `Mix-VND1`, takes several random points from *Locate* neighborhood and starting from each of them carries out search applying a basic sequential VND heuristic `Seq-VND1`. The used `Seq-VND1` explores *Allocate* and *Alternate* neighborhoods in that order. The second mixed VND heuristic, named `Mix-VND2`, performs more thoroughly exploration of the solution space than `Mix-VND1` applying `Seq-VND1` on each point from *Locate* neighborhood. These three VND variants together with three local searches in the three defined neighborhood structures (Allocate, Alternate, and Locate) are

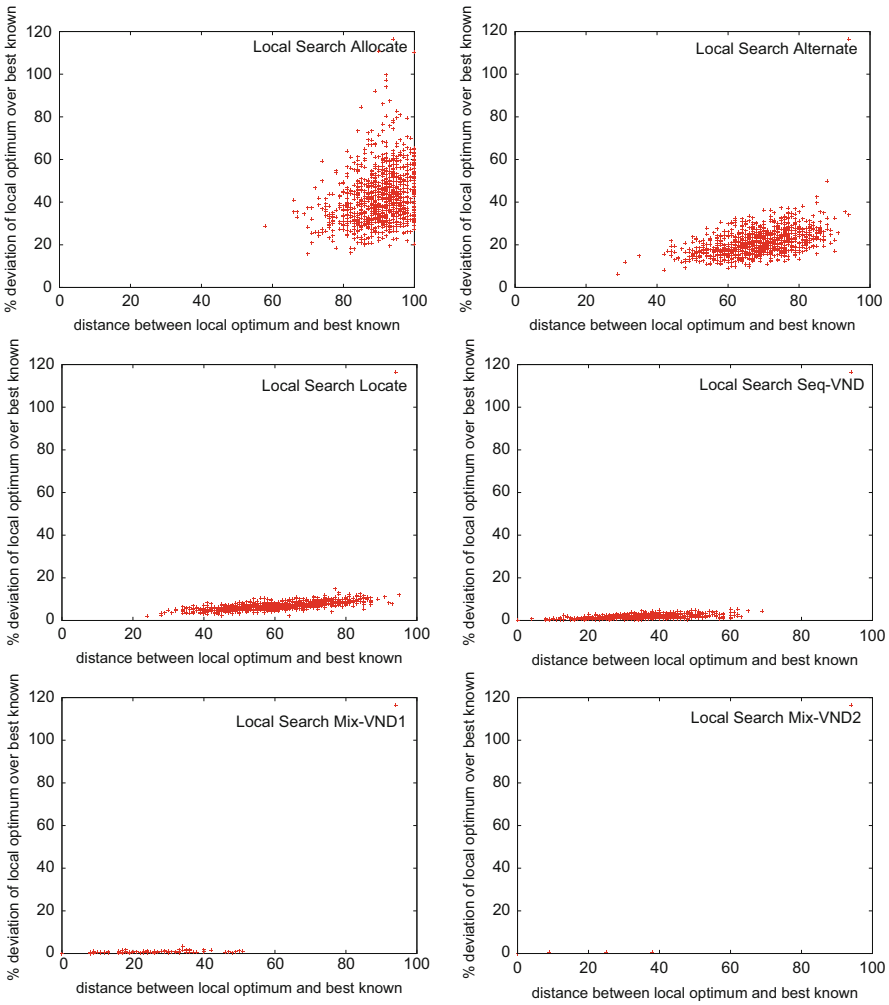


Fig. 13 Multistart on 1000 random initial solutions for AP instance with $n = 100$ and $p = 15$

experimentally compared on AP instance with $n = 100$ and $p = 15$ (see [21]). One thousand initial points are generated at random in the solution space, and a local search is conducted from each one in each type of neighborhood. The distance-to-target diagram [16] presented at Fig. 13 shows the distribution of local minima. Each point (x, y) plots the distance x and percentage deviation y of the local minimum from the best known solution. The results are summarized in Table 3.

Comparing the results in Table 3, we observe that (i) the search in first (*Allocate*) and second (*Alternate*) local neighborhoods is faster than the others, but they have the worse quality; (ii) `Seq-VND` gives better results than the third local search (*Locate*) and is considerably faster; (iii) all three multistart VND local searches

Table 3 Comparison of six local searches on AP instance with $n = 100$ and $p = 15$

	`Allocate`	`Alternate`	`Locate`	`Seq-VND`	`Mix-VND1`	`Mix-VND2`
Average % dev	41.28	21.17	7.41	1.62	0.35	0.16
Minimum % dev.	15.87	6.14	2.58	0.00	0.00	0.00
Maximum % dev.	110.63	49.84	14.6	5.37	3.41	0.68
Average CPU time (sec)	0.002	0.004	0.15	0.06	3.36	27.15

found the best known solution in the AP instance; and (iv) the mixed VND versions give the best results, although, as expected, solution times are longer than solution time of `Seq-VND`.

We see in Fig. 13 that the number of local minima is reduced when VND is used. This is due to the larger neighborhood in VND. For example, the `Seq-VND` neighborhood clearly contains each of the individual neighborhoods, *Allocate*, *Alternate*, and *Locate*. It is remarkable that `Mix-VND2`, which utilizes the largest neighborhood, yields only four local minima for this problem instance.

Conclusions

Local search represents one of the most popular classical heuristic technique that improves (locally) the current feasible solution of some continuous or discrete optimization problem. For that purposes, usually only one neighborhood structure is defined and explored to improve the incumbent solution. Searching for the better solution in such neighborhood is repeated until there is no better solution, i.e., until the local minimum with respect to that predefined neighborhood is reached.

Since the local minimum with respect to one neighborhood structure is not necessary local in another, one needs to construct deterministic local search in cases when more than one neighborhood is used. Such procedures are known as variable neighborhood descent (VND). In this VND survey, we first propose a possible classification of neighborhood structures in solving optimization problems and also provide some simple examples to clearly illustrate the basic ideas. Then, we discuss possible general ways of combining several neighborhoods in the deterministic fashion. Needless to say that the number of possible combination of VND local search variants is large, and they could include problem specific knowledge in building heuristic for each particular problem. Therefore, VND area is an open avenue for the future research in the area of optimization.

Cross-References

▸ Guided Local Search
▸ Multi-Start methods

- Restart Strategies
- Theory of Local Search

Acknowledgments The works of Nenad Mladenović and Raca Todosijević are partly supported by the Ministry of Education and Science, Republic of Kazakhstan (Institute of Information and Computer Technologies), project number 0115PK00546, and also by the Ministry of Education, Science and Technological Development of Serbia, project number 174010. The works of Abraham Duarte and Jesús Sánchez-Oro are partly supported by the Spanish "Ministerio de Economía y Competitividad" and by "Comunidad de Madrid" with grants refs. TIN2012-35632-C02 and S2013/ICE-2894, respectively.

References

1. Brimberg J, Hansen P, Mladenović N (2015) Continuous optimization by variable neighborhood search. In: Wiley encyclopedia of operations research and management science. Wiley, Hoboken, p 1–13. https://doi.org/10.1002/9780470400531.eorms1107
2. Carrasco R, Pham A, Gallego M, Gortázar F, Martí R, Duarte A (2015) Tabu search for the maxmean dispersion problem. Knowl-Based Syst 85:256–264
3. Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A (1997) Combinatorial optimization. Wiley, Chichester
4. Deza M, Huang T (1998) Metrics on permutations, a survey. J Comb Inf Syst Sci 23:173–185
5. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evolut Comput 1(1):53–66
6. Duarte A, Escudero LF, Martí R, Mladenović N, Pantrigo JJ, Sánchez Oro J (2012) Variable neighborhood search for the vertex separation problem. Comput Oper Res 39(12):3247–3255
7. Duarte A, Martí R (2007) Tabu search and GRASP for the maximum diversity problem. Eur J Oper Res 178(1):71–84
8. Duarte A, Sánchez A, Fernández F, Cabido R (2005) A low-level hybridization between memetic algorithm and VNS for the max-cut problem. In: ACM genetic and evolutionary computation conference, New York
9. Feige U (1998) A threshold of Ln N for approximating set cover. J ACM 45(4):634–652
10. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
11. Gallego M, Laguna M, Martí R, Duarte A (2013) Tabu search with strategic oscillation for the maximally diverse grouping problem. J Oper Res Soc 64(5):724–734
12. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co., New York
13. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549
14. Glover F (1998) A template for scatter search and path relinking. In: Selected papers from the third European conference on artificial evolution, AE'97. Springer, London, pp 3–54
15. Hansen P, Mladenović N (1999) An introduction to variable neighborhood search. In: Meta-Heuristics. Springer, Boston, pp 433–458
16. Hansen P, Mladenović N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic Publisher, New York, pp 145–184
17. Hansen P, Mladenović N (2006) First vs. best improvement: an empirical study. Discret Appl Math 154(5):802–817
18. Hansen P, Mladenović N, Todosijević R, Hanafi S (2016) Variable neighborhood search: basics and variants. EURO J Comput Optim 1–32. https://doi.org/10.1007/s13675-016-0075-x
19. Holland JH (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge, MA

20. Hoos H, Süttzle T (2004) Stochastic local search: foundations & applications. Morgan Kaufmann Publishers Inc., San Francisco
21. Ilić A, Urošević D, Brimberg J, Mladenović N (2010) A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur J Oper Res 206(2):289–300
22. Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. The IBM research symposia series. Springer, New York, pp 85–103
23. Laarhoven PJM, Aarts EHL (1987) Simulated annealing: theory and applications. Kluwer Academic Publishers, Norwell
24. Laguna M, Gortázar F, Gallego M, Duarte A, Martí R (2014) A black-box scatter search for optimization problems with integer variables. J Glob Optim 58(3):497–516
25. Love RF, Morris JG, Wesolowski GO (1988) Facilities location: models and methods. Elsevier Science Publishing Co., New York
26. Lü Z, Hao JK, Glover F (2011) Neighborhood analysis: a case study on curriculum-based course timetabling. J Heuristics 17(2):97–118
27. Makedon FS, Papadimitriou CH, Sudborough IH (1985) Topological bandwidth. SIAM J Algebr Discret Methods 6(3):418–444
28. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York
29. Martí R, Duarte A, Laguna M (2009) Advanced scatter search for the max-cut problem. INFORMS J Comput 21(1):26–38
30. Martí R, Reinelt G, Duarte A (2012) A benchmark library and a comparison of heuristic methods for the linear ordering problem. Comput Optim Appl 51(3):1297–1317
31. Mjirda A, Todosijević R, Hanafi S, Hansen P, Mladenović N (2016) Sequential variable neighborhood descent variants: an empirical study on travelling salesman problem. Int Trans Oper Res. https://doi.org/10.1111/itor.12282
32. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
33. Moscato P (1993) An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search. Ann Oper Res 41(1–4):85–121
34. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199(1):285–304
35. Papadimitriou CH (1977) The Euclidean travelling salesman problem is NP-complete. Theor Comput Sci 4(3):237–244
36. Papadimitriou CH, Steiglitz K (1998) Combinatorial optimization: algorithms and complexity. Dover, Mineola
37. Pardo EG, Mladenović N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252
38. Peiró J, Corberán A, Martí R (2014) GRASP for the uncapacitated r-allocation p-hub median problem. Comput Oper Res 43:50–60
39. Ruiz R, Stützle T (2006) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177:2033–2049
40. Sánchez Oro J, Mladenović N, Duarte A (2014) General variable neighborhood search for computing graph separators. Optim Lett 1–21. https://doi.org/10.1007/s11590-014-0793-z
41. Sánchez Oro J, Pantrigo JJ, Duarte A (2014) Combining intensification and diversification strategies in VNS. An application to the vertex separation problem. Comput Oper Res 52, Part B(0):209–219. Recent advances in variable neighborhood search
42. Talbi EG (2009) Metaheuristics: from design to implementation. Wiley, Hoboken
43. Todosijević R, Urošević D, Mladenović N, Hanafi S (2015) A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. Optim Lett. https://doi.org/10.1007/s11590-015-0867-6
44. Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196(1):611–634

Part III
Metaheuristics

Ant Colony Optimization: A Component-Wise Overview

13

Manuel López-Ibáñez, Thomas Stützle, and Marco Dorigo

Contents

Introduction . . . 372
Combinatorial Optimization Problems and Constructive Heuristics . . . 374
The ACO Algorithmic Framework . . . 376
Choice of Pheromone Trails and Heuristic Information . . . 377
Solution Construction . . . 379
Global Pheromone Update . . . 381
Pheromone Update Schedule . . . 384
Initialization of Pheromones . . . 384
Pheromone Reinitialization . . . 384
Local Pheromone Update . . . 385
Pheromone Limits . . . 386
Local Search . . . 386
ACO Algorithms as Instantiations of the ACO Metaheuristic . . . 387
ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP . . . 388
Finding a Better ACO Configuration for the TSP . . . 389
Finding a Better ACO Configuration for the QAP . . . 392
Applications of ACO to Other Problem Types . . . 394
Continuous Optimization Problems . . . 394
Multi-objective Problems . . . 395
Dynamic Problems . . . 396
Stochastic Problems . . . 396
ACO in Combination with Other Methods . . . 397
ACO and Tree Search Methods . . . 397
ACO and Exact Methods . . . 397
ACO and Surrogate Models . . . 398

M. López-Ibáñez (✉)
Alliance Manchester Business School, University of Manchester, Manchester, UK
e-mail: manuel.lopez-ibanez@manchester.ac.uk

T. Stützle · M. Dorigo
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be; mdorigo@ulb.ac.be

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_21

Parameter Adaptation 398
Conclusions 399
Cross-References 400
References 401

Abstract

The indirect communication and foraging behavior of certain species of ants have inspired a number of optimization algorithms for NP-hard problems. These algorithms are nowadays collectively known as the ant colony optimization (ACO) metaheuristic. This chapter gives an overview of the history of ACO, explains in detail its algorithmic components, and summarizes its key characteristics. In addition, the chapter introduces a software framework that unifies the implementation of these ACO algorithms for two example problems, the traveling salesman problem and the quadratic assignment problem. By configuring the parameters of the framework, one can combine features from various ACO algorithms in novel ways. Examples on how to find a good configuration automatically are given in the chapter. The chapter closes with a review of combinations of ACO with other techniques and extensions of the ACO metaheuristic to other problem classes.

Keywords

Ant colony optimization · Automatic configuration · Combinatorial optimization · Metaheuristics

Introduction

Ant colony optimization (ACO) [31, 33, 35] is a metaheuristic that generates candidate solutions by repeated applications of a probabilistic solution construction procedure. In ACO, the probabilities that bias the solution construction are computed from two types of numerical information associated with construction decisions: *heuristic information*, which is derived from the problem instance being tackled, and *artificial pheromone trails*, which are adapted based on the search performance to bias the solution construction toward high-quality solutions. Pheromone trails are the result of a learning mechanism that tries to identify the solution components that, when appropriately combined, lead to high-quality solutions.

As the name suggests, ACO was inspired by entomological experiments on real ant colonies. These experiments demonstrated that some ant species can find the shortest path between their nest and a food source and showed how this mechanism is enabled by the ants' pheromone trail laying and following behavior [22]. The equations used in computer simulations that mimicked the probabilistic behavior of real ant colonies inspired computer scientists to define the way artificial ants take decisions when solving combinatorial optimization problems [30, 36, 38]. Nonetheless, rather than faithfully simulating the behavior of natural ants, the focus of ACO research, since its early days, has been the computational performance of

ACO algorithms and the quality of the solutions that can be generated. To this aim, additional algorithmic techniques have been quickly added to the first ACO algorithms to make them competitive with other metaheuristics. One example of such algorithmic techniques is the exploitation of heuristic information for guiding the construction of solutions [30, 36, 38]; another example is the use of local search algorithms for improving the solutions constructed by the ants [34, 35, 47, 123, 125].

The first ACO algorithm, *ant system*, was proposed by Dorigo et al. [30, 36–38]. After the publication of the first journal paper on ant system in 1996 [38], the research on ACO has gained momentum, and a large number of algorithmic improvements have been proposed. A first improvement over the basic form of ant system was elitist ant system (EAS) [30]. Among the most successful of these successors have been ant colony system (ACS) [34, 45] and max-min ant system (MMAS) [122, 123, 125]. Generally speaking, the main features of these improvements over the basic ant system include mechanisms to intensify the search around high-quality solutions and mechanisms to maintain a sufficient level of search space exploration.

When viewing algorithms from a component-wise perspective, that is, when considering that an algorithm is composed of algorithmic components that fulfill specific tasks for which alternative procedures may exist, there are similarities between various components of MMAS, ACS, and other ACO algorithms such as EAS [30, 38], rank-based ant system (RAS) [19], best-worst ant system (BWAS) [21], and others. In this chapter, we present the available ACO algorithms for combinatorial optimization from such a component-wise perspective. This perspective makes the contributions of each of the ACO algorithms more easily identifiable, which allows a more flexible approach toward their implementation. One additional contribution of this chapter is to provide such an implementation. The main advantage of this framework is that it enables the composition of new ACO variants. While one possible approach would be to manually explore different ACO algorithm designs and test their effectiveness, this exploration may also be done using automatic methods [65]. For illustration purposes, we demonstrate here how to find the best configuration of such a framework for the traveling salesman problem (TSP) and the quadratic assignment problem (QAP) using irace, an automatic algorithm configuration method [83].

This chapter is structured as follows. In section "Combinatorial Optimization Problems and Constructive Heuristics" we introduce some basic notions of combinatorial optimization and construction heuristics. Next, in section "The ACO Algorithmic Framework", we present the ACO metaheuristic as an algorithmic framework for single-objective combinatorial optimization problems, explaining ACO algorithms from a component-wise perspective. Section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP" describes how historical ACO algorithms are instantiations of this framework and how ACO algorithms can be automatically configured. Section "Applications of ACO to Other Problem Types" describes recent developments in ACO applied to problems with particular characteristics, including continuous optimization and mixed-variable, multi-objective, and dynamic problems. Section "ACO in Combination with Other Methods" reviews hybridizations of ACO with other methods, such as tree search

methods, exact methods, and surrogate models. We also briefly discuss recent experiments in dynamically changing the parameters of ACO algorithms during execution. Finally, section "Conclusions" concludes with some perspectives for future research in ACO.

Combinatorial Optimization Problems and Constructive Heuristics

Many problems of enormous practical and theoretical importance can be modeled as combinatorial optimization problems (COPs) [106]. Although practical problems usually have a large number of constraints to be satisfied and uncertainties or multiple objectives to be considered, even simpler COPs without such complicating factors may be very difficult to solve.

A COP can be defined as consisting of a set of instances [106]. An instance of a combinatorial optimization problem $(\mathscr{S}, \Omega, f)$ is defined by

- A search space $\mathscr{S}$ given by the possible value assignments to discrete decision variables x_i, with $i = 1, \ldots, n$;
- A set of constraints Ω among the decision variables;
- An objective function $f : \mathscr{S} \to \mathbb{R}$, which may have to be minimized or maximized.

The notion of *problem* refers to the general abstract task to be solved, while the *instance* is defined as a concrete instantiation of this task with fully specified data.

One of the most studied combinatorial optimization problems is the traveling salesman problem (TSP). In the TSP, a graph $G = (N, E)$ is given with $n = |N|$ nodes, a set E of edges fully connecting the nodes, and distances associated with the edges $d_{ij}, \forall (i, j) \in E$. The goal is to find a Hamiltonian tour of minimal length. Such a tour can be represented as a permutation $\pi = (\pi_1, \ldots, \pi_n)^t$ of the n nodes, where π_i is the node index at position i. Thus, $\mathscr{S}$ is the space of such permutations. The objective function in the TSP is

$$\min_{\pi \in \mathscr{S}} d_{\pi_n \pi_1} + \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{1}$$

It is important to note that the permutations in the TSP are cyclic, that is, for a TSP solution, the absolute position of a node in the permutation is not relevant.

Another well-known combinatorial problem is the quadratic assignment problem (QAP), where two matrices are given describing the distances between n locations and the flows (of persons, goods, etc.) between n facilities. The goal is to find an

assignment of facilities to locations that minimizes the sum of the products between distances and flows. More formally, given two $n \times n$ matrices $[d_{ij}]$ and $[f_{ij}]$ with $i, j = 1, \ldots, n$, a solution to the QAP is an assignment of facilities to locations (or, equivalently, of locations to facilities). Because the number of facilities is the same as the number of locations, such an assignment can be represented by a permutation π, where π_i is the facility assigned to location i. The objective function in the QAP is

$$\min_{\pi \in \mathscr{S}} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot f_{\pi_i \pi_j} \tag{2}$$

Solving a combinatorial optimization problem can often be described as choosing a subset from a finite set of *solution components* $\mathscr{C} = \{c_1, c_2, \ldots\}$ and determining their precise combination that results in a solution that satisfies the problem constraints, i.e., a *feasible* solution, and that optimizes the objective function, i.e., an optimal solution. (For what follows, we assume without loss of generality that the objective function is to be minimized.) In the TSP, for example, the solution components c_{ij} are typically taken to be the edges (i, j) of the given graph, whereas in the QAP, they are the possible individual assignments of a facility j to a location i. Commonly, problems are solved by either searching only in the feasible candidate solution space or, if this is deemed to be too difficult, by allowing the evaluation of infeasible candidate solutions but biasing the search in some way toward feasible ones. For both the TSP and the QAP, a search in the feasible space can be easily enforced, as any permutation is a feasible candidate solution.

Many COPs with practical applications belong to the class of NP-hard problems [49], and, hence, they are considered hard to solve. The NP-hardness of many of these problems, including the TSP and the QAP, implies that, in the worst case, the time needed to find the optimal solution for a given instance grows exponentially with instance size. Thus, instead of searching for an optimal solution and proving its optimality, which may require an infeasible amount of computation time, heuristic algorithms are used to generate good solutions within a reasonable time.

Perhaps the simplest heuristic methods are constructive heuristics. A constructive heuristic starts from an empty or partial candidate solution and then iteratively extends it by adding solution components to it. In many cases, a heuristic estimation of the quality of solution components is available and can be used to guide the choice of which solution components to add. Greedy constructive heuristics rank solution components according to their heuristic value and, at each construction step, they add the best-ranked solution component to the current partial solution. If more than one solution component has the best heuristic value, tiebreaking can be done either randomly or by a secondary heuristic.

An example of constructive heuristic for the TSP is the *nearest neighbor* heuristic. Starting from a random node i and an empty candidate solution $\pi = \langle\rangle$, it selects the solution component c_{ij}, $j \in N \setminus \{i\}$ with the smallest distance d_{ij} and adds it to π such that $\pi_1 = i$ and $\pi_2 = j$. The next solution component c_{jk} chosen

is the one that minimizes the distance $d_{jk}, k \in N \setminus \{i, j\}$, and so on until a complete permutation π of the nodes in N is obtained.

A constructive heuristic for the QAP would be guided by the fact that facilities with a high total flow to other facilities should be placed at locations that are central and, thus, have a small sum of distances to other locations. Thus, one may precompute values $f = (f_1, \ldots, f_n)^t$, where $f_i = \sum_{j=1}^{n} f_{ij}$, and analogously $d = (d_1, \ldots, d_n)^t$, where $d_k = \sum_{l=1}^{n} d_{kl}$, and then assign the facility with the largest f_i to the locations with the smallest d_k and iterate these steps.

If the heuristic values of solution components remain constant during the solution construction, the construction heuristic is called *static*. Otherwise, when the heuristic values are a function of the partial solution generated so far, one talks of an *adaptive* construction heuristic. In the QAP case, one may obtain an adaptive heuristic by using the intuition that facilities with a high interaction with already assigned ones should be put on a location that is as close as possible to already chosen locations. Adaptive heuristics generally require higher computation times as the heuristic values need to be updated or even recomputed after each construction step; however, making decisions dependent on a partial solution may lead to higher-quality complete candidate solutions.

Constructive heuristics are basic ingredients of many metaheuristics. As an example, semi-greedy heuristics [64] are a randomized form of constructive heuristics, i.e., they make randomized decisions during the constructive search, thus generating many different solutions. The GRASP metaheuristic [42, 43] extends upon semi-greedy heuristics by combining randomized adaptive construction heuristics with local search algorithms. Another example is iterated greedy algorithms [112], which consist in the repeated application of constructive heuristics that start from partial solutions. A destruction mechanism generates these partial solutions by removing solution components from a complete candidate solution.

The ACO Algorithmic Framework

ACO algorithms can also be seen as an extension of construction heuristics. The main characteristics that distinguish an ACO algorithm from other metaheuristics and, in particular, other constructive metaheuristics are the following: (i) it is a population-based algorithm, where m solutions are generated at each iteration, (ii) solutions are generated by a probabilistic constructive mechanism that is biased by numerical information called pheromones (and possibly by heuristic information), and (iii) the pheromones are updated during the run of the algorithm using the quality of generated solutions as feedback.

The general outline of an ACO algorithm is given in Fig. 1. After initializing data structures and parameters, an ACO algorithm generally iterates through two procedures: solution construction (procedure ConstructSolutions) and pheromone update (procedure UpdatePheromones). Additionally, a local search algorithm may be used to improve some or all the solutions constructed in the current iteration.

```
procedure ACO_Metaheuristic
    repeat
        for each ant do
            repeat
                ExtendPartialSolutionProbabilistically()
            until solution is complete
        for each ant ∈ SelectAntsForLocalSearch() do    // optional
            ApplyLocalSearch(ant)                       // optional
        EvaporatePheromones()
        DepositPheromones()
    until termination criteria met
end
```

Fig. 1 General pseudo-code of the ACO metaheuristic for NP-hard problems. First, each ant constructs a complete solution by probabilistically choosing solution components to extend their current partial solution (section "Solution Construction"). Some of these solutions are sometimes further improved by means of local search (section "Local Search"). Finally, pheromone values are updated according to the solutions constructed by the ants (section "Global Pheromone Update"). The update usually consist of two complementary steps: evaporation decreases pheromone values and deposit increases them

Clearly, each of these procedures may in turn consist of different phases and, depending on the particular ACO algorithm implemented, the particular choices taken within these phases may differ quite strongly. These differences, however, mainly concern specific building blocks or parameter settings that are used within the ACO algorithm phases. These building blocks and parameters are referred as algorithmic components.

In the following, we present the algorithmic components that have been proposed in typical ACO algorithms for combinatorial optimization and discuss alternative options proposed in the literature for implementing these components.

Choice of Pheromone Trails and Heuristic Information

In ACO, the solution construction is probabilistically biased by two types of information: pheromone trails and heuristic information.

The pheromone trails are a set $\mathscr{T}$ of numerical values associated, in principle, to all solution components, that is, $\forall c \in \mathscr{C}$, $\exists \tau_c \in \mathscr{T}$. Pheromone trails are updated during the run of an ACO algorithm according to the quality of previously generated solutions through the mechanisms of pheromone deposit and pheromone evaporation. During the solution construction procedure, pheromone trails bias the choice of solution components toward constructing high-quality solutions. A high value of τ_c represents a higher probability of adding solution component c to the solution being constructed. It is therefore important to appropriately define the set of

solution components $\mathscr{C}$, and the corresponding pheromone trails $\mathscr{T}$, when applying an ACO algorithm to a particular problem.

For many problems, the appropriate definition of solution component and, thus, of pheromone trail is rather straightforward. In the TSP, the adjacency information is relevant, that is, which node is the direct successor or predecessor to another one, and therefore a solution component $c_{ij} = (i, j)$ typically refers to the edge connecting nodes i and j. Thus, τ_{ij} represents the pheromone trail associated with the edge. This definition of solution component is also related to the choices made in the nearest neighbor heuristic, which underlies the construction mechanism used in almost all ACO applications to the TSP. Differently, in the QAP, the individual assignments of facilities to locations are of crucial importance. Therefore, a solution component c_{ij} denotes that facility j is assigned to location i in a candidate solution π, that is, $\pi_i = j$ and the pheromone trails τ_{ij} then represent the desirability of assigning facility j to location i. When constructing solutions for the QAP, the order in which facilities are chosen for assigning them to locations may also have an impact on the solution that is eventually constructed. Such an order may also be determined during solution construction by the ants through pheromone trails that indicate which facility should be chosen next for an assignment to a location [120]. Hence, one could have two types of pheromones, the first type τ_{ij} would refer to the desirability of assigning facility j to location i and the second one τ'_{kl} would refer to the desirability of generating an assignment for facility l directly after having assigned facility k. However, no effective implementation of such a second type of pheromone trails seems to be available.

For more complex problems than the TSP and the QAP, alternative definitions of solution components, and the corresponding pheromone trails, may have to be considered [101]. How to define solution components and their associated pheromone trails is itself an ongoing research topic [17, 99]. However, a basic guideline would be to consider effective greedy constructive heuristics and then define the pheromone trails to bias the most important decisions done by this heuristic. Using this guideline, standard definitions of solution components and pheromone trails will be available for many known problems; however, for more complex problems, some research effort may be required to determine the most appropriate solution components and pheromone trails.

The heuristic information $\mathscr{H}$ is also a set of numerical values associated with solution components ($\forall c \in \mathscr{C}, \exists \eta_c \in \mathscr{H}$). However, in contrast to the pheromone trails, heuristic information is not updated during the run of the algorithm in dependence of the quality of the solutions generated. Instead, η_c is a value that is either constant for each solution component, when static heuristic information is used, or a value that is a function of the partial solution generated so far, when adaptive heuristic information is used. The actual formulation of the heuristic information is specific to each problem. However, its computation should be fast as the value of η_c is used at every step of the solution construction by each ant. In some problems there is no readily available heuristic information that effectively guides solution construction, and thus, no heuristic information is used. This is, for example, the case in most ACO applications to the QAP.

Solution Construction

Solution construction is the process by which a new set of solutions is generated at each iteration of an ACO algorithms. In ACO terms, a solution is constructed by an ant; in optimization terms, each ant corresponds to the execution of a probabilistic solution construction procedure. In particular, each ant constructs one solution by starting from an empty solution $s = \langle\rangle$ and adding, at each construction step i, one solution component c_j from a set of candidate components N_i. The probabilistic choice at each step of the solution construction can be modeled by a probability distribution that assigns to each solution component the probability with which it is chosen. Thus, for each c_j there is a probability $Pr(c_j)$ of being chosen. This probability depends on the pheromone trails, the heuristic information, and the current partial candidate solution s, that is, $Pr(c_j|\mathscr{T}, \mathscr{H}, s)$. There are many different ways of computing this probability. Many ACO algorithms use the probabilistic rule that was introduced for ant system (AS) to choose one element from N_i as follows:

$$\Pr(c_j) = \frac{{\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}}{\sum_{c_k \in N_i} {\tau_k}^{\alpha} \cdot {\eta_k}^{\beta}} \quad \forall\, c_j \in N_i, \tag{3}$$

where α and β are parameters that control the relative importance of pheromone trails and heuristic information on the decision probabilities. If α tends to zero, the solution construction becomes a probabilistic multi-start greedy algorithm; if β tends to zero, the solution construction is biased only by the pheromone trails, and heuristic information is neglected. This rule assigns a probability in a fashion similar to the well-known roulette wheel selection of evolutionary algorithms [51], where the value ${\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}$ plays the role of the fitness assigned to a candidate. Clearly, many different ways can be used to define probability distributions for the solution components. For example, Maniezzo [84, 85] uses in his ANTS algorithm an additive way of combining pheromone trails and heuristic information:

$$\Pr(c_j) = \frac{\alpha \cdot \tau_j + (1-\alpha) \cdot \eta_j}{\sum_{c_k \in N_i} \alpha \cdot \tau_k + (1-\alpha) \cdot \eta_k} \quad \forall c_j \in N_i. \tag{4}$$

The above equation has the advantage over the most common one (Eq. 3) that the operations used (multiplication instead of exponentiation and sum instead of multiplication) are measurably faster in current CPUs. However, one needs to ensure that the range of values of the pheromone trails and of the heuristic information is similar to avoid undesired biases.

A method for making the construction more deterministic was proposed by Dorigo and Gambardella [34] with their ant colony system (ACS) algorithm. They introduced the *pseudorandom proportional rule*, which is controlled by a parameter q_0. At each construction step i, a random value q is drawn uniformly from $[0, 1)$; if

$q > q_0$, the probability of c_j is computed as in Eq. 3; otherwise (i.e., when $q \leq q_0$) the choice is made as

$$c_j = \arg\max_{c_k \in N_i} {\tau_k}^{\alpha} \cdot {\eta_k}^{\beta}, \tag{5}$$

that is, when $q \leq q_0$, the solution component with maximum probability is chosen deterministically. A larger value of q_0 is equivalent to a more "greedy" construction procedure, which usually results in faster convergence, whereas smaller values of q_0 lead to more varied solutions and, thus, to more exploration.

Solution construction is one of the critical components of ACO, thus techniques have been proposed to improve its efficacy and efficiency. Efficacy is sometimes improved by using *lookahead* [96], that is, by considering more than one component at a time when choosing which component should be added to the current partial solution. Efficiency can be greatly improved, particularly in large problem instances, by restricting the choice of solution components to *candidate lists* [33, 34]. Candidate lists contain the most promising solution components, and their definition depends on the instance data and/or on the current partial solution. The most efficient candidate lists depend only on the instance data and are typically computed at the start of the algorithm. One example of candidate lists is the nearest neighbor lists in the TSP, which store the *cl* nearest neighbors to each city ordered according to nondecreasing distances. Even a small value of *cl*, such as 20, is sufficient to generate very-high-quality (or even optimal) solutions to the TSP. Such candidate lists are crucial when applying ACO algorithms to large TSP instances.

Candidate lists may also save memory if the algorithm only stores the pheromone values of those solution components that are part of the candidate lists. Another approach to save memory used in some ACO variants is to avoid explicitly storing the pheromone trails of all solution components by using, for example, a hash table [3] to store only the pheromone values associated with solution components that have appeared in previouslyconstructed solutions. This might be beneficial when the fraction of pheromone values explicitly stored is small enough (relative to the total size of $\mathscr{T}$) to compensate for the fact that insertion, deletion and look-up in the hash table require more than constant time.

Independent of the usage of candidate lists, solution construction may be speeded up when using static heuristic information by precomputing the values of ${\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}$, $\forall c_j \in \mathscr{C}$, as all ants use the same total values in Eqs. 3 and 5, and each ant's partial solution only affects which precomputed values are considered at each construction step. If an adaptive heuristic information is used, the total values would depend on the partial solution of each ant; hence, they cannot be precomputed in this way.

Most ACO algorithms construct solutions starting from an empty solution. However, a few proposals have studied the possibility of starting from partially constructed solutions with the goal of partially destroying a very good solution and reconstructing from it a, hopefully better, new one. This is the same concept applied by iterated greedy [112]. Examples of this kind of ACO algorithm are ACO algorithms using external memory [1], iterated ants [133], and cunning ants [132].

Considering the effectiveness of the solution construction, these methods are useful as they avoid the most expensive construction steps that happen when solutions are empty or almost empty. In the Enhanced ACO [47], solution construction is guided by the global-best solution, the best solution found since the start of a run of the algorithm, by adopting the choice done for that solution with a fixed probability at each construction step. This approach somehow introduces a guided perturbation into the best solution found so far and, thus, a straightforward way of increasing the exploitation of good solutions in ACO. It has also the advantage of resulting in faster solution construction as no probabilistic choices among several candidate components need to be done; thus, it is deemed to be useful particularly for tackling large instance sizes.

Typically, construction rules take into account the pheromone trails associated with only single solution components. However, for specific problems it may be useful to include also the pheromone associated with other solution components. An example are scheduling problems, for which the *pheromone summation rule* was proposed [92]. Its goal is to avoid that jobs are scheduled in a position too far away from positions where they occur in high-quality solutions.

Global Pheromone Update

As mentioned above, the pheromone trails are modified during the run of an ACO algorithm in order to bias the construction of new solutions. Pheromone update usually consists of two complementary steps: pheromone evaporation and pheromone deposition. The general idea behind these two steps is to bias the pheromone trails so as to favor the construction of high-quality solutions. This general idea can be implemented in a number of different ways.

Pheromone evaporation decreases the pheromone values by some factor with the goal of reducing the effect of previous pheromone depositions and, in this way, help to forget previous poor decisions. Pheromone evaporation itself can be applied, in principle, to some or all the pheromone trails. It can be described as

$$\mathscr{T}_{\text{new}} = \text{evaporation}(\mathscr{T}_{\text{old}}, \rho, S^{\text{eva}}), \tag{6}$$

where $\rho \in (0, 1)$ is a parameter called evaporation rate and S^{eva} denotes the set of solutions that are selected for evaporating the pheromone trails. Most of the existing ACO algorithms do not make use of the solutions in S^{eva} to implement the pheromone evaporation and simply reduce the pheromone trail value of all solution components by the same factor:

$$\tau_j = (1 - \rho) \cdot \tau_j \qquad \forall \tau_j \in \mathscr{T}. \tag{7}$$

A value of $\rho = 1$ would mean that pheromone trails are reset at each iteration of the algorithm and, thus, no learning occurs. A value of $\rho = 0$ would mean that no evaporation occurs, which would result in an unlimited accumulation of

pheromones. For intermediate values of ρ, the amount of pheromone decreases geometrically as a function of this parameter.

In some ACO algorithms, the global pheromone evaporation only applies to some specific solution components. For example, in ACS pheromone evaporation affects only the solution for which pheromone is deposited in the current iteration:

$$\tau_j = (1 - \rho) \cdot \tau_j \qquad \forall \tau_j \mid c_j \in s^{\text{eva}}, \tag{8}$$

where $\forall \tau_j | c_j \in s^{\text{eva}}$ are all the pheromone trails associated with solution components that occur in the solution s^{eva} chosen for pheromone deposition (see also below). Equation 6 is also general enough to include the pheromone evaporation applied in population-based ACO [55], where the pheromone trails are defined as a function of the solution components occurring in a set of candidate solutions and where pheromone evaporation corresponds to the reduction of the amount of pheromone that is associated with components of the solutions that leave this set of candidate solutions.

Evaporation has the effect of slowing down the convergence of the algorithm as it reduces the probability of constructing again solutions previously constructed. However, while pheromone deposition selectively reinforces the pheromone trails associated with some solution components, pheromone evaporation has the effect of decreasing the probability of selecting those solution components less recently reinforced, thus allowing the algorithm to focus on the most recently found solutions and to "forget" previous pheromone depositions.

Pheromone deposition consists in increasing the pheromone values of a few selected solution components. These solution components belong to one or more solutions previously constructed by the ACO algorithm. In a general form, the pheromone deposition can be written as

$$\tau_j = \tau_j + \sum_{s_k \in S^{\text{upd}} | c_j \in s_k} w_k \cdot F(s_k), \tag{9}$$

where $S^{\text{upd}} \subseteq S^{\text{eva}}$ is the set of solutions chosen to deposit pheromones, w_k is a weight that is associated with solution $s_k \in S^{\text{upd}}$, and $F(s_k)$ is a function that is nondecreasing with respect to the quality of the solution s_k – that is, whenever $f(s_i) < f(s_l)$ in the minimization case, then it follows that $F(s_i) \geq F(s_l)$. The amount $w_k \cdot F(s_k)$ therefore corresponds to the amount of pheromone that is deposited by a solution s_k.

The solutions used for updating the pheromone trails (S^{upd}) have a strong influence in the behavior of an ACO algorithm. Ant system (AS) selects all solutions constructed in the latest iteration. In contrast, there are alternative methods that consider one single solution for the pheromone deposition, e.g., the iteration-best update (ib-update), which uses the best solution from those generated in the most recent algorithm iteration; the global-best update (gb-update), which uses the

best solution found since the start of the algorithm; and the restart-best update (rb-update), which uses the best solution found since the pheromones were reinitialized (see also below on pheromone reinitialization). Intermediate alternative methods would define a set of candidate solutions that may comprise a number of the best solutions of the latest algorithm iteration and solutions such as the global-best or the restart-best ones and use these to deposit pheromone. The particular set of solutions that is chosen to deposit pheromone has a direct effect on the speed of convergence and possible stagnation behavior of the algorithm, which occur when the pheromone trails have converged and the same solutions are constructed over and over again. The gb-update provides the fastest convergence but may more easily lead to such stagnation, while allowing all candidate solutions to deposit pheromones may delay the convergence of the algorithm [35].

In addition to the choice of solutions that form S^{upd}, the amount of pheromone deposited by these solutions also has a direct impact on the search behavior of ACO algorithms. A typical setting in ACO algorithms is to make the amount of pheromone deposited inversely proportional to the objective function value (in the minimization case), that is, to set $w_k \cdot F(s_k) = 1/f(s_k)$. This is the case, for example, for MMAS [125]. (Instead of making the amount of pheromone deposited a function of the quality of the solution, one may also deposit a constant amount of pheromone if the bias toward good solutions is ensured by choosing ib-update, gb-update, or similar biases toward the best candidate solutions.) Various other ACO algorithms add additional weighting factors that may be dependent or not on the quality of the solution relative to others. ACS uses a pheromone update by setting $w_k \cdot F(s_k) = \rho/f(s_k)$, where ρ is the evaporation rate. AS initially used a setting of $w_k \cdot F(s_k) = Q/f(s_k)$, where Q is a parameter; however, in many recent implementations, one simply uses $Q = 1$. Various ACO algorithms use an unequal weighting for the amount of pheromone deposited. For example, RAS [19] makes the weight dependent on the rank of a solution in the current algorithm iteration by choosing $w_k \cdot F(s_k) = \max\{0, w - r\}/f(s_k)$, where $w = |S^{\text{upd}}|$ is the number of solutions that deposit pheromone after each iteration (this parameter is called *rasrank* in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP") and r is a solutions' rank in the current iteration. The largest amount of pheromone ($w/f(s_k)$) in RAS is assigned to the global-best solution s_{gb}. This corresponds to a choice where S^{upd} comprises the global-best solution and the $w - 1$ best-quality solutions of the current iteration. In EAS [30, 36, 38], the usual AS deposition rule is followed, that is, all solutions generated in the current iteration deposit pheromone; in addition, the global-best solution deposits an amount of pheromone $w_{\text{gb}} \cdot F(s_{\text{gb}}) = Q \cdot m_{\text{elite}}/f(s_{\text{gb}})$, where Q is the same parameter as for AS and m_{elite} is the multiplicative factor that increases the weight of the global-best solution. In BWAS [21], not only the global-best solution deposits pheromone, but also further evaporation is applied, following Eq. 7, to the pheromone trails associated with the solution components that appear in the worst solution of the current iteration and that do not appear in the global-best one.

Pheromone Update Schedule

As said above, the choice of the solutions that deposit pheromone has a strong impact on the search behavior of ACO algorithms. The main goal of pheromone update schedules is to adapt the choice of the solutions that deposit pheromone during the ACO algorithm run. In the simplest case, this is done following a predefined schedule. The first algorithm to make such a predefined schedule explicit is MMAS in its application to the TSP [118,125] (related ideas are also discussed in [90]). In particular, when local search is applied, the central idea is to shift from a frequent use of the ib-update, which is used by default, toward an increasing frequency of the gb-update (or an rb-update). This shift has the effect of changing the search behavior from a more explorative one toward a more exploitative one that searches for new, improving solutions around the best-so-far ones. When used after the reinitialization of the pheromone trails (see section "Pheromone Reinitialization"), the schedule can also switch in the pheromone deposit between ib-update, rb-update, and gb-update. In particular, if the restart-best solution is used every iteration and no new restart-best solution is found for a number of iterations, the schedule switches to gb-update. As a result, the pheromone trails will implicitly interpolate between the restart-best solution and the global-best solution in a way that resembles the ideas underlying path relinking [50].

The update schedule may have a critical impact on the performance of an ACO algorithm, because it determines the balance between speed of convergence and exploration capabilities. It is also very sensitive to the termination criteria: an update schedule that performs well for long computation times may converge too slowly if the algorithm is terminated much earlier.

Initialization of Pheromones

Two alternative ways of initializing the pheromone trails have been proposed: either using a very small initial value (ACS and BWAS) or using a rather large value (MMAS), where small and large are relative to the amount of pheromone deposited in the global pheromone update. A small value results in a rather quick bias toward the best solutions, while a large value results in a more explorative initial search phase (though depending also on other parameters such as the evaporation rate). Neither the original AS (nor EAS and RAS) specify how the pheromones are initialized and leave it open as a parameter τ_0 to be specified by the user.

Pheromone Reinitialization

It has been shown that resetting the pheromone values back to their initial value may help in long runs by increasing the exploration of the search space [125]. This procedure is often called *restart*, since it is equivalent to restarting the run, although the information about the global-best solution found is kept. While most

ACO algorithms do not implement such restarts, restarting has been shown to be very effective for problems such as the TSP and the QAP where the use of strong local search algorithms leads to fast convergence.

MMAS was the first ACO algorithm to employ pheromone reinitialization to avoid stagnation. In particular, MMAS computes a measure, called branching factor, of the potential alternatives encoded in the pheromone trails. When the branching factor goes under a certain threshold value (close to 1), pheromone values are reset to their maximum value ($\tau_{\max}$).

Another ACO algorithm that uses pheromone reinitialization is best-worst ant system (BWAS) [21], which reinitializes the pheromone values to τ_0 whenever the distance, in the decision space, between the global-best solution and the worst solution found in the last iteration falls under a threshold value.

In order to avoid very frequent restarts, there is often a "grace period" when no restarts are allowed; for example, a certain number of iterations since the last restart-best solution was found.

Local Pheromone Update

Local pheromone update is a mechanism that works during solution construction. It modifies the amount of pheromone associated with solution components that have just been chosen by the ants. The first such mechanisms were studied during the design of AS in the so-called ant-density and the ant-quantity models [20, 30, 37] but were abandoned due to their poor performance. In later research, local pheromone update to diminish the pheromone trails associated with chosen solution components was explored in the design of the ACS algorithm [34]. In particular, ACS uses local pheromone update following

$$\tau_j = (1 - \xi) \cdot \tau_j + \xi \cdot \tau_0 \qquad \forall \tau_j \mid c_j, \tag{10}$$

where c_j is the solution component just selected by an ant during solution construction, $\xi \in (0, 1)$ is a parameter controlling the strength of the local update, and τ_0 is the initial pheromone value, which is set to a very small value, much lower than the expected amount of pheromone deposited in the global pheromone update. The effect of this update rule in ACS is to increase the search exploration during construction making used solution components less attractive.

Although the local pheromone update is very effective when combined with the other characteristics of ACS, it is difficult to incorporate it on its own to other ACO algorithms, because it relies on a strong gb-update and a lack of pheromone evaporation.

An important implementation detail is whether the solutions are constructed in *parallel*, that is, each ant chooses one solution component at a time, or *sequentially*, that is, each ant completes its own solution before the next one starts constructing their own. The different construction schemes may introduce differences in the effect of the local pheromone update.

Pheromone Limits

MMAS introduced the concept of explicit pheromone limits that restrict the possible values the pheromone strength can take to the range $[\tau_{\min}, \tau_{\max}]$. The goal is to prevent stagnation, which occurs when the pheromone values of some solution components are so low that there is no possibility the component will ever be selected (or alternatively, the pheromone values of some solution components are so high that there is no possibility that any other component will ever be selected).

The original proposal showed how to adapt the limits within a run of MMAS. In practice, however, the upper pheromone limit seems to be less important as the maximum amount of pheromone possible on any solution component is already limited by pheromone evaporation.

The definition of $\tau_{\min}$ and $\tau_{\max}$ is problem specific. In general, $\tau_{\max}$ should be an estimation of the pheromone added by the optimal solution. In the TSP, for instance, it corresponds to $\tau_{\max} = \frac{1}{\rho \cdot f(s_{\text{opt}})}$; however, since the optimal solution s_{opt} is not known, the global-best solution s_{gb} is used instead.

The main parameter controlling the pheromone limits is p_{dec}, which is the probability that an ant chooses exactly the sequence of solution components that reproduces the best solution found so far. The value $\tau_{\min}$ is given by

$$\tau_{\min} = \frac{\tau_{\max} \cdot (1 - \sqrt[n]{p_{\text{dec}}})}{n' \cdot \sqrt[n]{p_{\text{dec}}}} \tag{11}$$

where n' is an estimation of the number of solution components available to each ant at each construction step. This value often corresponds to $n/2$ (or to half the size of the candidate lists if they are used). Nonetheless, in some implementations of MMAS, and specially when local search is used to improve the solutions constructed by the ants, the above computation may be simplified to $\tau_{\min} = \frac{\tau_{\max}}{2 \cdot n}$.

ACS also implicitly uses pheromone trail limits. Due to the way its local pheromone update rule is defined, the pheromone trails can never fall under the value τ_0, and they cannot grow above $1/F(s_{\text{gb}})$, thus limiting implicitly the pheromone trails to the range $[\tau_0, 1/F(s_{\text{gb}})]$.

Local Search

In many combinatorial problems, the use of local search algorithms is essential for obtaining competitive results. Local search algorithms start from an initial solution and iteratively apply small changes to it, as defined by a *neighborhood operator*, in order to find an improved solution. Two of the most basic local search algorithms are best improvement and first improvement, which replace the current solution with the best- and first-improving solution found in the neighborhood, respectively. These algorithms stop if, after examining the whole neighborhood of the current solution, no improved solution can be found, thus stopping at a local optimum. Other local

search algorithms that do not necessarily stop at a local optima, such as tabu search or simulated annealing, have also been combined with ACO algorithms [82, 120].

When combined with an ACO algorithm, local search is typically applied to the solutions constructed by the ants. The decision about which ant applies local search to its solution depends on the effectiveness and efficiency of the specific local search [47]. If the local search is relatively fast and effective, it may be worth applying it to all ants' solutions. Otherwise, restricting it either to the iteration-best ant or to a candidate set of promising solutions may save computation time, while still giving a chance to further improve the best solution found by the ants.

ACO Algorithms as Instantiations of the ACO Metaheuristic

Given the algorithmic components discussed above, then ACO algorithms from the literature can be described as combinations of specific components. (This list includes only a small subset of the ACO algorithms and algorithmic components proposed in the literature. A more comprehensive component view of ACO is left for further work.)

In particular, AS uses Eq. 3 for solution construction for all ants, Eq. 7 for pheromone evaporation, and Eq. 9 for the pheromone deposition with all ants using the same weight $w_1 = \cdots = w_m$, where m is the number of ants. EAS, RAS, BWAS, and MMAS all use the same solution construction rules and the same evaporation mechanism as AS, but they differ from AS in the way pheromone is deposited; also, they add new algorithmic components. In particular, the pheromone deposition of EAS uses the global-best solutions with a weight m_{elite}. Similarly, the only difference between AS and RAS is the way pheromone is deposited, which also uses the global-best solution and weights defined according to a parameter called *rasrank*, already explained in section "Global Pheromone Update". The pheromone deposition of BWAS also uses the global-best solution and, in addition, applies further evaporation to pheromone trails associated with solution components that appear in the worst solution of each iteration and not in the global-best one. In addition, BWAS performs pheromone reinitialization depending on the average distance between the iteration-worst solution and the global-best one. MMAS adds several features: it enforces minimum and maximum limits to pheromone values and uses an update schedule to switch between iteration-best, global-best, and restart-best deposition and pheromone reinitialization according to branching factor, and pheromones are initialized to a high initial value. MMAS variants that use the pseudorandom proportional rule (Eq. 5) have also been considered in the literature [124]. Finally, ACS is the ACO algorithm that structurally differs the most from AS: it adds the pseudorandom proportional rule to AS solution construction, pheromone is deposited using the global-best solution, and evaporation is performed at the same time on those values updated during pheromone deposition (Eq. 8). Moreover, further evaporation is performed during solution construction (local pheromone update, section "Local Pheromone Update") by each ant.

ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP

We present in this section a software framework that unifies the implementation of several ACO algorithms found in the literature. The code is publicly available at http://iridia.ulb.ac.be/aco-tsp-qap/. The implementation aims at generality, by clearly separating the general components of the ACO metaheuristic from the problem-specific components that vary from one problem to another. In this chapter, we present an implementation for the TSP and the QAP. However, it is possible to extend the framework to other combinatorial optimization problems with far less effort than an implementation from scratch.

In its current form, the software implements AS, MMAS, EAS, RAS, ACS, and BWAS. In addition to the usual ACO parameters (α, β, ρ, m, ξ, etc), it also allows combining various algorithmic components by setting specific parameters. In particular:

- Pheromone limits as defined by MMAS can be enabled or disabled independently of the algorithm chosen (except for ACS).
- Pheromone reinitialization (restart) can also be enabled or disabled independently of the algorithm. In particular, when more than res_{it} iterations have passed since the last restart-best solution was found, a restart condition is tested. This restart condition may be the branching factor going under a threshold value (res_{bf}), as in MMAS, or the distance between the global-best and the iteration-worst ants going under a threshold value (res_{dist}). In addition, a setting of *always* (*never*) means that the restart condition is always (never) satisfied.
- Any algorithm may use the pseudorandom proportional rule of ACS (Eq. 5) by simply using a value $q_0 > 0$.

In the problem-specific part, the implementation for the TSP follows what is provided by the ACOTSP software [119], that is, heuristic information, candidate lists, and various types of local search, including 3-opt first improvement with don't-look-back bits (*dlb-bits*) and nearest neighborhood lists (*nnls*). The part corresponding to the QAP does not implement heuristic information nor candidate lists, since these techniques have not proved useful for the QAP. It does implement, however, various local search algorithms, such as the fast 2-exchange best- and first-improvement local searches proposed by Taillard [128] and the robust tabu search proposed in the same paper.

The update schedule (section "Pheromone Update Schedule") is implemented as specific to each problem and ACO algorithm; however, it would be straightforward to extend the update schedule to all ACO algorithms. In the case when MMAS is used together with local search, the user can control the frequency with which the restart-best (or global-best) solution, instead of the iteration-best one, is used to update the pheromone trails, by means of a parameter called *schedule length* (*slen*). Higher values mean longer emphasis on the iteration-best solution, thus possibly increasing exploration at the expense of faster convergence.

Table 1 Default parameter settings for each ACO algorithm from the literature. Parameters that are available only for particular ACO algorithms are described under the table. The default parameter configuration used in the experiments is based on MMAS (see Table 2)

	m	ρ	q_0	*ph-limits*	*Restart*	Restart parameters
	TSP / QAP	TSP / QAP				
AS, EAS, RAS	25 / 5	0.5 / 0.2	0.0	No	Never	
BWAS	25 / 5	0.5 / 0.2	0.0	No	Distance	$res_{\text{dist}} = 0.05$
MMAS	25 / 5	0.2 / 0.2	0.0	Yes	Branch-factor	$res_{\text{bf}} = \begin{cases} 1.00001 & \text{(TSP)} \\ 1.1 & \text{(QAP)} \end{cases}$
ACS	10 / 5	0.1 / 0.2	0.98	No	Never	

$m_{\text{elite}} = n$ when using EAS, *rasranks* $= 6$ when using RAS, $\xi = 0.1$ when using ACS, *slen* $= 250$ (TSP) or 20 (QAP) when using MMAS

In the two following sections, we show how to find a good performing ACO algorithm for a specific class of instances of the TSP and the QAP by automatically configuring the proposed ACO framework. In particular, there is, for each problem, a set of training instances and another set of testing instances. The automatic configuration tool irace [83] is used here to find a parameter configuration given the set of training instances. Finally, this parameter configuration is compared with the default configuration for each problem by evaluating both configurations on the set of test instances.

The implementation of the proposed ACO framework assigns default values, consistent with the literature, to certain parameters depending on the particular ACO algorithm chosen. These default values are documented in Table 1 for completeness. For the purpose of automatic configuration, we consider a single default configuration based on MMAS. Table 2 summarizes, for each problem, the parameters of the proposed ACO framework, their domain, and default values considered in the automatic configuration experiments described below. Although some parameters are still exclusive to a particular ACO algorithm (e.g., ξ can only be used together with ACS), the framework allows using components from one ACO algorithm in others (e.g., the restart mechanism of BWAS can be enabled for any algorithm by setting parameter *restart* to the value "*distance*").

Finding a Better ACO Configuration for the TSP

In the case of the TSP, we consider random Euclidean TSP instances, in particular 50 instances of each size $\{1000, 1500, 2000, 2500, 3000\}$ for training and other 50 instances of each size for testing. A single run of irace has a maximum budget of 25,000 runs of the ACO software, and each run is stopped after 60 CPU seconds. Since the goal is to investigate whether it is possible at all to find a better ACO algorithm than the default one from the literature and in order to speed up the process, irace starts from the default configuration for the TSP (Table 2) as an initial solution. After irace finishes, the configuration found is evaluated on the test

Table 2 Domains and default values of the parameter settings of the ACO framework considered in the automatic configuration experiments for the TSP and the QAP. A value of n/a means that the parameter does not exist for the corresponding problem. A value of "–" means that the parameter has no value in the default configuration (because it depends on another parameter setting that is not enabled by default). The list of parameters enabled only for certain values of other parameters is given under the table

	TSP		QAP	
Parameter	Domain	Default	Domain	Default
algorithm	{AS, EAS, RAS, ACS, MMAS, BWAS}	MMAS	{AS, EAS, RAS, ACS, MMAS, BWAS}	MMAS
m	$[1, 500]$	25	$[1, 10]$	5
α	$(0.0, 5.0)$	1.0	$(0.0, 5.0)$	1.0.
β	$(0.0, 10.0)$	2.0	n/a	n/a
ρ	$(0.01, 1.0)$	0.2	$(0.01, 1.0)$	0.2
q_0	$(0.0, 1.0)$	0.0	$(0.0, 1.0)$	0.0
cl	$[5, 50]$	20	n/a	n/a
ξ	$(0.01, 1.0)$	–	$(0.01, 1.0)$	–
rasrank	$[1, 100]$	–	$[1, 100]$	–
m_{elite}	$[1, 750]$	–	$[1, 750]$	–
p_{dec}	$(0.001, 0.5)$	–	$(0.001, 1)$	0.005
ph-limits	{Yes, no}	Yes	{Yes, no}	Yes
slen	$[20, 500]$	250	$[5, 250]$	20
restart	{Never, branch-factor, distance, always}	Branch-factor	{Never, branch-factor, distance, always}	Branch-factor
res_{bf}	$(1.0, 2.0)$	1.00001	$(1.0, 2.0)$	1.1
res_{dist}	$(0.01, 0.1)$	–	$(0.01, 0.1)$	–
res_{it}	$[1, 500]$	250	$[1, 50]$	5
localsearch	{None, 2-opt, $2.5-opt, 3-opt$}	3-opt	{None, best-2-opt, short-tabu-search, long-tabu-search}	Best-2-opt
dlb-bits	{Yes, no}	Yes	{Yes, no}	No
nnls	$[5, 50]$	20	n/a	n/a
rasrank	When $algo =$ RAS			
m_{elite}	When $algo =$ EAS			
ξ	When $algo =$ ACS			
p_{dec}	when *ph-limits* = yes (and for the TSP only if also *restart* = never)			
ph-limits	When $algo \neq$ ACS			
slen	When $algo =$ MMAS			
res_{bf}	When *restart* = branch-factor			
res_{dist}	When *restart* = distance			
res_{it}	When *restart* ≠ never			
dlb-bits, *nnls*	When $localsearch \neq none$			

Table 3 Best configurations found by irace for the TSP

algo	*m*	α	β	ρ	q_0	ξ	*cl*	*nnls*	*ph-limits*	*slen*	*restart*	res_{it}
ACS	28	3.07	5.09	0.32	0.53	0.21	22	9	–	–	Branch-factor ($res_{\text{bf}} = 1.74$)	212
MMAS	40	0.94	4.11	0.72	0.14	–	18	12	Yes	191	Branch-factor ($res_{\text{bf}} = 1.91$)	367

Common parameters: *localsearch* = 3-opt + dlb-bits

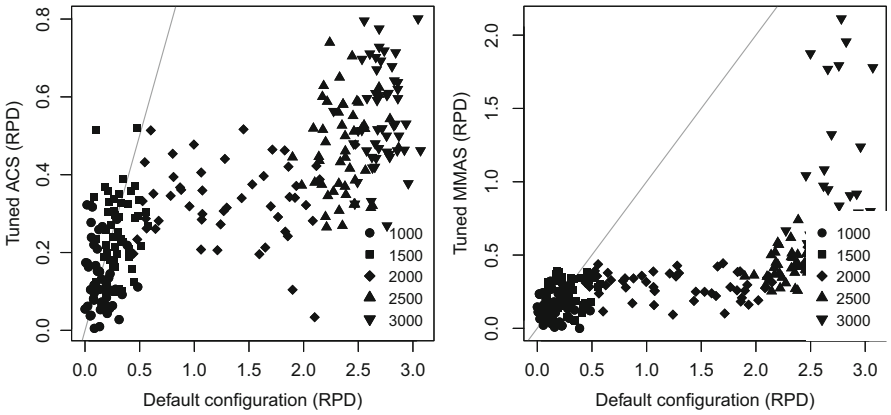


Fig. 2 Comparison of tuned vs. default configurations of ACOTSP on the test TSP instances. Tuned ACS configuration (*left*) and tuned MMAS configuration (*right*). Each point gives the relative percentage deviation from the optimal solution on one particular instance

instances. In particular, we select two configurations found by irace that improve over the default (see Table 3): one is a tuned variant of MMAS and the other is a tuned variant of ACS. These two configurations use a higher value of β and a stronger evaporation ρ than the default configuration. Since $q_0 > 0$, this MMAS variant uses the pseudorandom proportional rule from ACS, which is a MMAS variant that has also been explored previously in the literature [124]. It is also interesting that the restart strategy is typically much more aggressive than the default, with a much larger threshold branching factor, which results in more frequent restarts, only limited by the grace period (res_{it}). Other parameters, such as the size of the candidate list (*cl*), remain mostly unchanged from the default configuration.

Figure 2 compares the results obtained by these two tuned configurations with those obtained using the default configuration on the test instances. We ran each configuration once on each test instance up to 60 CPU seconds, and we computed the relative percentage deviation (RPD), with respect to the optimal solution, of the best solution found throughout the run. Both plots show that the solutions obtained

Table 4 Results on the test TSP instances obtained by the default configuration and two tuned configurations found by irace. Objective function values are given as relative percentage deviation from the optimal solution. ΔCI denotes the 95% confidence interval around the mean difference between the default configuration minus the configuration in that column (thus, positive values would denote that the default configuration is worse)

	Default	Tuned ACS	Tuned MMAS
mean	1.37	0.35	0.38
sd	1.08	0.18	0.32
ΔCI		[0.904, 1.14]	[0.875, 1.1]

by the tuned configurations are, in most instances, better than those obtained by the default configuration, and the differences are specially large for the largest instances. These observations are further confirmed by comparing the mean and standard deviation of the values obtained by each configuration (Table 4). In particular, a 95% confidence interval around the mean difference between the default and the tuned configurations does not contain the value zero, which indicates that the observed differences are statistically significant.

Finding a Better ACO Configuration for the QAP

In the case of the QAP, we consider two different instance sets: RR, where the distance matrix entries correspond to the Euclidean distances between points randomly generated in a square of side length 300 and the flow matrices are randomly generated according to a uniform distribution, and RS, where the distance matrix is generated as in the previous case and the flow matrix shows a structure similar to that found in real-world QAP instances. Within each set, we consider 100 instances, and we use half of them for training and the other half for testing the ACO configurations generated. For tuning the ACO framework, we consider a similar setup as in the TSP case, that is, each run of irace has a maximum budget of 25,000 runs of the ACO software, and each run stops after 60 CPU seconds.

For each of the two training sets, RR and RS, Table 5 reports the best configuration found in three independent runs of irace. These two configurations are fairly similar but quite different from the default one. First of all, it is somehow surprising that none of them uses pheromone limits, despite this being one of the key characteristics of MMAS. This could be due to the stronger restart strategies, which do not allow pheromone values to reach the limits and, thus, enforcing the limits adds an overhead that never pays off. Also, the schedule-length (*slen*) value is more than five times higher than the default, which implies a much higher exploitation of the iteration-best solution.

The two configurations of Table 5 are run on each test instance for 60 CPU seconds. Since for these instances the optimal objective values are not known, the relative percentage deviation (RPD) is computed with respect to a reference

Table 5 Best configurations found by irace for the QAP

	algo	*m*	α	ρ	q_0	*dlb-bits*	*ph-limits*	*slen*	*restart*	res_{it}
RR	MMAS	6	0.324	0.29	0.062	Yes	No	153	Distance ($res_{\text{dist}} = 0.051$)	22
RS	MMAS	4	0.164	0.342	0.284	Yes	No	170	Branch-factor ($res_{\text{bf}} = 1.822$)	40

Common parameters: *localsearch* = best-2-opt

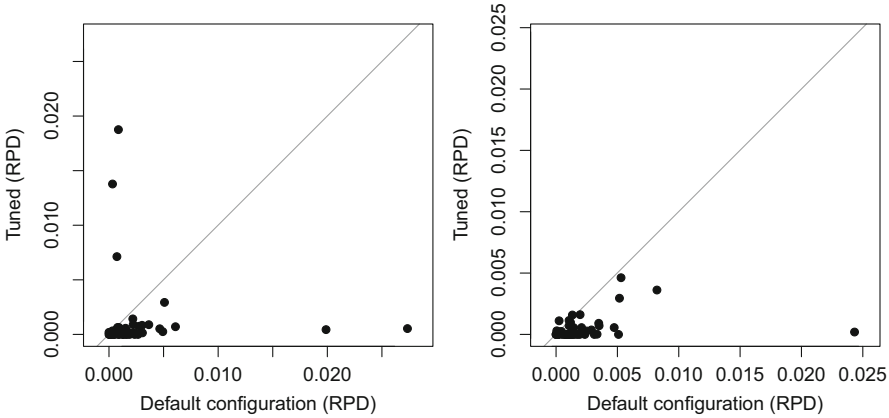


Fig. 3 Comparison of tuned vs. default configurations on the QAP test instances. RR instances (*left*) and RS instances (*right*). Each point gives the relative percentage deviation from the optimal solution on one particular instance

solution, for each instance, obtained by running the default configuration ten times with a time limit of 600 CPU seconds and keeping the best solution found.

Figure 3 compares the above configurations with the default on the test instances. The improvements of the tuned configurations over the default are not so clear for the QAP as they were for the TSP. In general, most RPD values are extremely small, which indicates that the variations over the best-known solutions are very small for all configurations. In both cases, there are a few instances where the default configuration obtains better RPD values than the tuned one. The statistical analysis in Table 6 shows that the mean difference between the default and tuned configuration for the RS instances is very small but still statistically significant (since the confidence interval does not contain zero), whereas for the RR instances, the null hypothesis of zero difference cannot be rejected (since the corresponding CI contains the value zero). The latter result is likely caused by the relatively large RPD values obtained by the tuned configuration on three particular instances, as shown on the left side of Fig. 3. These results could provide the motivation for exposing additional algorithmic components to automatic configuration, such as the precise update schedule of MMAS, which is known to be very sensitive to

Table 6 Results on the QAP test instances obtained by the default configuration and the tuned configurations found by irace. CI denotes the 95% confidence interval around the mean difference between the default configuration minus the configuration in that column (thus, positive values would denote that the default configuration is worse)

RR instances		
	Default	Tuned ACO
Mean	0.00241	0.00109
sd	0.00466	0.00335
ΔCI		[−0.000352, 0.00299]

RS instances		
	Default	Tuned ACO
Mean	0.00210	0.000485
sd	0.00364	0.000941
ΔCI		[0.00062, 0.00261]

the instance characteristics [125]. Extending the proposed framework with new algorithmic components from other ACO algorithms may also further improve the results on the QAP.

Applications of ACO to Other Problem Types

Although the ACO metaheuristic was primarily designed for tackling single-objective combinatorial optimization problems, its main ideas have been extended to tackle other types of problems. In this section, we provide an overview of such problems and how ACO has been adapted to deal with their particular characteristics. Comprehensive reviews of ACO applications are available in the literature [35, 126].

Continuous Optimization Problems

In continuous optimization problems, the domain of the decision variables is the set of real numbers or a subset thereof. In some problems, the simplest approach is to discretize the real-valued domain, which would allow the direct application of ACO. This was, for example, the approach followed by [70] when applying ACO to the protein–ligand docking problem, where a discrete ACO algorithm was combined with a local search for continuous domains.

In many problems, this discretization approach is not feasible, and the problem must be tackled in the continuous domain. A first group of algorithms for continuous optimization is inspired by the behavior of some ant species [14, 39, 97]. However, these algorithms diverge from the basic framework of the ACO metaheuristic, for example, by requiring the direct communication among ants instead of a pheromone structure. Therefore, they should rather be considered a separate class of algorithms.

A second group of algorithms directly translates the ideas underlying the ACO metaheuristic to the continuous domain. For example, Socha and Dorigo [116] replace the discrete pheromone distributions by Gaussian kernel functions that take the role of the pheromone model that is updated by the ants. Other similar approaches are found in [114, 131]. More recently, Liao et al. [74, 76]

proposed a unified model of various ACO algorithms for continuous optimization problems. Their model allows the automatic configuration of new ACO algorithms for continuous optimization very much in the line of what is presented in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP". They show that their final configured continuous ACO algorithms are competitive to other state-of-the-art algorithms for continuous optimization.

These approaches have also been extended to mixed-variable (continuous and discrete) problems [75, 115], by using appropriate ways of representing pheromone trails when handling discrete and continuous variables.

Multi-objective Problems

In many real-world problems, a solution is evaluated according to multiple, conflicting criteria (objectives). If there is a priori information about the importance of each objective, the objectives can be aggregated according to a preference model, and solutions are compared in this way. In the absence of such preference model, one can only say that a solution is better than another if the former is not worse in all objectives and better in at least one of them. Thus, the goal becomes to find (an approximation of) the Pareto set, i.e., the set of solutions not dominated by any other feasible solution [40, 117].

There are several proposals in the literature on how to apply the ACO metaheuristic to multi-objective combinatorial optimization problems. A few of these proposals assume an order of importance among the objectives [46]; however, most proposals attempt to approximate the Pareto set [5, 48, 80, 113]. As pointed out by López-Ibáñez and Stützle [79], many of these multi-objective ACO (MOACO) algorithms share similar algorithmic components combined in different ways. For example, Pareto ACO [27], BicriterionAnt [67], and COMPETants [26] use a different pheromone set $\mathscr{T}$ (a matrix, in their case) per objective, whereas MACS [10] and mACO-3 [2] use a single $\mathscr{T}$. Some papers have specifically compared both design choices [2, 82]. Moreover, there are basically two ways of updating the pheromone trails in MOACOs: either selecting the best solution for each objective, as done in COMPETants, Pareto ACO, and mACO-4 [2], or selecting (a subset of) all nondominated solutions, as done in BicriterionAnt, MACS, and mACO-3. Finally, a few MOACO algorithms make use of multiple colonies, where a colony is understood as a group of ants that construct solutions only according to the pheromone trails associated with their colony. The pheromone trails of each colony may correspond to multiple pheromone matrices (which are aggregated during solution construction), and colonies may exchange solutions [67]. Nonetheless, it is straightforward to define multi-colony variants of most MOACO algorithms [79].

Most of the MOACO algorithms were proposed for bi-objective problems. In the bi-objective case, the results presented in the literature indicate that the use of one pheromone matrix per objective and of multiple colonies, each of them with their own pheromone matrices, is necessary for MOACO algorithms to perform well on different regions of the objective space [48, 79, 80]. When applied to problems

with more than two objectives, this approach quickly becomes too computationally expensive. Notable exceptions are MOACO algorithms based on population-based ACO [4, 56]. However, a thorough comparison with other MOACO approaches on problems with more than two objectives is missing in the literature.

Dynamic Problems

Dynamic problems are those in which some information about the problem changes or becomes available after the start of the algorithm. This is the typical case in network routing, where ACO algorithms specifically designed for these problems have achieved notable results [23, 24, 130]. These algorithms differ significantly from the classical ACO algorithms for combinatorial optimization problems: ants are launched asynchronously in a distributed way, and no global pheromone update takes place. However, there are also in the literature dynamic variants of classical combinatorial problems, such as the dynamic TSP and the dynamic vehicle routing problem (VRP), where cities may appear or disappear and the distances between them may change. ACO algorithms for the dynamic TSP [41, 53], the dynamic QAP [54], and the dynamic VRP [29, 98] follow more closely the general outline of ACO algorithms discussed in section "The ACO Algorithmic Framework". In addition, they use specific routines to modify the pheromone trails after detecting changes in the problem data or structure. A real-world application of ACO to the scheduling of hospital nurses in Austria is described by Gutjahr and Rauner [63]. More recently, Lissovoi and Witt [77] formally analyze which type of changes in the dynamic shortest path can be tracked by a constant number of ants using a simplified MMAS variant. A recent overview of ACO algorithms for dynamic optimization problems is given by Leguizamón and Alba [73].

Stochastic Problems

In some optimization problems, either the objective function, the decision variables, or the constraints are not deterministic but subject to uncertainty or noise, and they are specified only within certain bounds or in terms of a probability distribution. The first stochastic problem to which ACO algorithms have been applied is the probabilistic TSP (PTSP), where one is given for each city a probability that it requires a visit. The goal in the PTSP is to find a tour of minimal expected length over all the cities. The first ACO algorithm for the PTSP was proposed by Bianchi et al. [11], who were using ACS. This algorithm was later improved upon by Guntsch and Branke [52] and further by Balaprakash et al. [7]. The ACO algorithms developed in that latter paper were then shown to match or even surpass in some cases state-of-the-art performance for the PTSP [8], and extensions thereof are state-of-the-art for the vehicle routing problem with stochastic customers and demands [9]. Another early ACO proposal for problems under uncertainty is the S-ACO algorithm [59], which has been the basis of later applications, for example, to the selection of optimal screening policies for diabetic retinopathy [18]. The S-ACO algorithm was later extended by Birattari et al. [15] who integrated

F-Race for determining the global-best solution in an ACO algorithm for stochastic optimization and have shown positive results for this latter integration. Another notable example is the application of ACO to the VRP with stochastic demands [12]. A survey of various metaheuristics, including ACO, for stochastic combinatorial optimization problems is provided by Bianchi et al. [13].

ACO in Combination with Other Methods

In this section, we briefly overview research on combining ACO with other methods.

ACO and Tree Search Methods

Ants in ACO perform a probabilistic solution construction that can be seen as a stochastic exploration of a search tree. Thus, a natural extension of ACO is to make use of tree search techniques such as branch-and-bound. An example is the approximate nondeterministic tree search (ANTS) algorithm by Maniezzo [84, 85], which uses lower bounds in three different ways. First, it incorporates a lower bound estimate as the heuristic information. Second, it prunes feasible solution components during construction if the estimated solution cost is larger than a threshold (e.g., the cost of the best solution found so far). Third, the order of assignment of solution components during solution construction is influenced by lower bound computations.

A different alternative is to integrate concepts from ACO into tree search methods. An example of this approach is Beam-ACO [16], which incorporates the use of pheromone trails into beam search. Beam search is an incomplete tree search method that keeps a set of partial solutions (the beam) and, at each iteration, selects a number of potential extensions of each of them by adding an additional solution component. These potential extensions are selected based on heuristics [105]. The number of extensions is typically larger than the size of the beam; thus, only the best extensions, according to a lower bound estimate, are kept for the following iteration. Beam-ACO executes beam search several times, replacing the deterministic solution extension of beam search with the probabilistic construction of ACO. In this way, the extension of partial solutions is biased by pheromone trails that are updated with the best solutions found in previous executions of beam search [16]. If lower bound estimates are not reliable or computationally feasible for the problem at hand, an alternative is to perform *stochastic sampling* [78], that is, to complete partial solutions by means of the construction procedure of ACO and use the cost of the complete solution (or the best of several samples) as the cost estimate of the partial solution.

ACO and Exact Methods

The combination of heuristic and exact methods, sometimes called *matheuristics*, is a promising trend in optimization. Roughly speaking there are two types of matheuristics: those that use exact methods to solve simpler versions of a problem in

order to improve the solutions generated by a heuristic or provide better estimates of solution quality and those that use heuristic methods to provide initial solutions or to constrain the search space of exact methods in order to make their application feasible to large problems. (The exploitation of tree search techniques such as branch-and-bound or beam search discussed before, can actually also be seen as such a matheuristic approach.)

An example of the first type of matheuristic is the use of constraint programming (CP) techniques [86] to help ACO focus on feasible solutions. This is particularly useful in highly constrained problems, such as scheduling or timetabling, where a major challenge is to find feasible solutions among many infeasible ones. Classical ACO algorithms do not perform satisfactorily on such overly constrained problems. Meyer and Ernst [95] use CP techniques to identify in the ants' construction procedure whether specific solution components lead to infeasible solutions. An example of the second type of matheuristic is the work by Massen et al. [88, 89]. Their pheromone-based column generation algorithm uses an ACO algorithm to generate a heuristic set of feasible routes, from which an optimal subset that is a solution to a vehicle routing problem is selected by an exact solver. The combined algorithm is still a heuristic, because the exact solver does not consider all possible routes, but it allows applying the exact solver to problems with many feasible routes and black-box constraints.

ACO and Surrogate Models

When the evaluation of candidate solutions is very costly in terms of computation time, usually only a small number of candidates can actually be evaluated. Costly solution evaluation arises in the case of simulation-based optimization [6] and, frequently, in industrial settings [44, 129]. Applications with very small evaluation budgets have been rarely studied in the ACO literature, and preliminary results suggest that typical ACO parameter settings are not appropriate in such context [109]. In cases with very small evaluation budget, also the use of surrogate models during optimization may prove useful [68, 100, 134]. Surrogate modeling approaches build prediction models that estimate the quality of candidate solutions, such that only the most promising candidate solutions are actually evaluated. A first approach combining surrogate models and ACO algorithms for tackling combinatorial optimization problems was studied by Pérez Cáceres et al. [109]; further research is needed to generalize the results to other problems.

Parameter Adaptation

Good parameter settings may not only be found "offline," as done in section "ACOT-SP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP", but modified or adapted "online" while the algorithm is running. Good parameter settings may also be predicted according to instance features [102]. In the context of ACO, several parameters have a strong effect on the balance between search space

exploration and exploitation of the best found solutions, and their proper values may additionally depend strongly on how much computational time is available to the algorithm. Parameters β and ρ are the earliest and most frequently modified parameters in the literature [90, 93]. A significant effort has been done to define adaptive parameter choices, where the algorithm parameters are modified as a predefined function of the ACO search behavior [72, 111], or to define self-adaptive parameter adaptation schemes, where the algorithm modifies the parameters itself during execution [69, 87, 110]. A recent paper [127] critically reviews the works that have applied parameter adaptation to ACO algorithms. The same paper also shows that the convergence speed of MMAS can be significantly improved, without leading to early stagnation, by means of very simple prescheduled parameter variations, such as starting with a very high value of β and reducing it to a much smaller value after a number of iterations. López-Ibáñez and Stützle [81] demonstrate how such prescheduled variations can be automatically tuned for the purpose of improving the convergence speed of ACO algorithms and show that fast increments in the number of ants, fast decrements of β, and, in particular, slow decrements of q_0 produce remarkable improvements on the TSP. (The software described in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP" and published alongside this chapter is also able to replicate these parameter variation schemes.) Finally, Pellegrini et al. [108] empirically showed that parameter adaptation methods proposed in the literature (excluding prescheduled variations) may often worsen the performance of state-of-the-art ACO algorithms and only improve over static parameter settings when the latter produce very poor results and when only one parameter is carefully adapted.

Conclusions

In this chapter, we have reviewed ACO algorithms from a component-wise perspective, and we have provided an example implementation of ACO algorithms according to this perspective. We have also concisely reviewed trends in ACO research on applications to problems with challenging characteristics such as time-varying problem data, multiple objectives, and stochastic data as well as algorithmic developments that combine ACO algorithms with other techniques. We did not cover in this chapter research on the parallelization of ACO algorithms, which has the goal of either speeding up the execution of a single run of an ACO algorithm or of increasing the quality of the final solution obtained within the same wall-clock time (by evaluating more solutions than would be possible without parallelization) [107]. Another important area not covered in this chapter is the ongoing work on the theoretical understanding of ACO algorithms. Early theoretical work has focused on the convergence behavior of ACO algorithms [57, 58, 121]. Later work has analyzed the dynamics of ACO behavior [91] and its relationship to other algorithms [94]. An overview of early theoretical work on ACO is given by Dorigo and Blum [32]. More recent works theoretically analyze the convergence speed of ACO algorithms [28, 60–62, 66, 71, 103, 104].

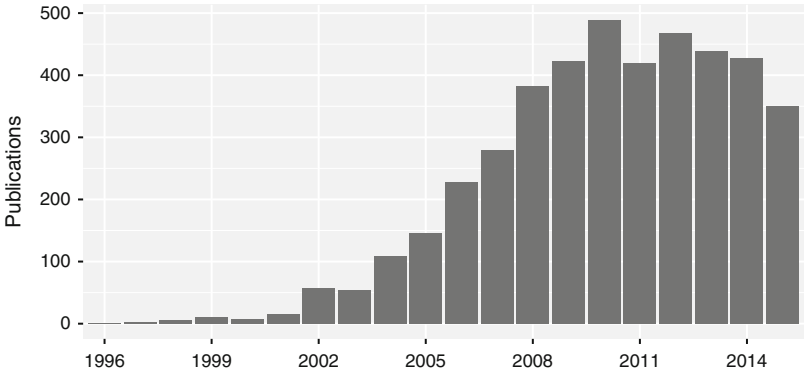


Fig. 4 Number of publications per year in the Scopus database for which their titles contain the keywords "ant system," "ant colony system," or "ant colony optimization"

These developments together with the large number of ACO algorithm variants that have been proposed in the literature and the large number of applications to a very diverse set of problems show that ACO has become a widely accepted and well-established metaheuristic. This fact is confirmed also by the number of publications that have as central topic some aspect related to ACO. Figure 4 shows the number of publications in the Scopus database that have one of the three terms "ant system," "ant colony system," or "ant colony optimization" in the article title. There is a very strong increase from the period 2000–2010, while after 2010 the number of publications remained at a high level. The success of ACO is also witnessed by a number of industrial applications. For example, AntOptima (www.antoptima.com) is a small company using the ACO methodology for tackling industrial problems in distribution and production management, while ArcelorMittal is using ACO algorithms in various areas relevant to steel production to improve operative production performance [25, 44]. ACO is also a central topic of journals and conferences specializing in the area of swarm intelligence such as the ANTS conference series started in 1998 (http://iridia.ulb.ac.be/ants/) and the journal "*Swarm Intelligence*." This success of ACO is due to (i) a truly original algorithmic idea inspired by a natural phenomenon, (ii) a strong versatility of the resulting algorithmic method, and (iii) a focus of ACO research on performance and a pragmatic approach trying to make it a useful technique. In the future, the ideas underlying the ACO metaheuristic promise to be of crucial importance when tackling challenging problems where aspects of constructive search, distributed information, and dynamic domains match well the inherent characteristics of ACO.

Cross-References

▸ GRASP
▸ Iterated Greedy

- ▶ Iterated Local Search
- ▶ Multi-objective Optimization

Acknowledgments The research leading to the results presented in this chapter has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement nº246939. Thomas Stützle and Marco Dorigo acknowledge support of the F.R.S.-FNRS of which they are a senior research associate and a research director, respectively.

References

1. Acan A (2004) An external memory implementation in ant colony optimization. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 73–84
2. Alaya I, Solnon C, Ghédira K (2007) Ant colony optimization for multi-objective optimization problems. In: 19th IEEE international conference on tools with artificial intelligence (ICTAI 2007), vol 1. IEEE Computer Society Press, Los Alamitos, pp 450–457
3. Alba E, Chicano F (2007) ACOhg: dealing with huge graphs. In: Thierens D et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO 2007). ACM Press, New York, pp 10–17
4. Angus D (2007) Population-based ant colony optimisation for multi-objective function optimisation. In: Randall M, Abbass HA, Wiles J (eds) Progress in artificial life (ACAL). Lecture notes in computer science, vol 4828. Springer, Heidelberg, pp 232–244
5. Angus D, Woodward C (2009) Multiple objective ant colony optimisation. Swarm Intell 3(1):69–85
6. April J, Glover F, Kelly JP, Laguna M (2003) Simulation-based optimization: practical introduction to simulation optimization. In: Chick SE, Sanchez PJ, Ferrin DM, Morrice DJ (eds) Proceedings of the 35th winter simulation conference: driving innovation, vol 1. ACM Press, New York, pp 71–78
7. Balaprakash P, Birattari M, Stützle T, Yuan Z, Dorigo M (2009) Estimation-based ant colony optimization algorithms for the probabilistic travelling salesman problem. Swarm Intell 3(3):223–242
8. Balaprakash P, Birattari M, Stützle T, Dorigo M (2010) Estimation-based metaheuristics for the probabilistic travelling salesman problem. Comput Oper Res 37(11):1939–1951
9. Balaprakash P, Birattari M, Stützle T, Dorigo M (2015) Estimation-based metaheuristics for the single vehicle routing problem with stochastic demands and customers. Comput Optim Appl 61(2):463–487
10. Barán B, Schaerer M (2003) A multiobjective ant colony system for vehicle routing problem with time windows. In: Proceedings of the twenty-first IASTED international conference on applied informatics, Insbruck, pp 97–102
11. Bianchi L, Gambardella LM, Dorigo M (2002) An ant colony optimization approach to the probabilistic traveling salesman problem. In: Merelo JJ et al (eds) Parallel problem solving from nature, PPSN VII. Lecture notes in computer science, vol 2439. Springer, Heidelberg, pp 883–892
12. Bianchi L, Birattari M, Manfrin M, Mastrolilli M, Paquete L, Rossi-Doria O, Schiavinotto T (2006) Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J Math Modell Algorithms 5(1):91–110
13. Bianchi L, Dorigo M, Gambardella LM, Gutjahr WJ (2009) A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput 8(2):239–287

14. Bilchev G, Parmee IC (1995) The ant colony metaphor for searching continuous design spaces. In: Fogarty TC (ed) Evolutionary computing, AISB Workshop. Lecture notes in computer science, vol 993. Springer, Heidelberg, pp 25–39
15. Birattari M, Balaprakash P, Dorigo M (2006) The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In: Doerner KF, Gendreau M, Greistorfer P, Gutjahr WJ, Hartl RF, Reimann M (eds) Metaheuristics – progress in complex systems optimization. Operations research/computer science interfaces series, vol 39. Springer, New York, pp 189–203
16. Blum C (2005) Beam-ACO – hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
17. Blum C, Dorigo M (2005) Search bias in ant colony optimization: on the role of competition-balanced systems. IEEE Trans Evol Comput 9(2):159–174
18. Brailsford SC, Gutjahr WJ, Rauner MS, Zeppelzauer W (2006) Combined discrete-event simulation and ant colony optimisation approach for selecting optimal screening policies for diabetic retinopathy. Comput Manag Sci 4(1):59–83
19. Bullnheimer B, Hartl RF, Strauss C (1999) A new rank-based version of the ant system: a computational study. Cent Eur J Oper Res Econ 7(1):25–38
20. Colorni A, Dorigo M, Maniezzo V (1992) Distributed optimization by ant colonies. In: Varela FJ, Bourgine P (eds) Proceedings of the first European conference on artificial life. MIT Press, Cambridge, pp 134–142
21. Cordón O, de Viana IF, Herrera F, Moreno L (2000) A new ACO model integrating evolutionary computation concepts: the best-worst ant system. In: Dorigo M et al (eds) Abstract proceedings of ANTS 2000 – from ant colonies to artificial ants: second international workshop on ant algorithms. IRIDIA, Université Libre de Bruxelles, Belgium, pp 22–29
22. Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the Argentine ant. J Insect Behav 3(2):159–168
23. Di Caro GA, Dorigo M (1998) AntNet: distributed stigmergetic control for communications networks. J Artif Intell Res 9:317–365
24. Di Caro GA, Ducatelle F, Gambardella LM (2005) AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. Eur Trans Telecommun 16(5):443–455
25. Díaz D, Valledor P, Areces P, Rodil J, Suárez M (2014) An ACO algorithm to solve an extended cutting stock problem for scrap minimization in a bar mill. In: Dorigo M et al (eds) Swarm Intelligence, 9th International Conference, ANTS 2014. Lecture notes in computer science, vol 8667. Springer, Heidelberg, pp 13–24
26. Doerner KF, Hartl RF, Reimann M (2003) Are COMPETants more competent for problem solving? The case of a multiple objective transportation problem. Cent Eur J Oper Res Econ 11(2):115–141
27. Doerner KF, Gutjahr WJ, Hartl RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. Ann Oper Res 131:79–99
28. Doerr B, Neumann F, Sudholt D, Witt C (2011) Runtime analysis of the 1-ANT ant colony optimizer. Theor Comput Sci 412(1):1629–1644
29. Donati AV, Montemanni R, Casagrande N, Rizzoli AE, Gambardella LM (2008) Time dependent vehicle routing problem with a multi ant colony system. Eur J Oper Res 185(3):1174–1191
30. Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy (in Italian)
31. Dorigo M (2007) Ant colony optimization. Scholarpedia 2(3):1461
32. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2–3):243–278
33. Dorigo M, Di Caro GA (1999) The ant colony optimization meta-heuristic. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 11–32
34. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66
35. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge

36. Dorigo M, Maniezzo V, Colorni A (1991) The ant system: an autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy
37. Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy
38. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26(1):29–41
39. Dréo J, Siarry P (2004) Continuous interacting ant colony algorithm based on dense heterarchy. Future Gener Comput Syst 20(5):841–856
40. Ehrgott M (2000) Multicriteria optimization. Lecture notes in economics and mathematical systems, vol 491. Springer, Berlin
41. Eyckelhof CJ, Snoek M (2002) Ant systems for a dynamic TSP: ants caught in a traffic jam. In: Dorigo M et al (eds) Ant algorithms. Third international workshop, ANTS 2002. Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 88–99
42. Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67–71
43. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Global Optim 6:109–113
44. Fernández S, Álvarez S, Díaz D, Iglesias M, Ena B (2014) Scheduling a galvanizing line by ant colony optimization. In: Dorigo M et al (eds) Swarm Intelligence. 9th International conference, ANTS 2014. Lecture notes in computer science, vol 8667. Springer, Heidelberg, pp 146–157
45. Gambardella LM, Dorigo M (1996) Solving symmetric and asymmetric TSPs by ant colonies. In: Bäck T, Fukuda T, Michalewicz Z (eds) Proceedings of the 1996 IEEE international conference on evolutionary computation (ICEC'96). IEEE Press, Piscataway, pp 622–627
46. Gambardella LM, Taillard ÉD, Agazzi G (1999) MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 63–76
47. Gambardella LM, Montemanni R, Weyland D (2012) Coupling ant colony systems with strong local searches. Eur J Oper Res 220(3):831–843
48. García-Martínez C, Cordón O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur J Oper Res 180(1):116–148
49. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman & Co, San Francisco
50. Glover F (1998) A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald EMA, Schoenauer M, Snyers D (eds) Artificial evolution. Lecture notes in computer science, vol 1363. Springer, Heidelberg, pp 1–51
51. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Boston
52. Guntsch M, Branke J (2003) New ideas for applying ant colony optimization to the probabilistic tsp. In: Cagnoni S et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2003. Lecture notes in computer science, vol 2611. Springer, Heidelberg, pp 165–175
53. Guntsch M, Middendorf M (2001) Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers EJW et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2001. Lecture notes in computer science, vol 2037. Springer, Heidelberg, pp 213–222
54. Guntsch M, Middendorf M (2002) Applying population based ACO to dynamic optimization problems. In: Dorigo M et al (eds) Ant algorithms. Third international workshop, ANTS 2002. Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 111–122
55. Guntsch M, Middendorf M (2002) A population based approach for ACO. In: Cagnoni S et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2002. Lecture notes in computer science, vol 2279. Springer, Heidelberg, pp 71–80

56. Guntsch M, Middendorf M (2003) Solving multi-objective permutation problems with population based ACO. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary multi-criterion optimization, EMO 2003. Lecture notes in computer science, vol 2632. Springer, Heidelberg, pp 464–478
57. Gutjahr WJ (2000) A Graph-based ant system and its convergence. Future Gener Comput Syst 16(8):873–888
58. Gutjahr WJ (2002) ACO algorithms with guaranteed convergence to the optimal solution. Inf Process Lett 82(3):145–153
59. Gutjahr WJ (2004) S-ACO: An ant-based approach to combinatorial optimization under uncertainty. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 238–249
60. Gutjahr WJ (2006) On the finite-time dynamics of ant colony optimization. Method Comput Appl Probab 8(1):105–133
61. Gutjahr WJ (2007) Mathematical runtime analysis of ACO algorithms: survey on an emerging issue. Swarm Intell 1(1):59–79
62. Gutjahr WJ (2008) First steps to the runtime complexity analysis of ant colony optimization. Comput Oper Res 35(9):2711–2727
63. Gutjahr WJ, Rauner MS (2007) An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. Comput Oper Res 34(3):642–666
64. Hart JP, Shogan AW (1987) Semi-greedy heuristics: an empirical study. Oper Res Lett 6(3):107–114
65. Hoos HH (2012) Programming by optimization. Commun ACM 55(2):70–80
66. Iacopino C, Palmer P (2012) The dynamics of ant colony optimization algorithms applied to binary chains. Swarm Intell 6(4):343–377
67. Iredi S, Merkle D, Middendorf M (2001) Bi-criterion optimization with multi colony ant algorithms. In: Zitzler E, Deb K, Thiele L, Coello Coello CA, Corne D (eds) Evolutionary Multi-criterion Optimization, EMO 2001. Lecture notes in computer science, vol 1993. Springer, Heidelberg, pp 359–372
68. Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Global Optim 13(4):455–492
69. Khichane M, Albert P, Solnon C (2009) An ACO-based reactive framework for ant colony optimization: first experiments on constraint satisfaction problems. In: Stützle T (ed) Learning and intelligent optimization. Third international conference, LION 3. Lecture notes in computer science, vol 5851. Springer, Heidelberg, pp 119–133
70. Korb O, Stützle T, Exner TE (2007) An ant colony optimization approach to flexible protein–ligand docking. Swarm Intell 1(2):115–134
71. Kötzing T, Neumann F, Röglin H, Witt C (2012) Theoretical analysis of two ACO approaches for the traveling salesman problem. Swarm Intell 6(1):1–21
72. Kovářík O, Skrbek M (2008) Ant colony optimization with castes. In: Kurkova-Pohlova V, Koutnik J (eds) ICANN'08: Proceedings of the 18th international conference on artificial neural networks, Part I. Lecture notes in computer science, vol 5163. Springer, Heidelberg, pp 435–442
73. Leguizamón G, Alba E (2013) Ant colony based algorithms for dynamic optimization problems. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for dynamic optimization, studies in computational intelligence, vol 433. Springer, Berlin/Heidelberg, pp 189–210
74. Liao T, Montes de Oca MA, Aydın D, Stützle T, Dorigo M (2011) An incremental ant colony algorithm with local search for continuous optimization. In: Krasnogor N, Lanzi PL (eds) Proceedings of the genetic and evolutionary computation conference, GECCO 2011. ACM Press, New York, pp 125–132
75. Liao T, Socha K, Montes de Oca MA, Stützle T, Dorigo M (2014) Ant colony optimization for mixed-variable optimization problems. IEEE Trans Evol Comput 18(4):503–518
76. Liao T, Stützle T, Montes de Oca MA, Dorigo M (2014) A unified ant colony optimization algorithm for continuous optimization. Eur J Oper Res 234(3):597–609

77. Lissovoi A, Witt C (2015) Runtime analysis of ant colony optimization on dynamic shortest path problems. Theor Comput Sci 61(Part A):73–85
78. López-Ibáñez M, Blum C (2010) Beam-ACO for the travelling salesman problem with time windows. Comput Oper Res 37(9):1570–1583
79. López-Ibáñez M, Stützle T (2012) The automatic design of multi-objective ant colony optimization algorithms. IEEE Trans Evol Comput 16(6):861–875
80. López-Ibáñez M, Stützle T (2012) An experimental analysis of design choices of multi-objective ant colony optimization algorithms. Swarm Intell 6(3):207–232
81. López-Ibáñez M, Stützle T (2014) Automatically improving the anytime behaviour of optimisation algorithms. Eur J Oper Res 235(3):569–582
82. López-Ibáñez M, Paquete L, Stützle T (2006) Hybrid population-based algorithms for the bi-objective quadratic assignment problem. J Math Modell Algorithms 5(1):111–137
83. López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58
84. Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369
85. Maniezzo V, Carbonaro A (2000) An ANTS heuristic for the frequency assignment problem. Futur Gener Comput Syst 16(8):927–935
86. Marriott K, Stuckey P (1998) Programming with constraints. MIT Press, Cambridge
87. Martens D, Backer MD, Haesen R, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Trans Evol Comput 11(5):651–665
88. Massen F, Deville Y, van Hentenryck P (2012) Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In: Beldiceanu N, Jussien N, Pinson E (eds) Integration of AI and OR techniques in contraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 7298. Springer, Heidelberg, pp 260–274
89. Massen F, López-Ibáñez M, Stützle T, Deville Y (2013) Experimental analysis of pheromone-based heuristic column generation using irace. In: Blesa MJ, Blum C, Festa P, Roli A, Sampels M (eds) Hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Heidelberg, pp 92–106
90. Merkle D, Middendorf M (2001) Prospects for dynamic algorithm control: Lessons from the phase structure of ant scheduling algorithms. In: Heckendorn RB (ed) Proceedings of the 2001 genetic and evolutionary computation conference – workshop program. Workshop "The Next Ten Years of Scheduling Research". Morgan Kaufmann Publishers, San Francisco, pp 121–126
91. Merkle D, Middendorf M (2002) Modeling the dynamics of ant colony optimization. Evol Comput 10(3):235–262
92. Merkle D, Middendorf M (2003) Ant colony optimization with global pheromone evaluation for scheduling a single machine. Appl Intell 18(1):105–111
93. Merkle D, Middendorf M, Schmeck H (2002) Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 6(4):333–346
94. Meuleau N, Dorigo M (2002) Ant colony optimization and stochastic gradient descent. Artif Life 8(2):103–121
95. Meyer B, Ernst AT (2004) Integrating ACO and constraint propagation. In: Dorigo M et al (eds) Ant colony optimization and swarm intelligence. 4th international workshop, ANTS 2004. Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 166–177
96. Michel R, Middendorf M (1998) An island model based ant system with lookahead for the shortest supersequence problem. In: Eiben AE, Bäck T, Schoenauer M, Schwefel HP (eds) Parallel problem solving from nature, PPSN V. Lecture notes in computer science, vol 1498. Springer, Heidelberg, pp 692–701
97. Monmarché N, Venturini G, Slimane M (2000) On how *pachycondyla apicalis* ants suggest a new search algorithm. Futur Gener Comput Syst 16(8):937–946
98. Montemanni R, Gambardella LM, Rizzoli AE, Donati AV (2005) Ant colony system for a dynamic vehicle routing problem. J Comb Optim 10:327–343

99. Montgomery J, Randall M, Hendtlass T (2008) Solution bias in ant colony optimisation: lessons for selecting pheromone models. Comput Oper Res 35(9):2728–2749
100. Moraglio A, Kattan A (2011) Geometric generalisation of surrogate model based optimization to combinatorial spaces. In: Merz P, Hao JK (eds) Proceedings of EvoCOP 2011 – 11th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 6622. Springer, Heidelberg, pp 142–154
101. Morin S, Gagné C, Gravel M (2009) Ant colony optimization with a specialized pheromone trail for the car-sequencing problem. Eur J Oper Res 197(3):1185–1191
102. Nallaperuma S, Wagner M, Neumann F (2014) Parameter prediction based on features of evolved instances for ant colony optimization and the traveling salesperson problem. In: Bartz-Beielstein T, Branke J, Filipič B, Smith J (eds) PPSN 2014. Lecture notes in computer science, vol 8672. Springer, Heidelberg, pp 100–109
103. Neumann F, Witt C (2006) Runtime analysis of a simple ant colony optimization algorithm. Electronic Colloquium on Computational Complexity (ECCC) 13(084)
104. Neumann F, Sudholt D, Witt C (2009) Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. Swarm Intell 3(1):35–68
105. Ow PS, Morton TE (1988) Filtered beam search in scheduling. Int J Prod Res 26:297–307
106. Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization – algorithms and complexity. Prentice Hall, Englewood Cliffs
107. Pedemonte M, Nesmachnow S, Cancela H (2011) A survey on parallel ant colony optimization. Appl Soft Comput 11(8):5181–5197
108. Pellegrini P, Birattari M, Stützle T (2012) A critical analysis of parameter adaptation in ant colony optimization. Swarm Intell 6(1):23–48
109. Pérez Cáceres L, López-Ibáñez M, Stützle T (2015) Ant colony optimization on a limited budget of evaluations. Swarm Intell 9(2-3):103–124
110. Randall M (2004) Near parameter free ant colony optimisation. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 374–381
111. Randall M, Montgomery J (2002) Candidate set strategies for ant colony optimisation. In: Dorigo M et al (eds) 3rd international workshop on Ant algorithms (ANTS 2002). Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 243–249
112. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177(3):2033–2049
113. Schilde M, Doerner KF, Hartl RF, Kiechle G (2009) Metaheuristics for the bi-objective orienteering problem. Swarm Intell 3(3):179–201
114. Socha K (2004) ACO for continuous and mixed-variable optimization. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 25–36
115. Socha K, Dorigo M (2007) Ant colony optimization for mixed-variable optimization problems. Technical Report TR/IRIDIA/2007-019, IRIDIA, Université Libre de Bruxelles
116. Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. Eur J Oper Res 185(3):1155–1173
117. Steuer RE (1986) Multiple criteria optimization: theory, computation and application. Wiley series in probability and mathematical statistics. John Wiley & Sons, New York
118. Stützle T (1998) Local search algorithms for combinatorial problems – analysis, improvements, and new applications. PhD thesis, FB Informatik, TU Darmstadt
119. Stützle T (2002) ACOTSP: a software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. http://www.aco-metaheuristic.org/aco-code/
120. Stützle T, Dorigo M (1999) ACO algorithms for the quadratic assignment problem. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 33–50
121. Stützle T, Dorigo M (2002) A short convergence proof for a class of ACO algorithms. IEEE Trans Evol Comput 6(4):358–365

122. Stützle T, Hoos HH (1996) Improving the ant system: a detailed report on the MAX–MIN ant system. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt
123. Stützle T, Hoos HH (1997) The MAX–MIN ant system and local search for the traveling salesman problem. In: Bäck T, Michalewicz Z, Yao X (eds) Proceedings of the 1997 IEEE international conference on evolutionary computation (ICEC'97). IEEE Press, Piscataway, pp 309–314
124. Stützle T, Hoos HH (1999) MAX–MIN ant system and local search for combinatorial optimization problems. In: Voß S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic, Dordrecht, pp 137–154
125. Stützle T, Hoos HH (2000) MAX–MIN ant system. Future Gener Comput Syst 16(8):889–914
126. Stützle T, López-Ibáñez M, Dorigo M (2011) A concise overview of applications of ant colony optimization. In: Cochran JJ (ed) Wiley encyclopedia of operations research and management science, vol 2. John Wiley & Sons, pp 896–911
127. Stützle T, López-Ibáñez M, Pellegrini P, Maur M, Montes de Oca MA, Birattari M, Dorigo M (2012) Parameter adaptation in ant colony optimization. In: Hamadi Y, Monfroy E, Saubion F (eds) Autonomous search. Springer, Berlin, pp 191–215
128. Taillard ÉD (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17(4-5):443–455
129. Teixeira C, Covas J, Stützle T, Gaspar-Cunha A (2012) Multi-objective ant colony optimization for solving the twin-screw extrusion configuration problem. Eng Optim 44(3):351–371
130. Torres CE, Rossi LF, Keffer J, Li K, Shen CC (2010) Modeling, analysis and simulation of ant-based network routing protocols. Swarm Intell 4(3):221–244
131. Tsutsui S (2006) An enhanced aggregation pheromone system for real-parameter optimization in the ACO metaphor. In: Dorigo M et al (eds) 5th international workshop on Ant colony optimization and swarm intelligence (ANTS 2006). Lecture notes in computer science, vol 4150. Springer, Heidelberg, pp 60–71
132. Tsutsui S (2007) Ant colony optimization with cunning ants. Trans Jpn Soc Artifi Intell 22:29–36
133. Wiesemann W, Stützle T (2006) Iterated ants: an experimental study for the quadratic assignment problem. In: Dorigo M et al (eds) 5th international workshop on Ant colony optimization and swarm intelligence (ANTS 2006). Lecture notes in computer science, vol 4150. Springer, Heidelberg, pp 179–190
134. Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, Bartz-Beielstein T (2014) Efficient global optimization for combinatorial problems. In: Igel C, Arnold DV (eds) Proceedings of the genetic and evolutionary computation conference, GECCO 2014. ACM Press, New York, pp 871–878

Evolutionary Algorithms

14

David Corne and Michael A. Lones

Contents

Introduction . . . 410
Principal Algorithms . . . 412
  Genetic Algorithms . . . 412
  Evolution Strategies and Evolutionary Programming . . . 414
  Estimation of Distribution Algorithms . . . 415
  Differential Evolution . . . 415
  Performance Comparisons . . . 416
Common Variants . . . 416
  Alternative Representations . . . 417
  Hybridization with Local Search . . . 418
  Multimodal Optimization . . . 418
  Multi-objective Optimization . . . 419
  Dynamic Optimization . . . 421
  Coevolving Solutions . . . 421
Applying Evolutionary Algorithms . . . 422
  Choosing a Methodology . . . 422
  Choosing Parameters . . . 422
  Software Tools . . . 423
Case Studies . . . 424
  Evolutionary Algorithms at Large . . . 424
  Using Niching and Coevolution to Understand Gene Regulation . . . 424
  Evolving Classifiers for Parkinson's Disease Diagnosis . . . 426
Conclusion . . . 426
Cross-References . . . 427
References . . . 427

D. Corne (✉) · M. A. Lones
Heriot-Watt University, Edinburgh, UK
e-mail: d.w.corne@hw.ac.uk; mal1@hw.ac.uk

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_27

Abstract

Evolutionary algorithms (EAs) are population-based metaheuristics, originally inspired by aspects of natural evolution. Modern varieties incorporate a broad mixture of search mechanisms, and tend to blend inspiration from nature with pragmatic engineering concerns; however, all EAs essentially operate by maintaining a population of potential solutions and in some way artificially 'evolving' that population over time. Particularly well-known categories of EAs include genetic algorithms (GAs), Genetic Programming (GP), and Evolution Strategies (ES). EAs have proven very successful in practical applications, particularly those requiring solutions to combinatorial problems. EAs are highly flexible and can be configured to address any optimization task, without the requirements for reformulation and/or simplification that would be needed for other techniques. However, this flexibility goes hand in hand with a cost: the tailoring of an EA's configuration and parameters, so as to provide robust performance for a given class of tasks, is often a complex and time-consuming process. This tailoring process is one of the many ongoing research areas associated with EAs.

Keywords

Population-based · Genetic algorithms · Evolutionary programming · Evolution strategies

Introduction

Evolutionary algorithms (EAs) are population-based metaheuristics. Historically, the design of EAs was motivated by observations about natural evolution in biological populations. Recent varieties of EA tend to include a broad mixture of influences in their design, although biological terminology is still in common use. The term "EA" is also sometimes extended to algorithms that are motivated by population-based aspects of EAs but which are not directly descended from traditional EAs, such as scatter search. The term evolutionary computation is also used to refer to EAs but usually as a generic term that includes optimization algorithms motivated by other processes but which generally involve a population of potential solutions adapting and improving over time. This includes algorithms inspired by other natural processes, such as ant colony optimization (ACO – inspired by the collective problem-solving behavior of social insects) and artificial immune systems (AIS – inspired by the adaptive molecular processes involved in the human immune system's ability to recognize and destroy harmful agents). It also includes algorithms inspired by social behavior, such as particle swarm optimization (PSO); however, in recent years, the convention has shifted toward using the term "swarm intelligence" to describe PSO, ACO, and other algorithms inspired more by interaction than evolution. Although these algorithms often resemble EAs, this is

not always the case, and they will not generally be discussed in this chapter. For a discussion of their commonalities and differences, the reader is referred to [1].

Over the years, EAs have become an extremely rich and diverse field of study. In part this arises from their inherent design flexibility – there are innumerable ways to specify an algorithm that operates according to the core concepts of evolution. In part this also arises from the fact that their performance tends to be highly problem dependent – an EA that works well on one task may well perform poorly on another, even quite similar, task. Arising from these factors and others, the sheer number of publications in this area is challenging for people new to the field. To address this, this chapter aims to give a concise overview of EAs and their application, with an emphasis on contemporary rather than historical usage.

The main classes of EA in contemporary usage are (in order of popularity) genetic algorithms (GAs), evolution strategies (ESs), differential evolution (DE), and estimation of distribution algorithms (EDAs). Multi-objective evolutionary algorithms (MOEAs), which generalize EAs to the multiple objective case, and memetic algorithms (MAs), which hybridize EAs with local search, are also popular, particularly within applied work. Special-purpose EAs, such as genetic programming (GP) and learning classifier systems (LCS), are also widely used. These are all discussed in this chapter.

Although these algorithms differ from each other in a number of respects, they are all based around the same core process. Each of them maintains a population of search points (known variously as candidate solutions, individuals, chromosomes, or agents). These are typically generated at random and are then iteratively evolved over a series of generations by applying variation operators and selection. Variation operators generate changes to members of the population, i.e., they carry out moves through the search space. After each generation, the objective value (or *fitness*) of each search point is calculated. Selection then removes the search points with the lowest objective values, meaning that only the best search points are maintained, and new search points are always derived from these. It is this combination of maintaining a population of search points and carrying out selection between search points that differentiates EAs from most other metaheuristics.

Each EA uses its own distinctive set of variation operators, which are sometimes inspired by the mutative and recombinative processes that generate diversity in biological evolution. The mutation operator resembles the generation of "moves" in other optimization algorithms and involves sampling the neighborhood around an existing search point in some fashion. A typical approach would be to randomly change one component of a solution, though a particular EA may use more than one kind of mutation operator. The recombination (or *crossover*) operator explores the region between two or more search points, for example, by randomly reassembling the components that make up two existing solutions. This process of searching the region between existing search points is also a distinctive feature of EAs, though its practical utility depends upon the structure of the search space. Some EAs, particularly "evolutionary programming" and older varieties of evolution strategy, do not use recombination at all.

Principal Algorithms

All EAs share certain common features, including in particular the broad concepts of *variation* and *selection*, as introduced in section "Introduction", operating over a series of *generations*. More fundamentally, all EAs work by seeking the "best" (in some sense) solutions they can find to a given optimization task. Put in another way, an EA is almost always used to find the solution data-structure x which optimizes a given function $f(x)$. For example, the task at hand may be straightforward numerical function optimization, where the candidate solution data-structure is a binary string that is interpreted as a list of real-valued parameters, and $f(x)$ is a mathematical function over those parameters whose result is a scalar value. Alternatively, the task at hand may be to find an ideal schedule for a collection of manufacturing tasks at a factory; in this case, the candidate solution data-structure might be an ordered sequence of task identifiers, and $f(x)$would be a program that simulates the schedule from the sequence, returning a vector of quality indicators including cost, time, and risk. Whatever the nature of the underlying data-structure being "evolved" and whatever the nature of $f(x)$, we can express a "canonical" EA in pseudocode as follows:

Preliminaries: determine a suitable way to represent solutions as data-structures, prepare the fitness function, and set $g = 0$.

Step 1: set $g = 0$, and generate and evaluate an initial population Sg of candidate solutions;
Step 2: apply *selection* operators to produce a set of "parent" solutions P;
Step 3: apply variation operators to P to produce a set of "chil" solutions C, and evaluate the fitness of each one;
Step 4: apply population update operations to the union of Sg and C to produce $S_{g+}1$, and increment g
Step 5: if a termination criterion has not been reached, go to Step 2; otherwise, finish and return the fittest member of the population.

In the above pseudocode, steps 2–5 constitute a "generation," whose major components are selection, variation, and then the "population update" step that leads to renewal of the population, now ready to enter the next generation. Individual EAs vary greatly in each of these steps, including the "preliminaries" step. In the remainder of this section, we provide brief introductions to the principal classes of EA that are in current use and then discuss existing understanding of their performance and applicability.

Genetic Algorithms

Genetic algorithms, or GAs, are one of the earliest forms of EAs and remain widely used. Candidate solutions, often referred to as *chromosomes* in the GA literature, comprise a vector of decision variables. Nowadays, these variables tend to have a

direct mapping to an optimization domain, with each decision variable (or *gene*) in the GA chromosome representing a value (or *allele*) that is to be optimized. However, it should be noted that historically GAs worked with binary strings, with real values encoded by multiple binary symbols, and that this practice is still sometimes used. GA solution vectors are either fixed-length or variable-length, with the former the more common of the two.

Given their long history, genetic algorithm implementations vary considerably. However, it is fairly common to use a mutation operator that changes each decision variable with a certain probability (values of 4–8% are typical, depending upon the problem domain). When the solution vector is a binary string, the effect of the mutation operator is simply to flip the value. More generally, if the solution vector is a "$k-$ary" string, in which each position can take any of a discrete set of k possible values, then the mutation operator is usually designed to choose a random new value from the available alphabet. If the solution vector is a string of real-valued parameters within a set range, the new value may be sampled from a uniform distribution in that range, or it may be sampled from a nonuniform (e.g., Gaussian) probability distribution centered around the current value. The latter is generally the preferred approach, since it leads to less disruptive change on average. Recombination is typically implemented using two-point or uniform crossover. Recombination tends to be applied at a high rate (e.g., 0.7, typically called the *crossover rate*); this means, for example, that when a variation operator is to be applied, the chance of this operator being a crossover operator will be 0.7; otherwise (depending on the algorithm), the operator may be the application of mutation to a single parent or simply copying a single parent. Two-point crossover chooses two *parent* solutions and two *crossover points* within the solutions. The values of the decision variables lying between these two points are then swapped to form two *child* solutions. Historically, "one-point" crossover was popular, which produced the first (second) child by copying the first (second) parent up to a randomly chosen point and then copied the second (first) parent thereafter. However, though more convenient for theoretical analyses, one-point crossover is rarely used in practice these days. Meanwhile, in uniform crossover, crossover points are created at each decision variable with a given probability. Other forms of crossover have also been used in GAs. Examples include line crossover and multi-parent crossover. Other variation operators, such as inversion, have been found useful for some problems.

Various forms of selection are used with GAs. Rank-based or tournament selection are generally preferred, since they maintain exploration better than the more traditional fitness-proportionate selection (e.g., roulette-wheel selection). Note, however, that the latter is still widely used. Rank-based selection involves ranking the population in terms of objective value. Population members are then chosen to become parents with a probability proportional to their rank. In tournament selection, a small group of solutions (typically three or four) are uniformly sampled from the population, and those with the highest objective value(s) become the parent(s) of the next child solution that is created. Tournament selection allows selective pressure to be easily varied by adjusting the tournament size.

Evolution Strategies and Evolutionary Programming

Evolutionary programming(EP) and evolution strategies (ES) also have a long history, starting earlier than the development of GAs. First described in the 1960s and 1970s, respectively (see [2] for a comprehensive account of the historical development), early ES and EP used only single-parent operators in the variation step (i.e., mutation). Meanwhile, EP was singular in focusing on using finite-state machines as the evolving data-structures, while ES soon introduced a recombination operator and championed the exploration of so-called "$m + n$" schemes, whereby a population of m parents would generate n children and the best n of the combined parents and children becomes the next generation of parents. In current research and practice, modern formulations of EP focus on numerical optimization over real-valued parameters and still eschew multi-parent operators in favor of paying attention to careful design of the mutation operator(s). Of the two styles, however, ES is more widely researched and deployed in practice, and particular variants of ES have shown particular prowess in numerical function optimization. Modern ESs incorporate strategies that carefully guide how the mutation operator is applied to each decision variable. Unlike GAs, ESs mutate every decision variable at each application of the operator and do so according to a set of *strategy parameters* that determine the magnitude of these changes. Strategy parameters usually control characteristics of probability distributions from which the new values of decision variables are drawn.

It is standard practice to adapt strategy parameters over the course of an ES run, the basic idea being that different types of move will be beneficial at different stages of search. Various techniques have been used to achieve this adaptation. Some of these involve applying a simple formula, e.g., the 1/5th rule, which involves increasing or decreasing the magnitude of changes based on the number of successful mutations that have recently been observed. Others are based around the idea of self-adaptation, which involves encoding the strategy parameters as additional decision variables and hence allowing evolution to come up with appropriate values. However, the most widely used contemporary approach is covariance matrix adaptation (CMA-ES), which uses a mechanism for estimating the directions of productive gradients within the search space and then applying moves in those directions. In this respect, CMA-ES has similarities with gradient-based optimization methods.

ESs use different recombination operators to GAs and often use more than two parents to create each child solution. For example, intermediate recombination gives a child solution the average values of each decision variable in each of the parent solutions. Weighted multi-recombination is similar but uses a weighted average, based on the fitness of each parent. Also unlike GAs, ESs tend to use deterministic rather than probabilistic selection mechanisms, whereby the best solutions in the population are always used as parents of the next generation.

Estimation of Distribution Algorithms

Like ESs, estimation of distribution algorithms [3], or EDAs, make use of probability distributions. However, rather than using them to describe a distribution of next moves, as an ES does, EDAs use them to describe a distribution of next sample solutions. The basic mechanism is quite simple. As for most EAs, the initial population of solutions is (typically) sampled from a uniform distribution. Selection is then used to remove the poorer members of this population, and a probability distribution is then constructed that attempts to model the statistics of the relatively high-fitness sample solutions that remain in the population. Importantly, this distribution is constructed in such a way that it "generalizes" the population members suitably well. The next generation of solutions is then constructed by sampling from this distribution. So, if the distribution was highly peaked around the existing samples, for example, the next generation would explore very little beyond the previous one. This pattern of building a distribution, sampling, and selection is then iterated in the usual generational fashion, with the hope that the final probability distribution will characterize solutions that are, or are close to, globally optimal.

While an EDA may be used with any kind of probability distribution, in practice, it is necessary to choose a distribution that induces an appropriate trade-off between efficiency and expressiveness. More expressive models, such as Bayesian networks and Markov models, can capture dependencies between decision variables but can be expensive to construct and sample from. Simple univariate distributions, by comparison, are cheap to build and sample from but are unable to capture dependencies between variables. This trade-off is reflected in the range of EDAs in common use. Population-based incremental learning (PBIL) and the compact genetic algorithm (CGA) are both examples of computationally efficient EDAs that build simple univariate models based on discrete variables. Because of their simplicity, they can be applied to large problem instances. Bayesian optimization algorithms (BOAs) lie at the other end of the spectrum: these can express dependencies between variables, and certain varieties can be applied to both discrete and continuous variables, but they are far more demanding of computational resources.

Differential Evolution

Differential evolution [4,5], or DE, is a relatively recent EA formulation which uses a mechanism for adaptive search that does not make use of probability distributions. While its basic mechanism is similar to a GA, its mutation operator is quite different, using a geometric approach that is motivated by the moves performed in the Nelder-Mead simplex search method. This involves selecting two existing search points from the population, taking their vector difference, scaling this by a constant F, and then adding this to a third search point, again sampled randomly

from the population. Following mutation, DE's crossover operator recombines the mutated search point (the *mutant vector*) with another existing search point (the *target vector*), replacing it if the child solution (known as a *trial vector*) is of equal or greater objective value. There are two standard forms of crossover [6]: exponential crossover and binomial crossover, which closely resemble GA two-point crossover and uniform crossover, respectively. The comparisons between target vector and trial vector play the same role as the selection mechanism in a GA or ES. Since DE requires each existing solution to be used once as a target vector, the whole population is replaced in the course of applying crossover.

An advantage of using simplex-like mutations in DE is that the algorithm is largely self-adapting, with moves automatically becoming smaller in each dimension as the population converges. More generally, the authors of the method have claimed that this sort of self-adaptation means that the size and direction of moves are automatically matched to the search landscape, a phenomenon they term *contour matching*. When compared to CMA-ES, for example, this means that the algorithm has few parameters and is relatively easy to implement.

Performance Comparisons

Fair comparisons of optimization algorithms are inherently challenging [7] and arguably unachievable. Nevertheless, there have been some attempts to understand the comparative performance of different EAs, particularly within the domain of continuous optimization. In particular, a series of workshops held at two of the largest annual EA conferences, CEC and GECCO, have sought to define benchmark suites of real-valued function optimization problems suitable for comparing EAs (and other optimizers) [8–10]. Using these benchmarks, a number of authors have shown their algorithms to perform better than others, including variants of CMA-ES and DE (see [4]). It should be borne in mind that these are not exhaustive studies, either in terms of problems or approaches. The "no free lunch theorem"(NFLT) [11] may also be considered when attempting to generalize these results to a wider spectrum of problems, although, in itself, the NFLT does not apply in the case of comparisons based on the standard suites of test problems (since those suites are not closed under permutation [12]). A nice example of the perils of comparison study in this field is shown by a recent study that showed how quite different conclusions could be drawn from a comparative study by changing minimization problems into maximization problems [13].

Common Variants

The general purpose EAs introduced in the last section are applicable to a wide range of problems. However, over the course of EA history, algorithmic variants have been developed to deal with the characteristics of particular categories of problem. Some of these categories are quite broad, for example, problems with multiple solutions.

Others are more specific, such as discrete optimization problems. In this section, we discuss a number of these EA variants, focusing on those which are commonly used to solve real-world optimization problems.

Alternative Representations

In common with other optimization algorithms, most EAs are designed to work with and optimize vectors (or, equivalently, lists or arrays) of decision variables. Solutions to many kinds of problems can be represented, either directly or indirectly, in this form. However, EAs are not limited to working with vectors, and there are often advantages to working directly with representations that are more natural for the problem domain: for example, matrices [14], trees [15], graphs [16], rule sets, etc. The general approach is the same as for the EAs discussed in the previous section, except that specialized initialization routines and variation operators are used to randomly create, mutate, and recombine instances of the appropriate solution representation. These variants are typically based around GAs or ESs, since these two classes of EA can be readily adapted to use alternative solution representations. Nevertheless, DE [17] and EDA [18] have also been used successfully with other representations.

Genetic programming (GP) [15] is a well-known GA variant in which each candidate solution is a *program*; in early GP this was invariably achieved by encoding programs as tree-structures, in which each "node" in the tree corresponds to a function whose input parameters are the results returned by its child-nodes and which itself returns a result to its parent nodes (if any). GP is still mostly used to optimize computer programs or mathematical expressions, often expressed as tree-structures. However alternative ways to encode programs are now often explored in GP, such as so-called linear GP, in which programs are represented as a sequence of parameterized instructions interacting via registers. A particularly common current use of GP is *symbolic regression*, which involves finding a mathematical expression that fits a particular data set. Unlike standard mathematical approaches to regression, such as curve fitting, GP makes relatively few assumptions about the function that generated the data, allowing a wide exploration of the space of possible solutions. GP is also widely used for solving classification problems. More generally, the GP community is interested in automatic programming, i.e., finding computer programs that solve a particular task, and there are many variants of GP that use particular forms of program representation. See [19] and [20] for overviews.

Some EAs work with two different solution representations, using one of these when creating and manipulating search points, the other when evaluating search points, and a mapping process that converts the former into the latter [21]. Many of these approaches are motivated by biology, and hence this process is known as a *genotype-phenotype* mapping, with the representation used during search termed the *genotype* and the representation used for evaluation termed the *phenotype*. This approach can be used when the natural representation for a domain is not well suited to being evolved, i.e., where mutation and recombination do

not lead to productive solutions. This approach has also been widely used for generating complex structures, such as large neural networks [22], where a genotype representation can be chosen that compresses repetitive features such as symmetry and modularity. This is arguably an area in which EAs benefit from their relationship to biology, since biology provides a ready source of information on how to represent complex structures in an evolvable way.

Hybridization with Local Search

EAs are often considered to be global search algorithms, since they explore a relatively wide region of the search space and are relatively good at escaping local optima. However, their convergence to optimal solutions can be relatively slow when compared to local search algorithms. For this reason, EAs are often hybridized with local search, using it to locally optimize members of the population at regular intervals, hence speeding up convergence. Although the resulting hybrid algorithms are known by various names, the term *memetic algorithm* [23] (MA) has become popular in recent years. Memetic, in this case, refers to an analogy between the role of local search in these algorithms and the role of within-generation learning in biological systems, though the majority of memetic algorithms have no particular biological justification beyond this. In principle, these algorithms may involve hybridizing an EA with any or with multiple local search algorithms and consequently are very diverse. For a recent review, see [24].

Beyond hastening convergence, MAs are also seen as a means of introducing domain knowledge into EAs. This is done through the use of specialized local search operators that are relevant to a particular domain. For example, this approach underlies the success of MAs in the area of discrete optimization [25]. Related to the idea of problem specialization in memetic algorithms is the concept of hyper-heuristics in EAs, which has developed some traction in recent years [26]. Generally speaking, hyper-heuristics are applicable to domains in which a variety of so-called low-level heuristics exist (or can be invented) to build quick, good solutions. In the job shop scheduling problem, for example, "shortest-process-time" and "earliest-available-machine" are two examples of low-level "dispatch" heuristics that, when iterated, can build a single solution quickly. Hyper-heuristics are essentially mechanisms used to explore *combinations* of such lower-level heuristic strategies. The term hyper-heuristics is also used to describe the cases in which an EA (often GP) both creates anew and combines such low-level heuristics. In a nutshell, the broad idea of hyper-heuristics is to search a space of algorithms that can solve a class of problems, rather than search the space of solutions directly for a single problem instance.

Multimodal Optimization

An advantage of maintaining a population of search points is that EAs can be readily applied to multimodal optimization problems in which there is more than one

solution of interest. However, effective multimodal optimization generally requires some modification to the EA's underlying behavior, since, although EAs explore diverse areas of the search space, they eventually converge to fairly small areas. This behavior can be mitigated, to an extent, by varying the global selection pressure used when choosing parents; for example, in the case of tournament selection, the tournament size can be made small, increasing the likelihood that less fit members of the population will contribute to the next generation of search points. While this increases exploration, it decreases exploitation: meaning that multiple solutions may be found, but they are less likely to be optimal. Since EAs are stochastic, and there is the potential for them to converge on different optima during different runs, another simple approach to finding multiple solutions is to run an EA multiple times. However, there is no guarantee that all optima will be explored, and algorithmic biases (such as the manner in which the initial population is generated) may favor some solutions over others.

A more effective approach is to use some kind of *niching* technique [27]. These aim to preserve global diversity in the population, but without lowering local selective pressure. Niching approaches are motivated by the biological concept of evolutionary niches, in which species compete within a niche but not between niches. In optimization terms, a niche is a local region within the search space that contains a solution of interest, and the aim is for the population to be distributed across all the relevant niches. Niching has been studied for some time in GAs, and techniques include crowding [28], fitness sharing [29], spatial segregation [30], and clustering [14]. For comparative studies, see [6] and [8]. A simple but effective example of niching is probabilistic crowding [28]. This works at the operator level and always replaces parent solutions with their children, meaning that search points are usually replaced with nearby search points and the population remains spread across the search space. Similar techniques have also been developed for use in DE. Niching is less commonly used in ESs, in part due to their use of smaller populations, though examples do exist [31]. It is also common to use multi-objective evolutionary algorithms (see below) to solve multimodal problems, since these algorithms often have effective mechanisms for preserving population diversity.

Multi-objective Optimization

Multi-objective EAs, or MOEAs, are used to solve problems which have multiple and often conflicting objectives. A central concept for MOEAs, and multi-objective optimization in general, is that of a non-dominated solution. This is a solution which is no worse than any of the other solutions within the population when all objectives are taken into account, and the aim of an MOEA is to build and maintain a population of non-dominated solutions that cover all trade-offs between the objectives. This is known as the Pareto optimal front. Exactly how this is achieved varies between MOEAs. However, a well-known example is NSGA-II (non-dominating sorting genetic algorithm) [32]. Prior to selection, NSGA-II ranks all solutions in terms of dominance: those which are non-dominated are assigned

rank 1, those which are only dominated by rank 1 solutions are assigned rank 2, etc. The population is then ordered by rank, and by a measure of crowding distance within ranks, and the first half of the ordered population is copied directly into the next generation. The remainder of the population is then filled by breeding, with parents selected from the higher ranks. Hence, non-dominated solutions are preserved between generations, and new solutions are explored via interbreeding, resulting in a diverse set of non-dominated solutions that approximate the Pareto optimal front.

The core challenge faced by multi-objective optimization (and absent from single-objective optimization) is how to rank candidate solutions in a way that leads to effective selection pressure, especially when the entire population (or most of it) may be non-dominated. Another way in which multi-objective optimization differs from single-objective optimization is in the nature of the "best-so-far solution." In single-objective optimization, the "best-so-far" solution is trivial to define and to keep track of; in multi-objective optimization, the situation is vastly different: the solution is, technically, the entire Pareto front, which is usually a set of solutions, whose cardinality may vary from one to the entire search space. For MOEAs, this leads to certain technical issues which are invariably addressed by maintaining an *archive* of non-dominated solutions; this archive simply keeps track of the "best-so-far" approximation to the Pareto front but is also often used as a reservoir for selection of parents. Approaches to the main challenge – how to apply effective selection pressure among the current population – are far more varied. While the approach taken by NSGA-II, as detailed above, is a common and quite successful one, many other styles of MOEA exist, which take different approaches to this central question. In PAES [33], for example, there is only a single "current" population member. Selection is consequently simplified; however, the challenge shifts to the question of whether or not to update the current solution with a newly generated one when the two are non-dominated; PAES makes this decision with the aid of its archive, preferring to explore new areas of the search space than to stay close to solutions already in the archive. Meanwhile, a different breed of MOEAs in this respect is represented by MOEA/D [34]; bypassing the need to distinguish between non-dominated solutions for selection purposes, MOEA/D "decomposes" a multi-objective problem into many single-objective simplifications of it, each involving a different weighting of the objectives. MOEA/D conducts these single-objective searches in parallel (typically using a local search mechanism) and organizes occasional communication between them, as well as bookkeeping activities that build and maintain the archive. Effectively, each of MOEA/D's single-objective searches explores a different area of the Pareto front. There are many other approaches, and MOEAs are becoming increasingly used as it becomes recognized that real-world problems are almost invariably multi-objective in nature. Further discussion of the latter point, as well as a first introduction to MOEAs, may be found in [35], while an example of a fairly recent review of MOEAs may be found in [36].

Dynamic Optimization

So far, our discussion of optimization has only considered problems in which the search space remains fixed. In many real-world problems, this is not the case, and various EA approaches are used to handle these situations. Dynamic optimization is an area in which EAs might be expected to perform relatively well, since the natural diversity present in their populations provides a recovery mechanism that can respond to slow changes in the optimization landscape. This is especially the case when diversity maintenance techniques are implemented, such as those already discussed in the sections on multimodal and multi-objective optimization. However, this diversity may be insufficient when the optimization targets change rapidly or abruptly. A simple solution in this situation is to inject extra diversity into the population when a change is detected, for instance, by adapting the variation operators so that larger moves are made. Detection of change can be done by reevaluating a proportion of the population, looking for significant changes in fitness.

A variety of more elaborate approaches have been developed to handle dynamic optimization in EAs. An approach inspired by biological systems is to use redundancy in the encoding of a solution. Rather than replacing components of a solution when variation operators are applied, this allows old components to become recessive, i.e., to remain present within the solution but not be expressed during evaluation. Later in the evolutionary process, these components can become reactivated, in effect providing a mechanism to backtrack to previous search points. This is particularly useful when changes in the search space are cyclic. A well-known example is the use of *multiploidy* in GAs [37], where each solution has multiple chromosomes (only one of which is dominant) and variation operators are able to move information between chromosomes. Other approaches to handling dynamic search spaces include predicting change and using multiple populations; see [38] for a recent review.

Coevolving Solutions

Coevolutionary algorithms [39] are motivated by the interactions that occur between species during the course of biological evolution and the roles these interactions are thought to play in the evolution of complex organisms. Most coevolutionary algorithms use multiple populations, one per *species*. Coevolutionary relationships in biology can be cooperative or competitive. The latter class are particularly well known and are encapsulated in the idea of predator-prey patterns of evolution, where an arms race between two species can lead to the rapid emergence of complex adaptations. Similar ideas have been explored in EAs, the classic example being the coevolution of sorting networks and sorting algorithms [40]: the discovery of harder problems (the sorting networks in the first population) leads to selective pressure to

discover better solutions (the sorting algorithms in the second population), which leads to selective pressure to discover harder problems and so on. Competitive coevolution can be used to solve hard problems and is also useful in circumstances where a fitness function cannot be defined. However, competitive coevolution is known to be difficult to control, and pathological situations can lead to ineffective search. See [39] for a review.

Cooperative coevolution, by comparison, is seen as a useful mechanism for breaking down large problems into more tractable chunks [41]. The idea is that a solution to a problem is divided into sub-components. Each of these sub-components is then evolved in a separate population, with its objective value dependent upon how compatible it is with sub-components being evolved in other populations. Following this, the coadapted sub-components are then assembled to form a complete solution. In [42], for example, the authors describe how a cooperative coevolutionary variant of DE can be used to solve numerical optimization problems with up to 1000 variables. Cooperative coevolution can also take place within a single population. An example of this is a Michigan-style learning classifier system (LCS), a form of EA that coevolves a population of rules that can collectively solve difficult problems in classification and machine learning [43].

Applying Evolutionary Algorithms

Choosing a Methodology

It can be difficult to choose which EA to use for a particular task, since there are many different EAs in common use and relatively little in the way of objective comparative guidance. In practice, it may be necessary to try out different EAs to find out which is the best match to a problem, especially when the problem is poorly understood. However, given whatever is known about the problem at hand, it might be possible to leverage existing understanding of the strengths and weaknesses of particular algorithm frameworks. Some guidance on this matter is available in studies of comparative performance mentioned at the end of section "Introduction". It is hoped that section "Principal Algorithms" also provides useful pointers if the problem is multimodal, multi-objective, dynamic, or unusually large and complex. It is also notable that multi-objective and memetic algorithms, in particular, have become popular for solving difficult real-world problems.

Choosing Parameters

EAs invariably have many parameters, and once an algorithm has been selected, it is normal to carry out parameter tuning in order to obtain a better fit between the algorithm and the problem. It can be challenging to obtain optimal parameter settings, since parameters are typically both numerous and not independent of one another. DE, for instance, is notable for having relatively few parameters, and this is

often portrayed as a strength of the method. However, EAs are relatively forgiving, and good performance is likely to be possible with nonoptimal parameter settings. Nevertheless, guidance is available for choosing the settings of certain parameters [44], and a number of techniques have been developed for automating the choice of parameter settings [45].

Software Tools

Tools support is an important issue for many practitioners, and a particular EA methodology is likely to be more appealing if it has a mature supported implementation. Tools support is also important if it is necessary to handcraft a new algorithm to solve a particular problem, and in this situation the language used by the tool may also be a significant concern. Table 1 summarizes the features of some of the better known EA tools. GAs are widely supported by all of these. ES support is also widely available, though EvA2 stands out in this regard, with implementations of a wide range of ES variants. DE and EDAs are more recent algorithms, and this is reflected by fewer mature tools. However, EvA2 is again notable for having an implementation of BOA and other EDAs. GP support is offered by a number of these tools, with ECJ implementing a particularly wide range of GP variants. Most also offer support for multimodal and multi-objective approaches, though MOEA Framework stands out for the latter. All of these tools allow custom code to be written. Most use Java or C + +, though DEAP is notable as a mature Python implementation and HeuristicLab is available for C# users.

Table 1 Open-source EA frameworks

Tool	Language	Summary
DEAP https://code.google.com/p/deap/	Python	Distributed Evolutionary Algorithms in Python offers good support for GAs and ESs. Also implements GP and MOEAs
ECJ http://cs.gmu.edu/~eclab/projects/ecj/	Java	Continuously developed since 1998, Evolutionary Computation in Java has particular strength in GP but also implements GAs, DE, and MOEAs
EO http://eodev.sourceforge.net	C++	Evolving Objects is an established general-purpose EA library with implementations of GAs, GP, ESs, and EDAs
EvA2 http://www.ra.cs.uni-tuebingen.de/software/JavaEvA/	Java	EvA2 is a general-purpose EA framework but has particular strengths in ESs and EDAs, including BOA
HeuristicLab http://dev.heuristiclab.com/	C#	HeuristicLab implements many of the common EA varieties and also has support for other population-based and local search metaheuristics
MOEA framework http://www.moeaframework.org/	Java	A relatively new EA framework with considerable strength in MOEAs and multi-objective variants of DE and GP
OpenBEAGLE https://code.google.com/p/beagle/	C++	A long established EA framework with good support for GAs and ESs. Also implements GP and NSGA-II

Case Studies

Evolutionary Algorithms at Large

Now more than half a century since the first appearance of "EA"-style algorithms in the research literature (widely considered to be [46]), EAs have penetrated almost every area of science and industry and are regularly used in solving an immense range of optimization problems.

In terms of broad classes of problem, EAs have enabled practitioners and researchers to make particular headway in *combinatorial optimization* which, unlike *numerical optimization*, for example, was hitherto poorly served by classical algorithms. In combinatorial optimization, the task is typically to find an ideal permutation (or otherwise constrained arrangement) of a set of entities – such as a sequence of customer visits for a delivery vehicle or a sequence of production tasks for each of a set of machines in a factory. Before EAs, the primary approach used to solve such problems was integer programming and constraint programming (and variations thereon),which tend to require a complex (and often tortuous) problem reformulation step. EAs, however, provide a far more accessible and flexible approach to addressing such problems and are now commonly used in practice for combinatorial tasks such as vehicle routing, job shop scheduling, and facility allocation [47].

Meanwhile, beyond combinatorial optimization, the inherent flexibility of EAs has led to their use, to at least some extent, in every conceivable area of science, enterprise, and industry in which one or more important tasks can be formulated in terms of optimization and/or design. To name just a few of the areas in which EAs have had much impact, we can list aeronautical and automotive design [48], bioinformatics and biotechnology [49], chemical engineering [50], creative pursuits [51], finance and investment [52], manufacturing [53], and structural design [54]. To select a small number of case studies could not serve to characterize the true diversity of applied EAs. We therefore duck that challenge and take the liberty of concluding this chapter by providing two case studies from the authors' recent work, illustrating how some EA approaches are being applied to diverse and challenging optimization problems in just one corner of science.

Using Niching and Coevolution to Understand Gene Regulation

Understanding gene expression is fundamental to understanding living processes. The expression of each gene in an organism is determined by the binding of special proteins, called transcription factors, within a region of DNA upstream of its coding region. In higher organisms, such as humans, this regulatory region typically contains around 5–10 transcription factor binding sites. Characterizing these bindings sites, both individually and in combination, is a fundamental part

of reconstructing (and ultimately controlling) the genetic networks that underlie biological function.

Identifying binding sites is often reduced to an optimization problem that involves constructing a matrix model of the occurrence of each DNA base at each position within a short region of DNA. Candidate solutions to this problem can be evaluated by scanning them along the regulatory regions of groups of genes which are known to be expressed at a certain time or within a certain cell type, looking for matches to patterns embedded in the sequences. In most cases, this is a multimodal problem, since multiple binding sites are likely to be relevant to a particular regulatory context, and it is important to be able to identify these different optima. However, these binding sites can vary quite considerably in their degree of conservation, meaning that the objective values of different optima also vary considerably.

Identifying and preserving different modes within a multimodal search space is a challenging problem, especially when they have different relative finesses. In section "Common Variants", we discussed the idea of niching within the populations of EAs as a means of addressing multimodal problems. In [14] we used an EA furnished with a particular form of niching, termed population clustering, that uses a clustering algorithm to identify and preserve the different modes present within an evolving population of solutions. This allowed a number of different binding sites to be characterized and preserved during a single run. Compared to other forms of niching, it also had the benefit of explicitly identifying these different groups of solutions, allowing the progress of search to be visualized and for clustering parameters to be dynamically modified by an expert user. Even in the absence of dynamic modification, however, this was effective at identifying the clusters of binding sites within comparatively long regions of DNA.

Identifying binding sites is only one part of the problem. Another important aim is to understand the interactions between different binding sites during gene regulation. In [55], we used coevolution to explore solutions to this problem. This involved coevolving two populations, the first containing matrix models of binding sites and the second containing Boolean expressions describing their co-occurrence within binding regions. Members of the binding site population were used as leaves within the Boolean expressions. In essence, the problem of identifying binding sites and their co-expression was decomposed into two problems which were then solved in parallel, using coevolution to provide feedback between the populations. In comparison to a more traditional approach, which would involve sequentially learning binding sites and then learning their interactions, this allows search to be directed toward solutions that interact well with other solutions during the course of search. Such solutions, in turn, are more likely to be meaningful. This approach proved effective for reverse engineering the regulatory rules underlying differential gene expression within tissues. More surprisingly, it also provided a mechanism for solving harder instances of the single binding site optimization problem, with coevolution provided a means of implicitly decomposing the matrices associated with these harder problems.

Evolving Classifiers for Parkinson's Disease Diagnosis

This second case study concerns a problem in which the optimal representation for solutions is not clear in advance, requiring experimentation with different kinds of solution representation. As discussed earlier, EAs are entirely flexible in this regard. The problem involves building diagnostic classifiers for Parkinson's disease. These are required to reach their decision based on time series movement data recorded while patients and age-matched control were undergoing clinical assessments of motor function. Motor aspects of Parkinson's are incompletely understood, making it unclear what kind of features of the data are important.

To address this, we used EAs to explore two different relatively unconstrained classifier models. Initially we considered a GP-based approach, using it to discover mathematical expressions that describe overrepresented patterns of movement embedded in short segments of the time series data, i.e., a form of symbolic regression [56]. An advantage of this approach is that the resulting expressions were relatively interpretable, allowing us to gain insight into the basis of classifications and then pass this information on to our clinical partners. In particular, analysis of the evolved expressions identified specific aspects of the closing phase of a "finger tap" movement as highly discriminatory of Parkinson's disease patients versus control. These factors alone were indeed more discriminatory than standard metrics; however, overall classification performance was not quite as good as that of trained clinicians.

We then considered a more unusual method of representing programs, artificial biochemical networks [57]. These are abstract executable models of the networks of biochemical interactions that underlie the function of biological cells. In a nutshell, they attempt to capture the representation which biological evolution has selected to optimize complex behaviors, with the hypothesis that this makes them particularly suitable for use with EAs. Meanwhile, the use of an EA to search through the space of possible artificial biochemical network classifiers represents, in itself, a major and commonly understood strength of EAs: they can be tailored and deployed effectively with relative ease despite the complexity and diversity of the structure they are being used to evolve. By using this approach, we were now able to find classifiers that produced comparable performance to that of trained clinicians [58]; with accuracy at around 90% overall, accuracy was comparable to the diagnostic accuracies found in clinical diagnosis, and significantly higher than those found in primary and non-expert secondary care.

Conclusion

In this chapter we have provided a broad introduction to and overview of evolutionary algorithms, and the many varieties of them that appear in modern research and practice. We have seen that there are a handful of key EA 'families', such as 'genetic algorithms' and 'differential evolution'; meanwhile, to some extent cutting across

the principal types of EAs, there are several algorithm features and variants (such as co-evolution, or hybridisation with local search) that are often 'grafted' into an EA, in order to boost performance on a particular class of problems (or perhaps just for research purposes). Towards the end, we considered the real-world application of EAs, reporting that they are generally highly successful, and are growing to be common tools used for optimisation tasks in science and industry. However, for someone new to EAs, an early question might well be: "There are so many types and variants of EAs – which should I use?" In this chapter, we have effectively promised that EAs are likely to be a successful approach to any given optimization task – particularly if that task is not well served by other available schemes or heuristics. However, we have not necessarily made it easy for a novice to deliver on that promise. A common theme has been the very wide variety of EA designs that have been developed, and this great variety can be a barrier for those new to the field, who aim to understand the key aspects of EAs and how best to apply them to any given task. In closing, we attempt to provide some support for the novice via the following perspective: EAs should not be seen as a collection of algorithms; instead, they are far more fruitfully viewed as an approach to the engineering of problem-specific (or problem-class specific) optimization algorithms. The key lesson for practitioners, from half a century of research and practice in EAs, is, broadly speaking, that concepts from evolution (population, selection, variation) can be very powerful tools in optimization; but to achieve ideal performance on a given task, one must be pragmatic, and often creative, about the details.

Cross-References

- ▸ A History of Metaheuristics
- ▸ Evolution Strategies
- ▸ Evolutionary Algorithms for the Inverse Protein Folding Problem
- ▸ Genetic Algorithms
- ▸ Hyper-heuristics
- ▸ Memetic Algorithms
- ▸ Multi-objective Optimization
- ▸ Particle Swarm Methods

References

1. Lones MA (2014) Metaheuristics in nature-inspired algorithms. In: Proceedings of genetic and evolutionary computation conference (GECCO 2014), workshop on metaheuristic design patterns (MetaDeeP). ACM, pp 1419–1422
2. Fogel DB (1998) Evolutionary computation: the fossil record. Wiley-IEEE Press, Piscataway
3. Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm Evol Comput 1(3):111–128. https://doi.org/10.1016/j.swevo.2011.08.003. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000435

4. Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31. https://doi.org/10.1109/TEVC.2010.2059031. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5601760&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs~all.jsp%3Farnumber%3D5601760
5. Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359. https://doi.org/1008202821328. Available: http://link.springer.com/article/10.1023%2FA%3A1008202821328#page-1.
6. Zaharie D (2009) Influence of crossover on the behavior of differential evolution algorithms. Appl Soft Comput 9(3):1126–1138. https://doi.org/10.1016/j.asoc.2009.02.012. Available: http://www.sciencedirect.com/science/article/pii/S1568494609000325
7. García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heuristics 15(6):617–644. https://doi.org/10.1007/s10732-008-9080-4. Available: http://link.springer.com/article/10.1007/s10732-008-9080-4
8. Hansen N, Auger A, Finck S, Ros R (2010) Real-parameter black-box optimization benchmarking 2010: experimental setup. INRIA research report No. 7215. INRIA
9. Liang J, Qu B, Suganthan P, Hernández-Díaz A (2013) Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical report 201212. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, pp 3–18
10. Tang K, Li X, Suganthan PN, Yang Z, Weise T (2009) Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report. Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China
11. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893. Available: http://ieeexplore.ieee.org/xpls/abs~all.jsp?arnumber=585893
12. Igel C, Toussaint M (2003) On classes of functions for which no free lunch results hold. Inf Process Lett 86(6):317–321
13. Piotrowski AP (2015) Regarding the rankings of optimization heuristics based on artificially-constructed benchmark functions. Inf Sci 297:191–201. Available: http://www.sciencedirect.com/science/article/pii/S0020025514010937
14. Lones MA, Tyrrell AM (2007) Regulatory motif discovery using a population clustering evolutionary algorithm. IEEE/ACM Trans Comput Biol Bioinform 4(3):403–414. https://doi.org/10.1109/tcbb.2007.1044. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4288066
15. Koza J (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
16. Miller JF (2011) Cartesian genetic programming. https://doi.org/10.1007/978-3-642-17310-3_2
17. Veenhuis CB (2009) Tree based differential evolution. Lect Notes Comput Sci 5481:208–219
18. Kim K, Shan Y, Nguyen X, McKay RI (2014) Probabilistic model building in genetic programming: a critical review. Genet Program Evolvable Mach 15(2):115–167. https://doi.org/10.1007/s10710-013-9205-x. Available: http://link.springer.com/article/10.1007/s10710-013-9205-x
19. Poli R, Langdon W, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com
20. Luke S (2013) Essentials of metaheuristics. Published via http://lulu.com
21. Stanley KO, Miikkulainen R (2003) A taxonomy for artificial embryogeny. Artif Life 9(2):93–130. https://doi.org/10.1162/106454603322221487. Available: http://www.mitpressjournals.org/doi/abs/10.1162/106454603322221487 (pages 94 and 95)

22. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1(1):47–62. https://doi.org/10.1007/s12065-007-0002-4. Available: http://link.springer.com/article/10.1007/s12065-007-0002-4
23. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech concurrent computation program, C3P report 826
24. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol Comput 2:1–14. https://doi.org/10.1016/j.swevo.2011.11.003. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000691
25. Hao J (2012) Memetic algorithms in discrete optimization. In: Neri F, Cotta C, Moscato P (eds) Handbook of memetic algorithms. Springer, Berlin/Heidelberg. https://doi.org/10.1007/978-3-642-23247-3_6
26. Ross P (2005) Hyper-heuristics. In: Search methodologies. Springer, Berlin, pp 529–556
27. Singh G, Deb K (2006) Comparison of multi-modal optimization algorithms based on evolutionary algorithms. ACM, New York. https://doi.org/10.1145/1143997.1144200
28. Mengshoel OJ, Goldberg DE (2008) The crowding approach to niching in genetic algorithms. Evol Comput 16(3):315–354. https://doi.org/10.1162/evco.2008.16.3.315. Available: http://www.mitpressjournals.org/doi/abs/10.1162/evco.2008.16.3.315
29. Sareni B, Krahenbuhl L (1998) Fitness sharing and niching methods revisited. IEEE Trans Evol Comput 2(3):97–106. https://doi.org/10.1109/4235.735432. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=735432&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel4%2F4235%2F15834%2F00735432.pdf%3Farnumber%3D735432
30. Lim T (2014) Structured population genetic algorithms: a literature survey. Artif Intell Rev 41(3):385–399. https://doi.org/10.1007/s10462-012-9314-6. Available: http://link.springer.com/article/10.1007%2Fs10462-012-9314-6
31. Shir OM, Back T (2005) Dynamic niching in evolution strategies with covariance matrix adaptation. https://doi.org/10.1109/CEC.2005.1555018
32. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
33. Knowles J, Corne D (1999) The pareto archived evolution strategy: a new baseline algorithm for paretomultiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation (CEC'99), vol 1. IEEE
34. Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
35. Corne DW, Deb K, Fleming PJ, Knowles JD (2003) The good of the many outweighs the good of the one: evolutionary multi-objective optimization. IEEE Connect Newslett 1(1):9–13
36. Zhou A, Qu B, Li H, Zhao S, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm Evol Comput 1(1):32–49. https://doi.org/10.1016/j.swevo.2011.03.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000058
37. Goldberg D, Smith R (1987) Nonstationary function optimization using genetic algorithm with dominance and diploidy. In: Proceedings of the second international conference on genetic algorithms and their application (ICGA). Laurence Erlbaum Associates, pp 59–68
38. Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol Comput 6:1–24. https://doi.org/10.1016/j.swevo.2012.05.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650212000363
39. Popovici E, Bucci A, Wiegand RP, De Jong ED (2012) Coevolutionary principles. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Heidelberg. https://doi.org/10.1007/978-3-540-92910-9_31
40. Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D Nonlinear Phenom 42(1–3):228–234. https://doi.org/10.1016/0167-2789(90)90076-2. Available: http://www.sciencedirect.com/science/article/pii/0167278990900762

41. Potter MA, Jong KA (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29. https://doi.org/10.1162/106365600568086. Available: http://www.mitpressjournals.org/doi/abs/10.1162/106365600568086
42. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Inf Sci 178(15):2985–2999. https://doi.org/10.1016/j.ins.2008.02.017. Available: http://www.sciencedirect.com/science/article/pii/S002002550800073X
43. Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. J Artif Evol Appl 2009:1–25
44. Ochoa G, Harvey I, Buxton H (1999) On recombination and optimal mutation rates. In: Proceedings of genetic and evolutionary computation conference, vol 1, pp 488–495. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.2369
45. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31. https://doi.org/10.1016/j.swevo.2011.02.001. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000022
46. Fogel LJ (1962) Autonomous automata. Ind Res 4(2):14–19
47. Ochoa G, Blum C, Chicano F (2015) Evolutionary computation in combinatorial optimization. Springer International Publishing: Imprint: Springer, Cham
48. Bajpai RP (ed) (2014) Innovative design, analysis and development practices in aerospace and automotive engineering: I-Dad 2014, 22–24 Feb 2014. Springer Science & Business, Singapore
49. Gaurav A, Kumar V, Nigam D (2012) New applications of soft computing in bioinformatics: a review. J Pure Appl Sci Tech 11(1):12–22
50. Gupta SK, Ramteke M (2014) Applications of genetic algorithms in chemical engineering II: case studies. In: Applications of metaheuristics in process engineering. Springer, Cham, pp 61–87
51. Bentley P, Corne D (2002) Creative evolutionary systems. Morgan Kaufmann, San Francisco
52. Chen SH (ed) (2012) Genetic algorithms and genetic programming in computational finance. Springer Science & Business Media, New York
53. Gen M, Cheng R (1996) Genetic algorithms and manufacturing systems design, 1st edn. Wiley, New York
54. Adeli H, Sarma KC (2006) Cost optimization of structures: fuzzy logic, genetic algorithms, and parallel computing. Wiley, Chichester
55. Lones MA, Tyrrell AM (2007) A co-evolutionary framework for regulatory motif discovery. https://doi.org/10.1109/CEC.2007.4424978
56. Lones M, Alty JE, Lacy SE, Jamieson DR, Possin KL, Schuff N, Smith SL (2013) Evolving classifiers to inform clinical assessment of parkinson's disease. In: 2013 IEEE symposium on computational intelligence in healthcare and e-health (CICARE), pp. 76–82. IEEE
57. Lones M, Turner AP, Caves LS, Stepney S, Smith SL, Tyrrell AM (2014) Artificial biochemical networks: evolving dynamical systems to control dynamical systems. IEEE Trans Evol Comput 18(2):145–166
58. Lones MA, Smith SL, Tyrrell AM, Alty JE, Jamieson DS (2013) Characterising neurological time series data using biologically motivated networks of coupled discrete maps. BioSystems 112(2):94–101

Genetic Algorithms

15

Carlos García-Martínez, Francisco J. Rodriguez, and Manuel Lozano

Contents

Introduction . . . 432
GA Construction . . . 434
Representation . . . 435
Initial Population: Generating Candidate Solution from Scratch . . . 436
Evaluation . . . 437
Selection Operator . . . 438
Genetic Operators . . . 438
Evolution Model and Replacement Strategy . . . 442
Stop Condition . . . 442
Diversification Techniques for GAs . . . 445
Methods to Generate Raw Diversity . . . 445
Strategies to Maintain Diversity . . . 446
The Crossover Operator as Diversification Agent . . . 448
Diversification by Adapting GA Control Parameters . . . 449
Diversity Preservation Based on Spatial Separation . . . 450
Hybrid GAs . . . 451
Collaborative Teamwork Hybrid GAs . . . 453
Collaborative Relay Hybrid GAs . . . 453
Integrative Teamwork Hybrid GAs . . . 454
Integrative Relay Hybrid GAs . . . 455
GA Applications . . . 455
Conclusions . . . 458

C. García-Martínez (✉)
Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain
e-mail: cgarcia@uco.es

F. J. Rodriguez and M. Lozano
Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
e-mail: fjrodriguez@decsai.ugr.es; lozano@decsai.ugr.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_28

Cross-References 458
References 459

Abstract

This chapter presents the fundamental concepts of genetic algorithms (GAs) that have become an essential tool for solving optimization problems in a wide variety of fields. The first part of this chapter is devoted to the revision of the basic components for the design of GAs. We illustrate this construction process through its application for solving three widely known optimization problems as knapsack problem, traveling salesman problem, and real-parameter optimization. The second part of the chapter focuses on the study of diversification techniques that represent a fundamental issue in order to achieve an effective search in GAs. In fact, analyzing its diversity has led to the presentation of numerous GA models in the literature. Similarly, the hybridization with other metaheuristics and optimization methods has become a very fruitful research area. The third part of the chapter is dedicated to the study of these hybrid methods. In closing, in the fourth part, we outline the wide spectrum of application areas that shows the level of maturity and the wide research community of the GA field.

Keywords

Genetic Algorithms · basic components · GA design · population diversity · diversity maintenance · diversity generation · hybrid genetic algorithms

Introduction

Genetic algorithms, or GAs, are nowadays an important field in artificial intelligence and operations research, with more than 2,500 publications per year in the last 10 years. Contrary to their most common type of application, GAs were not initially presented in the 1970s for problem solving, but as an instrument for simulating the biological evolution of adaptive natural systems [58]. Nevertheless, it was precisely the simulated adaptive ability that promoted, in the subsequent years, the opportunity for tackling optimization problems successfully [26, 27, 43]. In this chapter, we focus on their utility for problem solving, so GAs can operationally be described as iterative algorithms that sample candidate solutions for a given problem, with the aim of reaching the best configurations.

GAs take their inspiration from the biological evolution of species. Thus, the following ideas belong to their core foundations:

- Species evolve in populations of individuals that compete for survival within an environment.
- New individuals appear because of the recombination of previous individuals.
- Darwin's theory of *natural selection* establishes that the fittest individuals, those better adapted to the environment, get higher chances of surviving and breeding.

Thus, less fitted individuals may probably perish at some point of the evolution process.
- Individuals' adaptation to the environment strongly depends on their phenotypic characteristics, which, in turn, are principally determined by their genotype.
- Individuals might eventually suffer mutations at their genotype level, which may affect their phenotype and adequacy to the environment.

Using these ideas, the following principles govern most of GAs for optimization:

- The environment is defined by the problem to be addressed.
- Individuals, also known as *chromosomes*, represent candidate solutions for the problem.
- Their *genotypes* encode the candidate solutions for the problem. The genotype-phenotype translation establishes how the chromosomes should be interpreted to get the actual candidate solutions.
- The *fitness* of individuals depends on their adequacy on the given problem, so fitter individuals are more probably to survive and breed.
- There is an evolving *population of individuals*, where new individuals may get into and other may perish.
- New individuals are generated as the consequence of the *recombination* and/or the *mutation* of previous ones.

It should be pointed out that similar approaches were being developed since the 1960s, and, thus, there are nowadays several optimization methods sharing a reasonable resemblance. All these approaches have been gathered up under the term *evolutionary algorithms* [29, 37], pointing out the fact that these strategies work with candidate solutions which evolve toward better configurations. However, a distinguishing feature of GAs at the time was the concept of recombination implemented as a crossover operator. This, in contrast with the neighborhood exploration of just one solution, carried out by the existing heuristics and by the operators of other algorithms, introduced the idea of combining the information from two candidate solutions to generate a new one. Years afterward, the idea of recombining two or more solutions was adopted by most representatives of evolutionary computation.

The main aim of this chapter is to introduce the general and practical guidelines for designing GAs for given problems. Thus, we will gather up classical ideas from their origins and the latest tendencies without going into the specific and theoretical details. The interested reader is referred to [95] for a survey of the works developed in this regard.

The chapter is organized as follows. GA Construction Section offers several hints to newcomers for the design and implementation of a GA. Three different and widely known classes of optimization problems are used for reviewing the basic GA components, and several classic alternatives are provided for each of them. Diversification Techniques for GAs Section analyzes the premature convergence problem of GAs produced by the lack of diversity and presents different methods and operators

that attempt to favor the diversity of the population in GAs. Hybrid GAs Section is devoted to another research area with an important number of algorithms reported in the last years and that is intended to exploit the complementary character of different optimization strategies. Finally, GA Applications Section outlines the application areas of GAs and the major challenges faced by GAs to deal with optimization problems in these areas.

GA Construction

In this section, we revise the basic ingredients for designing a GA. We comment some of the original and widely used strategies for each of these components without getting into a deep discussion whether they perform better or worse than other approaches.

Given an optimization problem with a set of decision variables arranged in a vector $\mathbf{x}$, let $f : X \rightarrow \Re$ be a function that maps any configuration of the decision variables to a quality value. The aim is to find the combination of values for the decision variables that maximizes, or minimizes, function f. This function is usually known as the objective function. We assume that function f is to be maximized for the rest of the chapter, being trivial most of the needed modifications for minimization problems. In some cases, there are additional restrictions that solutions must satisfy, which are modeled by a set of inequality and equality constraints, $g_j(\mathbf{x}) \leq 0$ and $h_j(\mathbf{x}) = 0$ with $j = 1, \ldots, M$, respectively. We will use the following widely known optimization problems throughout the chapter to illustrate the application of GAs to problems with very different characteristics:

Knapsack Problem: In the classic knapsack problem [22], we are given a set of N objects with profits $p_i \geq 0$ and weights $w_i \geq 0$ and a container, the knapsack, with a maximum capacity C. The goal is to decide which objects should be included in the knapsack in such a way that the sum of the profits of the selected objects is maximized and the capacity C is not exceeded by the sum of their respective weights. Note that the decision variables of this problem should indicate which objects are included into the knapsack and which are not. Likewise, given a valid candidate solution that satisfies the capacity constraint ($g(\mathbf{x}) = -C + \sum_{i \in \text{Knapsack}} w_i$), the objective function f is the sum of the profits p_i of those objects in the knapsack.

Traveling Salesman Problem: This problem [50] consists in the design of travel plans visiting a given set of cities or locations so that the traveled distance is minimized. The information given is the set of cities, all of them have to be visited, and the distances between each pair of cities. In this case, decision variables should represent the order in which the cities are visited, and the objective function is the sum of the distance between consecutive cities plus the journey back to the first city from the last one.

Real-Parameter Optimization: Here, we are given an objective function whose decision variables take values from the continuous space $\Re$. In the context of optimization with metaheuristics, this objective function is treated as a *black box* so analytical methods cannot be applied. This is often the case when the evaluation of candidate solutions involves the execution of an external simulation procedure, such as wind tunnels or driving simulators, whose internal processes are obscure or they are not subject of analysis. The distinguishing feature of these problems, with regard to combinatorial problem, is that decision variables take real values instead of binary or integer ones.

Whichever is the problem, the design of a GA goes throughout the following steps and not necessarily in this order:

Representation

The practitioner needs to specify how the decision variables of the problem are represented in the computer program to allow the GA to generate the different possible configurations. In particular, the GA community stresses the difference between the solution representation, *genotype*, from arrays of binary, integer (or elements from an alphabet), or floating-point variables to complex structures such as trees, graphs, or objects of specifically designed classes and the solution which is actually represented, *phenotype*. Consider the previous optimization problems:

Knapsack Problem – Representation: The classic representation for this problem is the *binary coding*, which is actually the traditional representation in GAs regardless the problem. Solutions are represented by an array of N binary variables, as much as objects in the problem, where the value 1 at the i-th position indicates that the i-th object should be included into the knapsack and 0 means the opposite. Therefore, the following example indicates that we should select the first, third, fourth, and seventh objects and discard the others.

$$[\,1\,0\,1\,1\,0\,0\,1\,]$$

Here, the array of binary variables is the genotype, and the interpretation of that genotype, the actual configuration of the knapsack, is the phenotype. However, note that we could have adopted a different representation such as a variable-length array or set with the integer indexes of the included objects ([1 3 4 7] in the case above). We assume from now on that binary coding is adopted for this problem.

Traveling Salesman Problem – Representation: Given that the important information here is the order in which cities are visited, a natural way to represent solutions is by means of *permutations* [72], such as the following one:

$$[\,3\,4\,5\,7\,2\,6\,1\,]$$

These permutations are the genotype from which different interpretations could be developed, the phenotype. The most commonly used is the one that establishes traveling relations between consecutive values, i.e., in the example above, the agent travels from the third city to the fourth one, from the fourth to the fifth, and so on; finally, the agent returns to the third city from the first one (last value in the solution). As previously, other coding schemes could be adopted, such as the binary encoding of each of the integers in the example ([011 100 101 111 010 110 001]). Particularly, an indirect technique with relative success in the field is the *random key encoding* [48]. This represents a permutation as the sorting order of an array of numbers. Therefore, the array [3.0 4.1 0.2 0.3 −0.2 7 1] would be first translated into the permutation [5 3 4 7 1 2 6], from which the appropriate interpretation should be extracted. For the rest of the chapter, we adopt the former representation, which indicates the order by which cities are visited.

Real-Parameter Optimization – Representation: Initially, real (and integer) optimization problems were addressed with the binary coding scheme, considering either a natural or a gray transformation [16,43,58]. This latter intends to minimize the number of hamming differences between close final interpretations. However, real encoding is much widely used lately for real-parameter optimization problems, which avoids the differentiation between genotype and phenotype [55,61,81]:

$$[\,0.5 \;\; 0.0001 \;\; 0.2342 \;\; 1.0 \;\; 100.0 \;\; 0.7 \;\; -7.2\,]$$

Initial Population: Generating Candidate Solution from Scratch

At the beginning of the search process, GAs conform an initial population of candidate solutions. Usually, there is not much information to be exploited to generate these candidate solutions and the following alternatives are commonly applied:

- *Random solutions*: When there is no other information apart from the considered representation (binary, integer, permutations, real, trees, etc.), one of the easiest and most widely used methodologies is to generate the initial population at random. Thus, the task of generating the initial population becomes the one of producing arrays of random binary, integer, or floating-point numbers, trees, graphs, or a mixture of them.
- *Diverse solutions*: As we will see later, the practitioner is often concerned with the presence of diverse solutions in the population. For the case of the initial population, many works recommend random procedures that incorporate some bias for generating diverse solutions and, thus, covering the search more uniformly [91]. One simple technique is a generalization of the Latin hypercube [60] which works as follows. Suppose that N initial solutions should be generated, whose genotype is an array with l variables that take values from an m-ary alphabet. First, l random permutations of $\{1, \ldots, N\}$ are composed.

Table 1 Latin hypercube sampling with $N = 6$, $m = 3$, and $l = 7$

Individual	P_1	P_2	P_3	P_4	P_5	P_6	P_7	Genotype
I_1	2	5	5	6	2	3	2	[3 3 3 1 3 1 3]
I_2	5	2	2	4	1	6	6	[3 3 3 2 2 1 1]
I_3	6	3	6	2	5	4	4	[1 1 1 3 3 2 2]
I_4	4	1	1	3	4	2	5	[2 2 2 1 2 3 3]
I_5	1	4	4	5	6	5	1	[2 2 2 3 1 3 2]
I_6	3	6	3	1	3	1	3	[1 1 1 2 1 2 1]

Then, solutions are generated concatenating the corresponding values of the permutations modulo m (plus 1 if needed), i.e., the first solution considers the first value of the permutations and so on. See an example in Table 1. Note that N should be a multiple of m to assure a uniform sample of the m values. In case of using a real encoding scheme, one possibility is to divide the domain of the variables in m disjoint intervals. The previous procedure is applied and the variable finally takes a uniform random value from the selected interval.

- *Heuristic solutions*: In case you have some information of the problem at hand, you may exploit it to sample initial solutions better than those randomly generated. For example, in the knapsack problem, objects with better weight-profit ratios can be given higher probabilities of being included into the knapsack; or in the traveling salesman problem, the nearest neighbor heuristic can be randomized to generate different solutions [115].

In [65], different population initialization techniques are reviewed and categorized into a taxonomy.

Evaluation

Solutions are evaluated to decide whether they will, or will not, take part in the evolution process. This means assigning a fitness value to these candidate solutions. The most common practice is to assign a fitness value which is computed exactly as, or directly from, the objective function $f(\cdot)$. For example, the sum of the profits of the included objects can be used in the knapsack problem or the traveled distance, in the traveling salesman problem. In some other occasions, you may be interested in another computation such as a scaling procedure to avoid the influence of the scale of the function, or including other information such as diversity maintenance or constraint violation, often addressed with penalty functions [100, 118].

A very different strategy is to consider a relative measure such as the ranking of the chromosomes in fitness order [94, 113]. Though there is information loss, a clear advantage is that fitness values do not depend on the scale of the objective function.

Selection Operator

This operator is traditionally the only one in charge of assigning more reproduction possibilities to better solutions and, therefore, biasing the search process toward the pursued objective. GAs apply this operator to select those individuals allowed to breed and generate new candidate solutions.

The first selection operator applied is known as the *roulette-wheel* method. The idea is to assign selection probabilities, or roulette portions, to the solutions in the population, which are proportional to their fitness values. Thus, better individuals get higher probabilities of being selected. Then, the roulette is drawn as many times as the number of individuals needed. An enhanced method is the *stochastic universal selection* of Baker [9]. In this case, the roulette has multiple equally spaced spinners, as much as individuals need to be selected, and is drawn just once. In contrast to roulette wheel, Baker's method reduces the stochastic possibility of selecting the best individual too many or not enough times.

Tournament is another selection operator widely applied nowadays because of its simplicity [47]. This method composes a set of t_{size} random individuals from the population, and the best one is selected for breeding. Multiple tournaments are simulated to get the parent population. Note that parameter t_{size} controls the bias toward the best solutions, being $t_{\text{size}} = 2$ a classic setting, i.e., binary tournament.

Genetic Operators

Crossover and mutation operators are the GA components that generate new candidate solutions, which are afterward evaluated and considered for the subsequent evolution iteration. Between these two, the crossover has traditionally got a superior significance, and mutation often acts in the background, being applied less frequently. The goal of the former is to combine the information of two or more parent chromosomes from the current population, while the second usually modifies a chosen chromosome random and locally.

There is not a clear methodology for the order in which these operators should be applied. Commonly, two GA parameters govern the frequency of crossover and mutation applications, namely, crossover and mutation probabilities, and the process is often carried as follows. The selected parent solutions firstly undergo crossover according to the crossover probability, and offspring is transferred to the mutation step; parents are otherwise transferred to the mutation phase. Subsequently, mutation is applied on the received solutions according to the mutation probability, and the result is temporarily stored for the next generation.

Given the strong interaction between the solution representation and genetic operators, we revise some classical proposals in the context of previous problems.

Crossover Operator

We show in this section some examples for combining the information from more than one parent through crossover application.

Table 2 Example of binary crossover operators with parents $P_1 = [\,0\,1\,1\,0\,1\,0\,0\,]$ and $P_2 = [\,1\,1\,1\,1\,0\,0\,1\,]$

Name	Intermediate steps			Result
One point Crossover	Cut point 4	Heads [0 1 1 0 [1 1 1 1	Tails 1 0 0] 0 0 1]	Offspring [0 1 1 0 0 0 1] [1 1 1 1 1 0 0]
Uniform Crossover		Genes from P_1 1, 3, 5, 6	Genes from P_2 2, 4, 7	Offspring [0 1 1 1 1 0 1]
Half uniform Crossover	Differences 1, 4, 5, 6	Genes from P_1 4, 5	Genes from P_2 1, 6	Offspring [1 1 1 0 1 0 1]

Knapsack Problem – Crossover Operator: The firstly proposed crossover operator took its inspiration from biology and is widely known as *one-point crossover* operator. Given two binary strings representing the corresponding parent solutions, this operator chooses a random position (4 in the example in Table 2) and combines the head of the first parent with the tail of the second one, generating a new candidate solution, and vice versa for a second candidate solution. However, other crossover operators such as uniform crossover [102] and half-uniform crossover [33] (see also The Crossover Operator as Diversification Agent Section) usually perform better. Both create a new offspring by selecting randomly each of the gene values from the parents, but the latter assures that half of the number of differences are taken from one parent and the other half from the other parent. In the particular case of the knapsack problem, the resulting solutions may violate the capacity constraint, so penalty functions or repair operators would be needed.

Traveling Salesman Problem – Crossover Operator: Problems using the permutation coding, such as this one, require crossover operators to accept permutations as input and produce valid permutations as output. Note that previous operators do not satisfy this necessity. Therefore, specific operators are proposed for this type of problems (see Table 3). The firstly proposed one was *partially mapped crossover* [45], which is aimed precisely at generating sequences of numbers without repetition. Its idea is to divide the two parent permutations in three segments each according to two randomly chosen positions (2–4 in Table 3). Then, the inner segment of one parent is combined with the outer ones of the other, and repeated genes in the outer segments are changed according to the mapping rules defined in the inner segment. *Order crossover* [23] is another option which intends to exploits the relative order of the gene values instead of their absolute positions, as previous operator does. Firstly sampling two random positions, this operator copies the inner segment into the offspring and continues copying those valid gene values from the secondly sampled position of the other parent (see Table 3). Given that different operators focus on different interpretations of the information transmitted, the practitioner should identify what is the one most appropriate for the adopted problem representation.

Table 3 Example of crossover operators for permutations with parents $P_1 = [\,1\,2\,3\,4\,5\,6\,7\,]$ and $P_2 = [\,2\,4\,5\,7\,1\,6\,3\,]$

Name	Intermediate steps			Result
Partially	Cross. points	Comb. and rep. removal	Mappings	Offspring
Mapped	2–4	[2 | 2 3 4 | 1 6 3]	2 → 4, 3 → 5, 4 → 7	[7 | 2 3 4 | 1 6 5]
Crossover		[1 | 4 5 7 | 5 6 7]	4 → 2, 5 → 3, 7 → 4	[1 | 4 5 7 | 3 6 2]
Order	Cross. points	Inner segment	Valid values	Offspring
Crossover	4–6	[− − − | 4 5 6 | −]	[2 4 | 5 7 1 6 | 3]	[2 7 1 | 4 5 6 | 3]
		[− − − | 7 1 6 | −]	[1 2 | 3 4 5 6 | 7]	[3 4 5 | 7 1 6 | 2]

Table 4 Example of real-parameter crossover operators with parents $P_1 = [\,0.3\;\;7\,]$ and $P_2 = [\,0.4\;\;1\,]$, $\alpha = 0.5$, and domains $x_1 \in [0, 1]$ and $x_2 \in [-10, 9]$

Name	Intermediate steps			
BLX-α	$I_i^a \mid p_i^1 - p_i^2 \mid$	Lower bound	Upper bound	Interval
		$\max(l_i, \min(p_i^1, p_i^2) - \alpha I_i^a)$	$\min(u_i, \max(p_i^1, p_i^2) + \alpha I_i^a)$	
o_1	0.1	$\max(0,\ 0.3 - 0.05)$	$\min(1,\ 0.4 + 0.05)$	[0.25, 0.45]
o_2	6	$\max(-10,\ 1 - 3)$	$\min(9,\ 7 + 3)$	[−2, 9]
PBX-α	$I_i^a \mid p_i^1 - p_i^2 \mid$	Lower bound	Upper bound	Interval
		$\max(l_i,\ p_i^1 - \alpha I_i^a)$	$\min(u_i,\ p_i^1 + \alpha I_i^a)$	
o_1	0.1	$\max(0,\ 0.3 - 0.05)$	$\min(1,\ 0.3 + 0.05)$	[0.25, 0.35]
o_2	6	$\max(-10,\ 1 - 3)$	$\min(9,\ 1 + 3)$	[−2, 4]

Real-Parameter Optimization – Crossover Operator: Similar to previous cases, there are many proposals for the combination of the information in two real-coded parent solutions. In this case, researchers pay attention to the capacities of the crossover to converge, expand, or even correlate the changes on the decision variables [56]. An interesting characteristic of most real-parameter crossover operator is whether they tend to sample new solutions near the centroid of the parents or near the parents. BLX-α [34] is an example of centroid-centered operator and PBX-α is its parent-centered version [39,76]. For each offspring gene o_i, both operators create an interval $I = [a, b]$ from which the actual value is uniformly and randomly drawn. In both cases, the amplitude of the interval depends on the difference between the parent gene values, $I_i^a = \text{abs}(p_i^1 - p_i^2)$, and the domain of the variable, $[l_i, u_i]$. The difference is the location of that interval, biased toward the average of previous parent gene values, in the case of BLX-α, or the value of the first parent, PBX-α. Table 4 shows an example of how both operators compute the intervals from where offspring gene values are sampled.

Mutation Operator

As for the crossover operator, mutation is strongly connected with the adopted representation. Therefore, we will see some simple examples for any of the three problems considered through this chapter. Apart, we shall mention that the

community usually distinguishes between applying mutation at individual or gene level. In the first case, the chromosome is mutated only once according to the mutation probability. One gene is randomly selected and modified, which could involve additional modifications in one or a few other genes. In the latter, all the genes of the chromosome are visited, and each one is mutated according to the mentioned probability, so chromosomes might suffer stronger alterations. In this case, the mutation probability is often much smaller.

Knapsack Problem – Mutation Operator: Flipping is the simplest mutation operator for binary strings. If mutation applies at the individual level, a random gene is selected and its value is changed, from 0 to 1 or the other way around. If mutation is at the gene level, each variable is visited once and flipped according to the mutation probability. One should notice that other operators might be more appropriate for the problem at hand. For example, in the case of the knapsack problem, flipping one bit from 0 to 1 might leave room for inserting another object in the knapsack, and changing one bit from 1 to 0 may make the solution unfeasible. Thus, you might prefer using another more specialized operator, such as flipping two bits with different values, instead of this general one.

Traveling Salesman Problem – Mutation Operator: Assigning random values to one or several genes is often applied in integer-coded problems; however, this strategy is not appropriate for permutation codings. Instead, mutation operators usually apply swappings, insertions, reorderings, or sublist inversions [29, 98]. Table 5 shows the application of swap [11, 88] and sublist inversion [58]. In both cases, two positions are randomly drawn. Notice, however, that sublist inversion modifies less arcs of the solution path in symmetric graphs than swapping the selected positions. This is due to the fact that traveling direction does not affect the solution cost in this kind of graphs. Thus, sublist inversion is often preferred for this problem.

Real-Parameter Optimization – Mutation Operator: Uniform and norm random variables, with a symmetric distribution around zero, are commonly used to perturb a given real-coded solution locally. In contrast to previous cases, it is usual to modify more than one and often all the variables of the solutions at the same time, by adding up a complete randomly drawn vector. Altering either one or all the variables, practitioners often pay attention at the global impact of the

Table 5 Example of mutation operators for permutations with chromosome $S = [\,2\,4\,5\,7\,1\,6\,3\,]$

Swap	Mut. points 2–5	Result [2 | 1 | 5 7 | 4 | 6 3]	Arcs removed $2 \to 4,\ 4 \to 5,$ $7 \to 1,\ 1 \to 6$	Added arcs $2 \to 1,\ 1 \to 5$ $7 \to 4,\ 4 \to 6$
Sublist Inversion	Mut. Points 2–5	Result [2 | 1 7 5 4 | 6 3]	Arcs removed $2 \to 4,\ 1 \to 6$	Added arcs $2 \to 1,\ 4 \to 6$

mutation so that the difference between the original and the mutated solution, under the Euclidean metric, for instance, is controlled. A widely used operator is the nonuniform mutation [61, 81], which reduces the intensity of mutation with the number of generations. Particularly, a variation vector $\Delta = [\ \delta_1\ \ldots\ \delta_i\ \ldots\ \delta_N\ ]$, computed as follows, is added up to the given solution $X = [\ x_1\ \ldots\ x_i\ \ldots\ x_L\ ]$:

$$\delta_i = \begin{cases} (u_i - x_i)(1 - z_i)^{\gamma}, & \text{with probability } 1/2 \\ (l_i - x_i)(1 - z_i)^{\gamma}, & \text{otherwise} \end{cases}$$

$$\gamma = \left(1 - \frac{t}{t_{\max}}\right)^{\beta}$$

where z_i is an uniform random variable in $[0, 1]$, u_i and l_i are the box bounds of variable x_i, β is a parameter with $\beta > 0$, and t and $t_{\max}$ are, respectively, the current and maximal number of generations.

Evolution Model and Replacement Strategy

The traditional GA applied a *generational* evolution strategy. This means that evolutions occur at generations where a complete new population is generated from the current one, which, in turn, becomes the current population for the next generation. The pseudocode of this generational GA is shown in Fig. 1. A possible undesirable effect of this model is that the new population might be worse, in quality terms, than the current population. So usually, the best solution from the current population is artificially inserted in the new population, particularly when there is not any better solution in the new population. This evolution model is known as *generational with elitism* [10].

Another commonly applied model is the *steady-state GA* [28], which contrarily generates a reduced set of new chromosomes, usually just one solution, that compete to enter into the population (see pseudocode in Fig. 2). In steady-state GAs, the designer has to specify a replacement policy, which determines whether the new solutions enter the population or are discarded and which solution from the population is removed to make room for the new one, if this is accepted. A simple policy to introduce the new solution into the population is replacing the worst solution if the new one is better. Other strategies consider diversity measures, like replacing the most similar solution [52] (see also Strategies to Maintain Diversity Section, crowding methods) or the worst parent.

Stop Condition

GAs evolve until a certain stopping criterion is met. Practitioners commonly consider a maximum number of generations, fitness evaluations, or a maximal processing time. These criteria are widely used when one is interested in comparing

Input:

NP: Population size

p_c: Crossover probability

p_m: Mutation probability

Output:

S: Best solution found

```
//Initial population
for i=1 to NP do
    p_i ← Generate solution;
    p_i.fitness ← Evaluate(p_i); //acc. to objective function f(·)
end
//Evolution
repeat
    P_0 ← ∅;
    //Offspring generation
    repeat
        parents ← Selection(P);
        offspring ← Crossover(parents); //acc. to p_c
        offspring ← Mutation(offspring); //acc. to p_m
        offspring.fitness ← Evaluate(offspring);
        P_0 = P_0 ∪ offspring;
    until |P_0| = NP;
    P ← P_0;
until Stop-condition is met;
return Best generated solution;
```

Fig. 1 Basic scheme of a generational GA

```
Input:
 NP: Population size
 pc: Crossover probability, usually equal to 1 in steady-state GAs
 pm: Mutation probability
Output:
 S: Best solution found
//Initial population
1  for i=1 to NP do
2      pi ← Generate solution;
3      pi.fitness ← Evaluate(pi); //acc. to objective function f(·)
4  end
//Evolution
5  repeat
6      parents ← Selection(P);
7      offspring ← Crossover(parents);
8      offspring ← Mutation(offspring); //acc. to pm
9      offspring.fitness ← Evaluate(offspring);
10     P ← Replacement(offspring, P);
11 until Stop-condition is met;
12 return Best generated solution;
```

Fig. 2 Skeleton of a steady-state GA

the performance of several algorithms, and the same computational resources have to be provided for a fair study.

On other occasions, especially when one is aimed at solving a particular problem, GAs are executed until a minimal solution quality is attained or the population converges so further progress is very difficult. In these cases, practitioners often need to be able to consult the progress of the search to decide at the moment, whether continuing the evolution or truncating it and applying a restart procedure (see also section "Methods to Generate Raw Diversity").

Diversification Techniques for GAs

There are two primary factors in the search carried out by a GA [20]: population diversity and selective pressure. In order to have an effective search, there must be a search criteria (the fitness function) and a selection pressure that gives individuals with higher fitness a higher chance of being selected for reproduction, mutation, and survival. Without selection pressure, the search process becomes random, and promising regions of the search space would not be favored over regions offering no promise. On the other hand, population diversity is crucial to a GAs ability to continue the fruitful exploration of the search space. If the lack of population diversity takes place too early, a premature stagnation of the search is caused. Under these circumstances, the search is likely to be trapped in a region not containing the global optimum. This problem, called premature convergence, has long been recognized as a serious failure mode for GAs [53, 119]. In the literature, many approaches have been proposed to introduce new methods and operators that attempt to favor the diversity of the population to overcome this essential problem of genetic algorithms. Next, we present a quick overview of them.

Methods to Generate Raw Diversity

Premature convergence causes a drop in the GAs efficiency; the genetic operators do not produce the feasible diversity to tackle new search space zones, and thus the algorithm reiterates over the known zones producing a slowing-down in the search process. Under these circumstances, resources may be wasted by the GA searching an area not containing a solution of sufficient quality, where any possible improvement in the solution quality is not justified by the resources used. Therefore, resources would be better utilized in restarting the search in a new area, with a new population. This is carried out by means of a restart operator [42, 44], which introduces chromosomes with high *raw diversity* to increase the average diversity level, thus to ensure the process can jump out the local optimum and to revolve again.

Eshelman's CHC algorithm [32] represents a GA with elitist selection and a highly disruptive recombination operator which restarts the search when the population diversity drops below a threshold level. The population is reinitialized by using the best individual found so far as a template for creating a new population. Each individual is created by flipping a fixed proportion (35%) of the bits of the template chosen at random without replacement. If several successive reinitializations fail to yield an improvement, the population is completely (100%) randomly reinitialized.

Another GA that utilizes population reinitialization is the micro GA [43]. In general, a micro GA is a small population GA which evolves for many generations. When after a number of generations the micro GA population converges, the evolutionary process is reinitialized by preserving the best individual and substituting the

rest of the population with randomly generated individuals. The first implementation of a micro GA was reported by Krishnakumar [69], who used a population size of five individuals, tournament selection, single-point crossover with probability, elitism, and restart operator. The population was considered converged when less than 5% of the population bits were different from the bits of the best individual.

A recent GA model, called saw-tooth GA [68], manages a variable population size with periodic reinitialization following a saw-tooth scheme with a specific amplitude and period of variation. In each period, the population size decreases linearly, and at the beginning of the next period, randomly generated individuals are appended to the population.

Strategies to Maintain Diversity

Pioneer works on the way diversity may be retained throughout the GA run focused on the design of alternative selection mechanisms. For example, in the linear ranking selection [8], the chromosomes are sorted in order of raw fitness, and then the selection probability of each chromosome is computed according to a linear function of its rank. With this selection mechanism, every individual receives an expected number of copies that depends on its rank, independent of the magnitude of its fitness. This may help prevent premature convergence by preventing super individuals from taking over the subpopulations within a few generations.

Disruptive selection [70] attempts to accomplish this objective as well. Unlike conventional selection mechanisms, this approach devotes more trials to both better and worse solutions than it does to moderate solutions. This is carried out by modifying the objective function of each chromosome, C, as follows: $f'(C) = |f(C) - \bar{f}|$, where $\bar{f}$ is the average value of the fitness function of the individuals in the population. A related selection method is the fitness uniform selection scheme (FUSS) [59]. FUSS generates selection pressure toward sparsely populated fitness regions, not necessarily toward higher fitness. It is defined as follows: if $f_{\min}$ and $f_{\max}$ are the lowest and highest fitness values in the current population, respectively, we select a fitness value uniformly in the interval $[f_{\min}, f_{\max}]$. Then, the individual in the population with fitness nearest to this value is selected. FUSS results in high selection pressure toward higher fitness if there are only a few fit individuals, and the selection pressure is automatically reduced when the number of fit individuals increases. In a typical FUSS population, there are many unfit and only a few fit individuals. Fit individuals are effectively favored until the population becomes fitness uniform. Occasionally, a new higher fitness level is discovered and occupied by a new individual, which then, again, is favored. Finally, another technique being worthy of mention is the repelling algorithm [114], which modifies the fitness function to increase the survival opportunity of chromosomes with rare alleles.

There are different replacement strategies for steady-state GAs that try to maintain population diversity as well. In [77], the authors propose a replacement strategy that considers two features of the individual to be included into the

population: a measure of the contribution of diversity to the population and the fitness function. It tries to replace a chromosome in the population with worst values for these two features. In this way, the diversity of the population increases and the quality of the solutions improves, simultaneously. The goal of this strategy is to protect those individuals that preserve the highest levels of useful diversity.

Other replacement methods to promote population diversity are the crowding methods [97]. They work as follows: new individuals are more likely to replace existing individuals in the parent population that are similar to themselves based on genotypic similarity. In this manner, the population does not build up an excess of similar solutions. Crowding methods promote the formation of stable subpopulations in the neighborhood of optimal solutions (niches), favoring the preservation of multiple local optima in multimodal problems. An effective crowding method is the restricted tournament selection (RTS) [52]. RTS initially selects two elements at random, A and B, from the population and perform crossover and mutation on these two elements resulting in a new element A'. Then, RTS scans ω (window size) more members of the population and picks the individual that most closely resembles A' from those ω elements. A' then competes with this element, and if A' wins, it is allowed to enter the population. Another type of crowding methods assumes that the parents would be those members of the population that are closer to the new elements. In this way, children compete with their parents to be included in the population, i.e., a family competition is held. These methods include deterministic crowding [78] and elitist recombination [106].

Some GA models were proposed in the literature that explicitly avoid the existence of duplicate individuals in the population, as a way to encourage diversity. In fact, early empirical studies confirmed that duplicate removal can enhance the performance of GA significantly [80]. For example, the non-revisiting GA [119] guarantees that no revisits ever occur in its fitness evaluations. It achieves this by interacting with a dynamically constructed binary space partitioning archive that is built up as a random tree for which its growth process reflects the evolution history of the GA and is a quick method to query whether there is a revisit. The entire previous search history is then used to guide the search to find the next unvisited position. A similar approach may be found in [51].

Finally, it is worth to mention that diploid GAs [67, 109] are evolutionary algorithms that manipulate a specific kind of diversity that becomes profitable to deal with dynamic optimization problems (in which some elements of the underlying model change over the course of the optimization). Most organisms in nature have a great number of genes in their chromosomes, and only some of the dominant genes are expressed in a particular environment. The repressed genes are considered as a means of storing additional information and providing a latent source of population diversity. Diploid GAs use diploid chromosomes which are different from natural ones in that the two strands of the diploid chromosomes are not complementary. Only some genes in a diploid chromosome are expressed and used for fitness evaluation by some predetermined dominance rules. Unused genes remain in the diploid genotype until they may later become useful (*latent diversity*).

The Crossover Operator as Diversification Agent

The mating selection mechanism determines the way the chromosomes are mated by applying the crossover to them. In the conventional GA, no mating strategy is applied to the results of selection; that is, parents are approved without any further examination after they are chosen at random or just by fitness. However, mates can be selected so as to favor population diversity [32, 107]. A way to do this is the negative assortative mating mechanism. Assortative mating is the natural occurrence of mating between individuals of similar genotype more or less often than expected by chance. Mating between individuals with similar genotype more often is called positive assortative mating and less often is called negative assortative mating. Fernandes et al. [35] assume these ideas in order to implement a parent selection mechanism for the crossover operator. A first parent is selected by the roulette wheel method, and n_{ass} chromosomes are selected with the same method. Then, the similarity between each of these chromosomes and the first parent is computed. If assortative mating is negative, then the one with less similarity is chosen. If it is positive, the genome that is most similar to the first parent is chosen to be the second parent. Clearly, the negative assortative mating mechanism increases genetic diversity in the population by mating dissimilar genomes with higher probability.

The crossover operator has always been regarded as one of the main search operators in GAs [66] because it exploits the available information in previous samples to influence future searches. This is why research has been focused on developing crossover operators with an active role as effective diversification agents. The half-uniform crossover used in the CHC algorithm [32] is a highly disruptive crossover that crosses over exactly half of the nonmatching alleles (the bits to be exchanged are chosen at random without replacement). This way, it guarantees that the two offspring are always at the maximum Hamming distance from their two parents, thus proposing the introduction of a high diversity in the new population and lessening the risk of premature convergence. It is worth of mention that CHC applies this operator along with (1) a reproduction restriction that assures that selected pairs of chromosomes would not generate offspring unless their Hamming distance is above a certain threshold and (2) a conservative selection strategy with high selective pressure (it keeps the N best elements appearing so far). In fact, Kemenade et al. [110] suggest that higher selection pressures allow the application of more disruptive recombination operators.

A crossover operator that was specifically designed with the aim of diversifying the search process of the real-coded GAs is BLX-α [34] (see also Crossover Operator section where its operation is described and, moreover, an illustrative example is presented). Nomura et al. [86] provide a formalization of this operator to analyze the relationship between the chromosome probability density functions before and after its application, assuming an infinite population. They state that BLX-α spreads the distribution of the chromosomes when $\alpha > \frac{\sqrt{3}-1}{2}$ or otherwise reduces it. This property was verified through simulations. In particular, the authors observed that BLX-0.0 makes the variances of the distribution of the chromosomes

decrease, reducing the distribution, whereas BLX-0.5 makes the variances of the distribution increase, spreading the distribution.

BLX-α has a self-adaptive nature in that it can generate offspring adaptively according to the distribution of parents without any adaptive parameter [12]. BLX-α uses probability distributions that are calculated according to the distance between the decision variables in the parents. If the parents are located closely to each other, the offspring generated by this operator might be distributed densely around the parents. On the other hand, if the parents are located far away from each other, then the offspring will be sparsely distributed around them. Therefore, it may fit their action range depending on the diversity of the population by using specific information held by the parents. In this way, depending on the current level of diversity in the population, it may favor the production of additional diversity (divergence) or the refinement of the solutions (convergence). This behavior is achieved without incurring into extra parameters or mechanisms to achieve the mentioned behavior.

Diversification by Adapting GA Control Parameters

Finding robust control parameter settings (such as mutation probability, crossover probability, and population size) is not a trivial task, since their interaction with GA performance is a complex relationship and the optimal ones are problem dependent. Furthermore, different control parameter values may be necessary during the course of a run to induce an optimal exploration/exploitation balance. For these reasons, adaptive GAs [30, 54, 63, 99, 108] have been built to dynamically adjust selected control parameters or genetic operators during the course of evolving a problem solution. Their objective is to offer the most appropriate exploration and exploitation behavior.

Some adaptive techniques were presented to endow the GA with useful diversity. Specifically, the mutation probability (p_m) was considered as a key parameter to accomplish this task [89, 119]. Next, we describe different adaptive mechanisms presented in the GA literature to control this parameter.

- *Deterministic control of* p_m. A direction followed by GA research for the variation of p_m lies in the specification of an externally specified schedule which modifies it depending on the time, measured by the number of generations. One of the most considered schedules consists in decreasing p_m during the GA run [36]. For example, [57] suggested the equation $p_m = 0.1 - 0.09 \cdot \frac{g}{G}$, where g is the generation number from 1 to G. This schedule follows the heuristic "to protect the exploration in the initial stages and the exploitation later," which has been considered to design other metaheuristics, such as simulated annealing.
- *Adaptive control of* p_m. In [101], a technique for the adaptive control at individual level of p_m was proposed, in which p_m is varied depending on the fitness values

of the solutions. Each chromosome C_i has its own associated p_m value, p_m^i, which is calculated as (maximization is assumed):

$$p_m^i = \frac{f_{\max} - f_i}{f_{\max} - \bar{f}} \text{ if } f_i \geq \bar{f}, \text{ and } p_m^i = 1 \text{ if } f_i < \bar{f},$$

where f_i is the chromosome's fitness, $f_{\max}$ is the population maximum fitness, and $\bar{f}$ is the mean fitness. In this way, high-fitness solutions are protected ($p_m^i = 0$), while solutions with subaverage fitnesses are totally disrupted ($p_m^i = 1$). This technique increases p_m when the population tends to get stuck at a local optimum and decreases it when the population is scattered in the solution space.

- *Self-adaptive control of* p_m. An extra gene, p_m^i, is added to the front of each chromosome, C_i, which represents the mutation probability for all the genes in this chromosome. This gene evolves with the solution [7,108]. The values of p_m^i are allowed to vary from p_m^l to p_m^h. The following steps are considered to mutate the genes in a chromosome C_i:

  1. Apply a meta-mutation on p_m^i obtaining q_m^i. This is carried out by choosing a randomly chosen number from the interval $[p_m^i - d, p_m^i + d]$, where d is a control parameter.
  2. Mutate the genes in C_i according to the mutation probability q_m^i.
  3. Write the mutated genes (including q_m^i value) back to the chromosome.

  Crossover is presently applied only to the chromosome and has no impact on p_m^i. Each offspring resulting from crossover receives the p_m^i value of one of its parents. The initial p_m^i values are generated at random from $[p_m^l, p_m^h]$.

  The self-adaptive control of GA parameters attempts to exploit the indirect link between favorable control parameter values and fitness values, with the parameters being capable of adapting implicitly, according to the topology of the objective function [7].

Diversity Preservation Based on Spatial Separation

GA models based on the spatial separation of individuals were considered as an important way to research into mechanisms for dealing with the premature convergence problem. One of the most important representatives are the distributed GAs (DGAs) [4,31,53]. Their premise lies in partitioning the population into several subpopulations, each one of them being processed by a GA, independently of the others. Furthermore, a migration mechanism produces a chromosome exchange between the subpopulations. DGAs attempt to overcome premature convergence by preserving diversity due to the semi-isolation of the subpopulations. Another important advantage is that they may be implemented easily on parallel hardware.

Moreover, we should also point out that it is possible to improve the diversification in DGAs by designing specific migration policies [6].

Making distinctions between the subpopulations of a DGA through the application of GAs with different configurations (control parameters, genetic operators, codings, etc.), we obtain the so-called heterogeneous DGAs [53]. They are suitable tools for producing parallel multiresolution in the search space associated with the elements that differentiate the GAs applied to the subpopulations. This means that the search occurs in multiple exploration and exploitation levels. In this way, a distributed search and an effective local tuning may be obtained simultaneously, which may allow premature convergence to be avoided and approximate final solutions to be reached. An outstanding example is GAMAS [90], a DGA based on binary coding that uses four subpopulations, denoted as species I–IV. Initially, species II–IV are created. Species II is a subpopulation used for exploration. For this purpose, it uses a high mutation probability ($p_m = 0.05$). Species IV is a subpopulation used for exploitation. So, its mutation probability is low ($p_m = 0.003$). Species III is an exploration and exploitation subpopulation; the mutation probability falls between the other two ($p_m = 0.005$). GAMAS selects the best individuals from species II–IV and introduces them into species I whenever those are better than the elements in this subpopulation. The mission of species I is to preserve the best chromosomes appearing in the other species. At predetermined generations, its chromosomes are reintroduced into species IV by replacing all of the current elements in this species.

The cellular GA (cGA) [3] is another kind of decentralized GA in which the population is arranged in a toroidal grid, usually of dimension two. The characteristic feature of cGAs is a neighborhood for each individual that is restricted to a certain subset of individuals in the immediate vicinity of its position. Individuals are allowed to interact only with other individuals belonging to their neighborhood. The overlapped small neighborhoods of cGAs help in exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration (diversification). In contrast, the exploitation (intensification) is provided inside each neighborhood to improve the quality of solutions and could be controlled by the use of appropriate genetic operators. Several studies have been carried out in order to investigate mechanisms to dynamically control the exploration/exploitation trade-off kept by cGAs. This task may be achieved through tuning the relationship between the size and/or shape of the neighborhood and the grid [2] or by adjusting the local selection method [5]. Finally, it is worth remarking that since cGAs only require communication between few closely arranged individuals, they are very suitable for a parallel implementation, as well.

Hybrid GAs

Over the last few years, a large number of search algorithms were reported that do not simply follow the paradigm of one single classical metaheuristic but they combine several components of different metaheuristics or even other

kind of optimization methods. These methods are commonly known as *hybrid metaheuristics* [14, 15, 49, 75, 103]. The main motivation behind the development of hybrid metaheuristics is to take advantage of the complementary character from a set of metaheuristics and other optimization methods to produce a *profitable synergy* [96] from their combination.

In particular, the *hybridization of GAs*, and evolutionary algorithms in general, is becoming popular due to its ability to handle several problems involving complex features such as complexity, noise, imprecision, uncertainty, and vagueness [49, 79, 92, 112]. We may also highlight that many different instantiations of hybrid GAs have been presented to solve real-world problems in many different fields such as protein structure prediction [105], image processing [13], job-shop scheduling [38], and machine learning [87], to name but a few.

The flexibility offered by the GA paradigm allows specialized models to be obtained with the aim of providing intensification and/or diversification, i.e., GAs specializing in intensification and/or diversification. The outstanding role played by GAs at present along with the great interest raised by their hybridizations with other algorithms endorse the choice of their specialist approaches as suitable ingredients to build hybrid metaheuristics with others search techniques [75].

The purpose of this section is to illustrate the different strategies used successfully in the literature to hybridize GAs and classify them according to a taxonomy based on those proposed by [103] and [93] for hybrid metaheuristics.

We firstly have split the different instances of hybrid metaheuristics into two main groups according to the architecture of the algorithms:

- *Collaborative hybrid metaheuristics*. In this category, different self-contained metaheuristics interchange information between them running sequentially or in parallel. These hybrid metaheuristics can be considered as black boxes, and the only cooperation takes place through the exchange of solutions, parameters, and so on from time to time.
- *Integrative hybrid metaheuristics*. In this case, one algorithm is in charge of the search process, whereas a subordinate method is embedded as a component of the master metaheuristic. This kind of hybridization addresses the functional composition of a single optimization method, replacing or improving a particular function within a metaheuristic with another metaheuristic.

At the same time, according to the way metaheuristics are executed, *collaborative* metaheuristics can be subdivided into two different categories:

- *Collaborative teamwork*. There are several metaheuristics that work in parallel and exchange some kind of information once in a while.
- *Collaborative relay*. Several metaheuristics are executed in a pipeline fashion. The output of each algorithm becomes the input of the next one.

Integrative hybrid metaheuristics can be also subdivided into teamwork or relay categories:

- *Integrative teamwork*. One metaheuristic (subordinate) becomes a component of another population-based metaheuristic (master) [103].
- *Integrative relay*. This kind of hybrid metaheuristics represents algorithms in which a given metaheuristic is embedded into a trajectory-based metaheuristic [103].

Collaborative Teamwork Hybrid GAs

In the case of *collaborative teamwork hybrid GAs*, we can highlight two schemes commonly used in the literature. In the first scheme, a single population is decentralized by partitioning it into several subpopulations (islands or demes), where island GAs are run performing sparse exchanges (migrations) of individuals. These kinds of models are usually referred to as distributed GAs [53] (see also Diversity Preservation based on Spatial Separation Section). Specifically, in this work several subpopulations are processed by GAs with different exploration or exploitation degrees. With this mechanism, refinement of the best solutions and expansion of the most promising zones are achieved in a parallel way. Communication between different GAs is done by sending the best solution of each population to the neighboring populations every five iterations.

In the second scheme, a set of heterogeneous metaheuristics including GAs is executed in parallel. For example, [104] combine a GA, tabu search, and a local search procedure that communicate through an adaptive memory that contains the search history. Multiple independent tabu search tasks run in parallel without any direct cooperation. However, the adaptive memory maintains a history of the search performed by these tabu search tasks. This information is used by the GA to generate individuals in unexplored regions. At the same time, elite solutions from the adaptive memory are improved with the local search procedure.

Collaborative Relay Hybrid GAs

In the *collaborative relay hybrid GAs*, a GA is executed, in a pipeline fashion, with another GA or other types of metaheuristic so that the output of each algorithm is used as input of the next one. Most instances of collaborative relay GAs follow the principle of favor exploration in the initial stages and exploitation later, inspired by the design of classical metaheuristics such as simulated annealing [75]. In this line, [17] presented an hybrid GA that combines a GA specialized for diversification and a local search process that improves the best individuals found by the GA. Whereas, [40] proposed a collaborative relay hybrid GA where the local refinement of solutions is performed by a specialized GA. They run a global real-coded GA during a determinate percentage of the available evaluations, and then they perform the local real-coded GA. The initial population for the local algorithm includes the best individuals of the global one.

Integrative Teamwork Hybrid GAs

In the case of *integrative teamwork hybrid GAs*, we can find two different approaches: *memetic algorithms* [84] and *GAs with metaheuristic-based components*. The classic scheme of memetic algorithms applies a local search method to solutions obtained by the GA that is in charge of the global search. The idea behind memetic algorithms is to provide an effective and efficient optimization method by achieving a trade-off between global exploration of GAs and local exploitation of local search procedures [83]. However, memetic algorithms also include combinations of GAs with problem-dependent heuristics, approximate algorithms, truncated exact methods, specialized recombination operators, etc. [84].

Many different instantiations of *memetic algorithms* have been presented to deal with a wide variety of application domains. However, the additional fitness function evaluations required for the local search method can increase significantly the computational cost of memetic algorithms. In order to mitigate this drawback, [83] proposed a memetic algorithm with local search chains that aims at focusing the local search action on promising areas of the search space. This is addressed by retaking the local search from solutions obtained in a previous application of the local search method, adopting also the former final strategy parameter values as its initial values.

In this line, we can find memetic algorithms in which the refinement procedure is performed by another GA instead of the classical local search procedure. With this idea, there have been presented several memetic algorithms that use GAs with a small population and short evolution or micro GAs (μGA) [64, 76, 85] (see also Diversification Techniques for GAs Section) to locally improve solutions. μGA models provide some advantages over classical local search methods. Most local search techniques find difficulties in following the right path to the optimum in complex search spaces. However, it was observed that μGAs are capable of following ridges of arbitrary direction in the search space regardless of their direction, width, or even discontinuities [64]. For example, the μGA presented in [64] is a GA with five individuals that encode perturbations. Aptitude values of these individuals depend on a solution given by the master GA. This feature ensures that search is focused on the neighborhood of the given solution, whereas low-sized population promotes high selection pressure levels.

With regard to *GAs with metaheuristic-based components*, we can find in the literature several proposals where the selection mechanism, the crossover, and the mutation operators of GAs have been replaced or extended by another classical metaheuristic. The approach proposed in [1] combines simulated annealing with a GA by extending the mutation and crossover operators with simulated annealing. Mutated and recombined solutions are accepted according to the standard simulated annealing acceptance condition. Kurahashi and Terano [71] proposed a tabu-based GA where after evaluating individuals in each iteration, they store the best individual of the generation into both long-term and short-term tabu lists. Then, the GA selection refers to the tabu lists in order to select individuals with dissimilar genotypes and consequently avoid premature convergence to local optima.

Integrative Relay Hybrid GAs

Finally, we undertake the study of *integrative relay hybrid GAs*. In this scheme, usually, a GA is used to perform one or more functions in the master metaheuristic. García-Martínez et al. [41] presented a GA designed specifically to play the role of the simulated annealing neighborhood operator. In particular, a steady-state GA creates one single candidate solution at each iteration, by crossing over the current solution of the master simulated annealing and another one from the population. Afterward, the master simulated annealing applies an acceptance mechanism to decide which solution becomes the new current solution, either the candidate solution or the current one. The other solution is inserted into the population by a replacement strategy. Another approach following this scheme was presented in [75]. In this work, the authors present an iterated local search with a perturbation operator based on a μCHC algorithm. CHC provides high diversity by means of high selective pressure, allowing diverse and promising solutions to be maintained. This behavior is desirable for a GA assuming the work of a perturbation operator.

GA Applications

GAs have had a great measure of success in search and optimization problems. The reason for a great part of their success is their ability to exploit the information accumulated about an initially unknown search space in order to bias subsequent searches into useful subspaces, i.e., their adaptation. This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools (enumerative, heuristic, etc.) are inappropriate, offering a valid approach to problems requiring efficient and effective search techniques.

With the aim of showing the broad variety of GA applications, we have searched for papers from scopus with the keywords "genetic algorithm" in the title (the query was submitted on May 2015). Table 6 outlines the number of documents belonging to different subject areas. The huge amount of papers (43,180) and the wide spectrum of subject areas with GA applications allow us to say that the research on GAs has reached a stage of great maturity, and there is an active and vibrant worldwide community of researchers working on these algorithms, as it may be confirmed in Fig. 3, which illustrates the number of publications appeared in each year in the period 1970–2014. Specifically, we may see that from approximately the last teen years, GAs involved more than 2,500 publications per year (reaching a peak of over 3,500 in 2010).

An additional remark from Table 6 concerns the outstanding activity coming from the engineering field, where challenging hard optimization problems arise, which are characterized by having multiple objectives, by being dynamic problems, by the high number of implied decision variables and the complex relationships among them, and, in many cases, by the strict feasibility constraints. To effectively face problems with such a diverse nature, the original GA scheme was extended in different ways:

- *Multiobjective GAs*. Multiobjective optimization problems require the simultaneous optimization (maximization or minimization) of several objectives that cannot be compared easily with each other. The goal of solving these problems is not to find an optimal solution but rather to find a set of nondominated solutions, the so-called Pareto set. Since the 1980s, the application of GAs in solving multiobjective optimization problems has been receiving a growing interest, and they are still one of the hottest research areas in the field of evolutionary computation [19, 24]. By evolving a population of solutions, these GA approaches are able to capture a set of nondominated solutions in a single run of the algorithm [25].

Table 6 Research papers on GAs by subject areas

Subject area	Documents
Engineering	24,095
Computer science	19,235
Mathematics	7,062
Physics and astronomy	3,058
Materials science	2,140
Energy	1,871
Decision sciences	1,863
Biochemistry, genetics, and molecular biology	1,615
Earth and planetary sciences	1,326
Chemical engineering	1,308
Environmental science	1,267
Social sciences	1,152
Chemistry	1,064
Business, management, and accounting	926
Agricultural and biological sciences	634
Medicine	589
Multidisciplinary	421
Economics, econometrics, and finance	204
Neuroscience	175
Pharmacology, toxicology, and pharmaceutics	140
Immunology and microbiology	100
Health professions	89
Arts and humanities	57
Psychology	25
Undefined	22
Nursing	7
Veterinary	5
Dentistry	2
Total	43,180

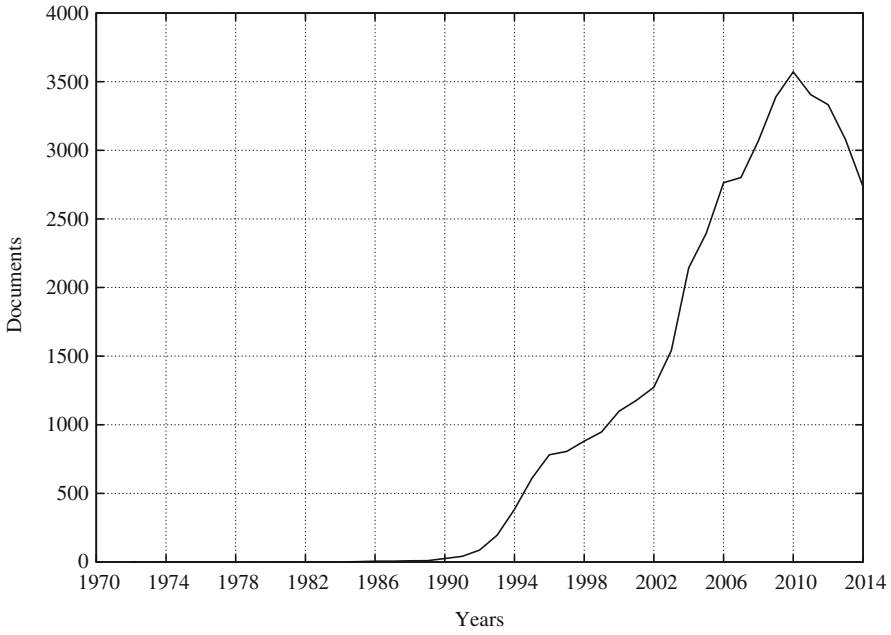


Fig. 3 GA publications per year

- *Constrained GAs*. In many science and engineering disciplines, it is common to encounter a large number of constrained optimization problems. Such problems involve an objective function that is subject to various equality and inequality constraints. The challenges in this optimization scenario arise from the various limits on the decision variables, the constraints involved, the interference among constraints, and the interrelationship between the constraints and the objective functions. Classical gradient-based optimization methods have difficulties in handling this kind of problems, as constrained optimization problems may usually lack an explicit mathematical formulation and have discrete definition domains. GAs are unconstrained search methods and lack of an explicit mechanism to bias the search in constrained search space. However, researchers have been able to tailor constraint handling techniques into GAs [18, 82, 111].
- *GAs for dynamic optimization problems*. In dynamic environments, the fitness landscape may change over time as a result of the changes of the optimization goal, problem instance, and/or some restrictions. Alternatives that were bad in the past can be good nowadays or vice versa; criteria that were important before become irrelevant now, etc., and moving in these dynamic scenarios is a challenge. For these cases, the goal of an optimization algorithm is no longer to find a satisfactory solution to a fixed problem, but to track the moving optimum in the search space as closely as possible. This poses great difficulties to standard GAs, because they cannot adapt well to the changing environment once

converged. Over the past decade, an important number of GA approaches have been developed to face problems with these features, and readers are referred to [21, 62, 116] for a comprehensive overview.

- *Multimodal GAs*. Many real-world problems require an optimization algorithm that is able to explore multiple optima in their search space. However, given a problem with multiple solutions, a simple GA shall tend to converge to a single solution. As a result, various mechanisms have been proposed to stably maintain a diverse population throughout the search [73, 74, 117], thereby allowing GAs to identify multiple optima reliably in these multimodal function landscapes. Many of these methods work by encouraging artificial niche formation through sharing [46] and crowding [52, 97, 106] (see section "Strategies to Maintain Diversity").

Conclusions

GAs have become a tool of choice since they offer practical advantages to researchers facing difficult optimization problems because they may locate high-performance regions of vast and complex search spaces. Other advantages include the simplicity of the approach, their flexibility, and their robust response to changing circumstances. In the previous sections, we firstly provided a comprehensive guide for newcomers, revising the basic ingredients for designing a GA and presenting the classical GA approaches to solve three widely known optimization problems with very different characteristics. Then, we undertook the study of more complex GA instances through two fruitful research lines in this field such as the preservation of diversity in GAs and the hybridizations of GAs with other metaheuristics. With regard to the former, we analyzed the premature convergence problem in GAs and outlined different approaches that have been proposed to introduce new methods and operators that attempt to favor the diversity in GAs. Similarly, we described different schemes in the literature to hybridize GAs with other metaheuristics and presented different instantiations of these schemes that have been successfully applied. Finally, we reviewed the current activity in the GA field in terms of number of publications and areas of interest, which shows the high level of interest in the field of GAs and predicts a promising evolution of this research area.

Cross-References

▸ A History of Metaheuristics
▸ Evolutionary Algorithms
▸ Memetic Algorithms
▸ Multi-objective Optimization
▸ Random-Key Genetic Algorithms

References

1. Adler D (1993) Genetic algorithm and simulated annealing: a marriage proposal. In: Proceedings of the IEEE international conference on neural network, pp 1104–1109
2. Alba E, Dorronsoro B (2005) The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. IEEE Trans Evol Comput 9(2):126–142
3. Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462
4. Alba E, Troya JM (2001) Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Gener Comput Syst 17(4):451–465
5. Al-Naqi A, Erdogan A, Arslan T (2013) Adaptive three-dimensional cellular genetic algorithm for balancing exploration and exploitation processes. Soft Comput 17(7): 1145–1157
6. Araujo L, Merelo J (2011) Diversity through multiculturality: assessing migrant choice policies in an island model. IEEE Trans Evol Comput 15(4):456–469
7. Bäck T, Schütz M (1996) Intelligent mutation rate control in canonical genetic algorithms. In: Raś Z, Michalewicz M (eds) Foundations of intelligent systems. Lecture notes in computer science, vol 1079, pp 158–167
8. Baker J (1987) Adaptive selection methods for genetic algorithms. In: Grefenstette J (ed) International conference on genetic algorithms applications and their application. Erlbaum Associates, pp 14–21
9. Baker J (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the international conference on genetic algorithms, pp 14–21
10. Baluja S, Caruanna R (1995) Removing the genetics from the standard genetic algorithm. In: Proceedings of the annual conference on machine learning, pp 38–46
11. Banzhaf W (1990) The "Molecular" traveling salesman. Biol Cybern 64:7–14
12. Beyer H, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. IEEE Trans Evol Comput 5(3):250–270
13. Bhandarkar S, Zhang H (1999) Image segmentation using evolutionary computation. IEEE Trans Evol Comput 3(1):1–21
14. Blum C (2010) Hybrid metaheuristics – guest editorial. Comput Oper Res 37(3):430–431
15. Blum C, Puchinger J, Raidl G, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11:4135–4151
16. Caruana R, Schaffer J (1988) Representation and hidden bias: gray versus binary coding for genetic algorithms. In: Proceedings of the fifth international conference on machine learning, pp 153–162
17. Chelouah R, Siarry P (2003) Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. Eur J Oper Res 148(2): 335–348
18. Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191(11–12):1245–1287
19. Coello C, Lamont G, Veldhuizen D (2006) Evolutionary algorithms for solving multi-objective problems. Springer, New York
20. Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput Surv 45(3):35:1–35:33
21. Cruz C, González JR, Pelta DA (2011) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput 15(7):1427–1448
22. Dantzig G (1957) Discrete variable extremum problems. Oper Res 5:266–277
23. Davis L (1985) Adaptive algorithms to epistactic domains. In: Proceedings of the international conference on artificial intelligence, pp 162–164
24. Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, Chichester/New York

25. Deb K (2008) Introduction to evolutionary multiobjective optimization. In: Branke J, Deb K, Miettinen K, Słowiski R (eds) Multiobjective optimization. Lecture notes in computer science, vol 5252. Springer, Berlin/Heidelberg, pp 59–96
26. De Jong K (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan
27. De Jong K (1993) Genetic algorithms are NOT function optimizers. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo
28. De Jong K, Sarma J (1993) Generation gaps revisited. In: Whitley LD (ed) Foundations of genetic algorithms. Morgan Kaufmann, San Mateo, pp 19–28
29. Eiben A, Smith J (2003) Introduction to evolutionary computation. Natural computing series. Springer, New York
30. Eiben A, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124–141
31. Eklund SE (2004) A massively parallel architecture for distributed genetic algorithms. Parallel Comput 30(5–6):647–676
32. Eshelman L (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. Foundations of genetic algorithms, vol 1. Morgan Kaufmann, San Mateo, CA, pp 265–283
33. Eshelman L, Schaffer J (1991) Preventing premature convergence in genetic algorithms be preventing incest. In: Proceedings of the international conference on genetic algorithms, pp 115–122
34. Eshelman L, Schaffer J (1993) Real-coded genetic algorithms and interval-schemata. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo, pp 187–202
35. Fernandes C, Rosa A (2008) Self-adjusting the intensity of assortative mating in genetic algorithms. Soft Comput 12(10):955–979
36. Fogarty TC (1989) Varying the probability of mutation in the genetic algorithm. In: Proceedings of the third international conference on genetic algorithms, pp 104–109
37. Fogel D (1998) Evolutionary computation: the fossil record. IEEE Press, New York
38. Gao J, Sun L, Gen M (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 35(9): 2892–2907
39. García-Martínez C, Lozano M, Herrera F, Molina D, Sánchez A (2008) Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 185:1088–1113
40. García-Martínez C, Lozano M, Herrera F, Molina D, Sánchez A (2008) Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 185(3):1088–1113
41. García-Martínez C, Lozano M, Rodríguez-Díaz F (2012) A simulated annealing method based on a specialised evolutionary algorithm. Appl Soft Comput 12(2):573–588
42. Ghannadian F, Alford C, Shonkwiler R (1996) Application of random restart to genetic algorithms. Inf Sci 95(1–2):81–102
43. Goldberg D (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, New York
44. Goldberg D (1989) Sizing populations for serial and parallel genetic algorithms. In: Schaffer J (ed) International conference on genetic algorithms. Morgan Kaufmann, pp 70–79
45. Goldberg D, Lingle R (1985) Alleles, Loci and the traveling salesman problem. In: Proceedings of the international conference on genetic algorithms, pp 154–159
46. Goldberg D, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Grefenstette J (ed) Proceedings of the international conference on genetic algorithms. L. Erlbraum Associates, pp 41–49
47. Goldberg D, Korb B, Deb K (1990) Messy genetic algorithms: motivation, analysis, and first results. Complex Syst 3:493–530

48. Gonçalves JF, Resende MG (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17(5):487–525
49. Grosan C, Abraham A (2007) Hybrid evolutionary algorithms: methodologies, architectures, and reviews. In: Grosan C, Abraham A, Ishibuchi H (eds) Hybrid evolutionary algorithms. Springer, Berlin/New York, pp 1–17
50. Grötschel M, Padberg MM (1978) On the symmetric traveling salesman problem: theory and computations. In: Optimization and operations research. Lecture notes in econocmics and mathematical systems, vol 157. Springer, pp 105–115
51. Gupta S, Garg ML (2013) Binary trie coding scheme: an intelligent genetic algorithm avoiding premature convergence. Int J Comput Math 90(5):881–902
52. Harik G (1995) Finding multimodal solutions using restricted tournament selection. In: Proceedings of the international conference on genetic algorithms. Morgan Kaufmann, pp 24–31
53. Herrera F, Lozano M (2000) Gradual distributed real-coded genetic algorithms. IEEE Trans Evol Comput 4(1):43–63
54. Herrera F, Lozano M (2003) Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. Soft Comput 7(8):545–562
55. Herrera F, Lozano M, Verdegay J (1998) Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. Artif Intell Rev 12:265–319
56. Herrera F, Lozano M, Sánchez A (2003) A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int J Intell Syst 18(3):309–338
57. Hinterding R, Michalewicz Z, Eiben A (1997) Adaptation in evolutionary computation: a survey. In: IEEE international conference on evolutionary computation, pp 65–69
58. Holland J (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
59. Hutter M, Legg S (2006) Fitness uniform optimization. IEEE Trans Evol Comput 10(5): 568–589
60. Iman R, Conover W (1982) A distribution-free approach to inducing rank correlation among input variables. Commun Stat Simul Comput 11(3):311–334
61. Janikow C, Michalewicz Z (1991) An experimental comparison of binary and floating point representation in genetic algorithms. In: Proceedings of the fourth international conference on genetic algorithms, pp 31–36
62. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Trans Evol Comput 9(3):303–317
63. Karafotias G, Hoogendoorn M, Eiben A (2015) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19(2):167–187
64. Kazarlis S, Papadakis S, Theocharis J, Petridis V (2001) Microgenetic algorithms as generalized hill-climbing operators for GA optimization. IEEE Trans Evol Comput 5(3):204–217
65. Kazimipour B, Li X, Qin A (2014) A review of population initialization techniques for evolutionary algorithms. In: Proceedings of the IEEE congress on evolutionary computation, pp 2585–2592
66. Kita H (2001) A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms. Evol Comput 9(2):223–241
67. Kominami M, Hamagami T (2007) A new genetic algorithm with diploid chromosomes by using probability decoding for non-stationary function optimization. In: IEEE international conference on systems, man and cybernetics, 2007. ISIC. pp 1268–1273
68. Koumousis V, Katsaras C (2006) A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. IEEE Trans Evol Comput 10(1):19–28
69. Krishnakumar K (1989) Micro-genetic algorithms for stationary and non-stationary function optimization. In: Intelligent control and adaptive systems. Proceedings of the SPIE, vol 1196, pp 289–296

70. Kuo T, Hwang S (1996) A genetic algorithm with disruptive selection. IEEE Trans Syst Man Cybern 26(2):299–307
71. Kurahashi S, Terano T (2000) A genetic algorithm with tabu search for multimodal and multiobjective function optimization. In: Whitley LD, Goldberg DE, Cant-Paz E, Spector L, Parmee IC, Beyer HG (eds) GECCO. Morgan Kaufmann, pp 291–298
72. Larrañaga P, Kuijpers C, Murga R, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artif Intell Rev 13(2):129–170
73. Li JP, Balazs ME, Parks GT, Clarkson PJ (2002) A species conserving genetic algorithm for multimodal function optimization. Evol Comput 10(3):207–234
74. Liang Y, Leung KS (2011) Genetic algorithm with adaptive elitist-population strategies for multimodal function optimization. Appl Soft Comput 11(2):2017–2034
75. Lozano M, García-Martínez C (2010) Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: overview and progress report. Comput Oper Res 37:481–497
76. Lozano M, Herrera F, Krasnogor N, Molina D (2004) Real-coded memetic algorithms with crossover hill-climbing. Evol Comput 12(3):273–302
77. Lozano M, Herrera F, Cano JR (2008) Replacement strategies to preserve useful diversity in steady-state genetic algorithms. Inf Sci 178(23):4421–4433
78. Mahfoud S (1992) Crowding and preselection revised. In: Männer R, Manderick B (eds) Parallel problem solving from nature, vol 2. Elsevier Science, pp 27–36
79. Mallipeddi R, Suganthan P (2010) Ensemble of constraint handling techniques. IEEE Trans Evol Comput 14(4):561–579
80. Mauldin M (1984) Maintaining diversity in genetic search. In: National conference on artificial intelligence, Austin, pp 247–250
81. Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer, Berlin/New York
82. Michalewicz A, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32
83. Molina D, Lozano M, García-Martínez C, Herrera F (2010) Memetic algorithms for continuous optimization based on local search chains. Evol Comput 18(1):27–63
84. Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. Kluwer Academic, Boston, pp 105–144
85. Noman N, Iba H (2008) Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 12(1):107–125
86. Nomura T, Shimohara K (2001) An analysis of two-parent recombinations for real-valued chromosomes in an infinite population. Evol Comput 9(3):283–308
87. Oh IS, Lee JS, Moon BR (2004) Hybrid genetic algorithms for feature selection. IEEE Trans Pattern Anal Mach Intell 26(11):1424–1437
88. Oliver I, Smith D, Holland J (1987) A study of permutation crossover operators on the TSP. In: Proceedings of the international conference on genetic algorithms and their applications, pp 224–230
89. Pereira A, de Andrade BB (2015) On the genetic algorithm with adaptive mutation rate and selected statistical applications. Comput Stat 30(1):131–150
90. Potts J, Giddens T, Yadav S (1994) The development and evaluation of an improved genetic algorithm based on migration and artificial selection. IEEE Trans Syst Man Cybern 24: 73–86
91. Preechakul C, Kheawhom S (2009) Modified genetic algorithm with sampling techniques for chemical engineering optimization. J Ind Eng Chem 15:110–118
92. Preux P, Talbi E (1999) Towards hybrid evolutionary algorithms. Int Trans Oper Res 6(6): 557–570

93. Raidl G (2006) A unified view on hybrid metaheuristics. In: Almeida F, Aguilera MB, Blum C, Vega JM, Pérez MP, Roli A, Sampels M (eds) Hybrid metaheuristics, LNCS, vol 4030. Springer, pp 1–12
94. Reeves C (2010) Genetic algorithms. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, vol 146. Springer, New York, pp 109–139
95. Reeves C, Rowe J (2001) Genetic algorithms: principles and perspectives. Kluwer, Norwell
96. Rodriguez F, Garcia-Martinez C, Lozano M (2012) Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: taxonomy, comparison, and synergy test. IEEE Trans Evol Comput 16(6):787–800
97. Sareni B, Krahenbuhl L (1998) Fitness sharing and Niching methods revisited. IEEE Trans Evol Comput 2(3):97–106
98. Serpell M, Smith J (2010) Self-adaption of mutation operator and probability for permutation representations in genetic algorithms. Evol Comput 18(3):491–514
99. Smith JE, Fogarty TC (1997) Operator and parameter adaptation in genetic algorithms. Soft Comput 1(2):81–87
100. Smith A, Coit D, Baeck T, Fogel D, Michalewicz Z (1997) Penalty functions. In: Bäck T, Fogel DB, Michalewics Z (eds) Handbook on evolutionary computation. Oxford University Press, New York, pp C5.2:1–C5.2:6
101. Srinivas M, Patnaik L (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
102. Syswerda G (1989) Uniform crossover in genetic algorithms. In: Proceedings of the international conference on genetic algorithms, pp 2–9
103. Talbi E (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564
104. Talbi EG, Bachelet V (2006) Cosearch: a parallel cooperative metaheuristic. J Math Model Algorithms 5(1):5–22
105. Tantar A, Melab N, Talbi E (2008) A grid-based genetic algorithm combined with an adaptive simulated annealing for protein structure prediction. Soft Comput 12(12):1185–1198
106. Thierens D (1998) Selection schemes, elitist recombination, and selection intensity. In: Proceedings of the 7th international conference on genetic algorithms. Morgan Kaufmann, pp 152–159
107. Ting CK, Li ST, Lee C (2003) On the harmonious mating strategy through tabu search. Inf Sci 156:189–214
108. Tuson A, Ross P (1998) Adapting operator settings in genetic algorithms. Evol Comput 6(2):161–184
109. Uyar Ai, Harmanci AE (2005) A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. Soft Comput 9(11): 803–814
110. van Kemenade C, Kok J, Eiben AE (1995) Raising GA performance by simultaneous tuning of selective pressure and recombination disruptiveness. In: Proceedings of the 1995 IEEE congress on evolutionary computation (CEC 1995), pp 346–351
111. Venkatraman S, Yen G (2005) A generic framework for constrained optimization using genetic algorithms. IEEE Trans Evol Comput 9(4):424–435
112. Vrugt J, Robinson B, Hyman J (2009) Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259
113. Whitley D (1989) The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Proceedings of the international conference on genetic algorithms. Morgan Kaufmann, pp 116–121
114. Wong YY, Lee KH, Leung KS, Ho CW (2003) A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. Soft Comput 7(8):506–515
115. Yang CH, Nygard K (1993) Effects of initial population in genetic search for time constrained traveling salesman problems. In: Proceedings of the ACM computer science conference, pp 378–383

116. Yang S, Ong Y, Jin Y (eds) (2007) Evolutionary computation in dynamic and uncertain environments. Studies in computational intelligence, vol 51. Springer, Berlin/London
117. Yao J, Kharma N, Grogono P (2010) Bi-objective multipopulation genetic algorithm for multimodal function optimization. IEEE Trans Evol Comput 14(1):80–102
118. Yeniay Ö (2005) Penalty function methods for constrained optimization with genetic algorithms. Math Comput Appl 10(1):45–56
119. Yuen SY, Chow CK (2009) A genetic algorithm that adaptively mutates and never revisits. IEEE Trans Evol Comput 13(2):454–472

GRASP

16

Paola Festa and Mauricio G. C. Resende

Contents

Introduction 466
Basic Components 467
Construction Phase 468
Local Search Phase 471
Enhancements 472
Reactive GRASP 473
Cost Perturbations 474
Bias Functions 475
POP in Construction 476
Hybridizations 476
Automatic Tuning 481
Conclusions 482
Cross-References 482
References 482

Abstract

GRASP (greedy randomized adaptive search procedure) is a multistart metaheuristic for computing good-quality solutions of combinatorial optimization problems. Each GRASP iteration is usually made up of a construction phase, where a feasible solution is constructed, and a local search phase which starts at the constructed solution and applies iterative improvement until a locally

P. Festa (✉)
Department of Mathematics and Applications "Renato Caccioppoli", University of Napoli FEDERICO II, Napoli, Italy
e-mail: paola.festa@unina.it

M. G. C. Resende
Amazon.com, Inc. and University of Washington, Seattle, WA, USA
e-mail: resendem@amazon.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_23

optimal solution is found. Typically, the construction phase of GRASP is a randomized greedy algorithm, but other types of construction procedures have been also proposed. Repeated applications of a construction procedure yields diverse starting solutions for the local search. This chapter gives an overview of GRASP describing its basic components and enhancements to the basic procedure, including reactive GRASP and intensification strategies.

Keywords
GRASP · Combinatorial optimization · Metaheuristics · Local search · Path-relinking · Hybrid metaheuristics

Introduction

Given a finite, or countably infinite, solution set X and a real-valued objective function $f : X \rightarrow R$, in any combinatorial optimization problem, one seeks a solution $x^* \in X$ with $f(x^*) \leq f(x), \forall\ x \in X$. Several of these problems can be solved in polynomial time, but many of them are computationally intractable, since exact polynomial-time algorithms to solve them are unknown [59]. Furthermore, most real-world problems found in industry and government are either computationally intractable by their nature or sufficiently large so as to preclude the use of exact algorithms. In such cases, heuristic methods are usually employed to find good, but not necessarily guaranteed, optimal solutions. The effectiveness of these methods depends upon their ability to adapt to avoid entrapment at local optima and exploit the basic structure of the problem. Building on these notions, various heuristic search techniques have been developed that have demonstrably improved our ability to obtain good solutions to difficult combinatorial optimization problems. The most promising of such techniques include simulated annealing [78], tabu search [61, 62, 65], genetic algorithms [69], biased random key genetic algorithms [70], scatter search and path-relinking [66], variable neighborhood search [72], and GRASP (greedy randomized adaptive search procedure) [42, 43].

GRASP (greedy randomized adaptive search procedure) is a multistart metaheuristic for producing good-quality solutions of combinatorial optimization problems. Unlike ant colony [38] and evolutionary algorithms [19], GRASP is not nature inspired, i.e., it is not inspired by the principles of natural evolution in the sense of nature's capability to evolve living beings to keep them well adapted to their environment. Instead, GRASP proceeds in iterations, and each GRASP iteration is usually made up of a construction phase, where a solution (feasible or even unfeasible) is constructed, and a local search phase that starts at the constructed solution and applies iterative improvement until a locally optimal solution is found. While, in general, the construction phase of GRASP is a randomized greedy algorithm, other types of construction procedures have been proposed. Repeated applications of a construction procedure yield diverse starting solutions for the local search, and the best local optimal solution found over all GRASP iterations is returned as final solution.

Table 1 Applications of GRASP: operations research problems

Routing	[15, 18, 23, 30, 81]
Logic	[37, 56, 100, 108, 113]
Covering and partition	[13, 14, 42, 60, 71, 107]
Location	[1, 34–36, 79, 126]
Minimum Steiner tree	[29, 90, 91, 118]
Optimization in graphs	[2, 17, 45, 85, 99, 109]
Assignment	[5, 41, 58, 86, 93, 101, 102, 112, 122]
Timetabling and scheduling	[4, 7, 10, 11, 26, 40, 44, 46, 82, 88, 117, 119–121]

Table 2 Applications of GRASP: some industrial applications

Manufacturing	[6, 21, 22, 27, 80, 96]
Transportation	[15, 20, 41, 123]
Telecommunications	[8, 9, 33, 79, 87, 103, 106, 124]
Graph and map drawing	[54, 55, 84, 89, 109]
Power systems	[24, 25, 39]
Computational biology	[49, 67, 75]
VLSI	[12, 13]

This chapter overviews GRASP by describing its basic components along with some among the most fruitful proposed enhancements, including reactive GRASP, intensification strategies, and hybridization with other metaheuristics. The chapter is organized as follows. Basic components, alternative construction mechanisms, and local search characteristics are described in the next section. Enhancements to the basic procedure, including reactive GRASP and intensification strategies, are discussed in the section "Enhancements". Section "Hybridizations" describes several state-of-the-art hybridizations of GRASP with other metaheuristics, while in section "Automatic Tuning" a few techniques are described to automatically tune the typical GRASP parameters. Tables 1 and 2 report a number of GRASP implementations that have appeared in the literature, covering a wide range of applications in several and heterogenous fields. The reader can refer to [50, 52, 53], which contain annotated bibliographies of the GRASP literature from 1989 to 2008.

Basic Components

Given a finite solution set X and a real-valued objective function $f : X \to R$ to be minimized, a basic GRASP metaheuristic [42, 43] is a multistart or iterative method, in which each iteration consists of two phases: construction of a solution and local search.

The construction phase builds a solution x, usually feasible, but optionally also infeasible. If x is not feasible, a repair procedure may be invoked to obtain feasibility. Once a solution x is obtained, its neighborhood is investigated by the local search until a local minimum is found. The best overall solution is kept as the

```
algorithm GRASP(f(·), g(·), MaxIterations, Seed)
1    x_best:=∅; f(x_best):=+∞;
2    for k = 1, 2, ...,MaxIterations→
3       x:=ConstructGreedyRandomizedSolution(Seed, g(·));
4       if (x not feasible) then
5          x:=repair(x);
6       endif
7       x:=LocalSearch(x, f(·));
8       if (f(x) < f(x_best)) then
9          x_best:=x;
10      endif
11   endfor;
12   return(x_best);
end GRASP
```

Fig. 1 Pseudo-code of a basic GRASP for a minimization problem

result. An extensive survey of the literature is given in [50]. The pseudo-code in Fig. 1 illustrates the main blocks of a GRASP procedure for minimization, in which `MaxIterations` iterations are performed and `Seed` is used as the initial seed for the pseudorandom number generator.

Construction Phase

Starting from an empty solution, a complete solution is iteratively constructed in the construction phase, one element at a time (see Fig. 2). The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by [74]. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e., those that can be added to the solution) in a candidate list C with respect to a greedy function $g : C \rightarrow R$. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection

```
procedure ConstructGreedyRandomizedSolution(Seed, g(·))
1    x:=∅;
2    Sort the candidate elements i according to their incremental costs g(i);
3    while (x is not a complete solution)→
4        RCL:=MakeRCL();
5        v:=SelectIndex(RCL, Seed);
6        x := x ∪ {v};
7        Resort remaining candidate elements j according to g(j);
8    endwhile;
9    return(x);
end ConstructGreedyRandomizedSolution;
```

Fig. 2 Basic GRASP construction phase pseudo-code

of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL). This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

The most used technique to build the RCL applies the $\min\max -\alpha$ percentage rules, which will be explained in the following.

At any GRASP iteration, let $g_{\min}$ and $g_{\max}$ be the smallest and the largest incremental costs, respectively, i.e.,

$$g_{\min} = \min_{i \in C} g(i), \qquad g_{\max} = \max_{i \in C} g(i). \tag{1}$$

The RCL is made up of elements $i \in C$ with the best (i.e., the smallest) incremental costs $g(i)$. There are two main mechanisms to build this list: a *cardinality-based* (CB) and a *value-based* (VB) mechanism. In the CB case, the RCL is made up of the k elements with the best incremental costs, where k is a parameter. In the VB case, the RCL is associated with a parameter $\alpha \in [0, 1]$ and a threshold value $\mu = g_{\min} + \alpha(g_{\max} - g_{\min})$. All candidate elements i whose incremental cost $g(i)$ is no greater than the threshold value are inserted into the RCL, i.e., $g(i) \in [g_{\min}, \mu]$. Note that the case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction. The pseudo-code in

procedure `ConstructGreedyRandomizedSolution`(`Seed`, α, k, $g(\cdot)$)

1 x:=$\emptyset$;
2 Initialize the candidate set C by all elements;
3 Evaluate the incremental cost $g(i)$ for all $i \in C$;
4 **while** ($|C| > 0$)→
5 $g_{min} := \min_{i \in C} g(i)$; $g_{max} := \max_{i \in C} g(i)$;
6 **if** (CB RCL is used) **then**
7 Sort candidate elements $i \in C$ according to $g(i)$;
8 RCL:=$C[1 \cdots k]$;
9 **else** RCL:=$\{i \in C \mid g(i) \leq g_{min} + \alpha(g_{max} - g_{min})\}$;
10 **endif**;
11 v:=`SelectIndex`(RCL, `Seed`);
12 $x := x \cup \{v\}$;
13 Update the candidate set C;
14 Reevaluate the incremental costs $g(i)$ for all $i \in C$;
15 **endwhile**;
16 **return**(x);

end `ConstructGreedyRandomizedSolution`;

Fig. 3 Refined pseudo-code of the GRASP construction phase

Fig. 3 is a refinement of the greedy randomized construction pseudo-code shown in Fig. 2.

Prais and Ribeiro in [104, 105] observed the behavior of GRASP and the quality of the GRASP output solutions for different RCL construction mechanisms, based on different strategies for the variation of the value of the parameter α:

(a) α is randomly chosen from a uniform discrete probability distribution;
(b) α is randomly chosen from a decreasing nonuniform discrete probability distribution;
(c) Fixed value of α, close to the purely greedy choice.

The authors incorporated these three strategies into the GRASP procedures developed for four optimization problems: (1) matrix decomposition for traffic assignment in communication satellite [106], (2) set covering [42], (3) weighted MAX-SAT [113,114], and (4) graph planarization [109,115]. The resulting heuristics were tested on a subset of state-of-the-art instances for each type of problem. The total number of iterations performed was fixed at 10,000. The observed conclusions can be summarized as follows. Strategy (c) presented the shortest average computation times for three out the four problem types. It was also the one with the least variability in the constructed solutions and, consequently, found the best solution the fewest times. Strategy (a) presented a high number of hits, and this behavior also illustrates the effectiveness of strategies based on the variation of the RCL parameter.

In [104, 106], Prais and Ribeiro also tested GRASP with a further RCL construction mechanism, in which the parameter α is self-adjusted and its value is periodically modified according to the quality of the obtained solutions. This extension of the basic GRASP is called *reactive GRASP* and will be described in detail in the next section devoted to the description of the enhancements to the basic GRASP.

Local Search Phase

As is the case for many deterministic methods, the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The *neighborhood structure* N for a problem relates a solution s of the problem to a subset of solutions $N(s)$. A solution s is said to be *locally optimal* if there is no better solution in $N(s)$ with respect to the objective function value. The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution. Figure 4 illustrates the pseudo-code of a generic local search procedure for a minimization problem.

In a GRASP framework, a local search starts from an initial solution $x_0 \in X$ and iteratively generates a sequence of improving solutions $x_1, \ldots, x_M$, where $M = \texttt{MaxIterations}$. At the k-th iteration, $k = 1, \ldots, M$, x_k is locally optimal respect to the neighborhood $N(x_{k-1})$ since $N(x_{k-1})$ is searched for an improving solution x_k such that $f(x_k) < f(x_{k-1})$. If such a solution is found, it is made the current solution. Otherwise, the search ends with x_{k-1} as a local optimum.

The effectiveness of local search depends on several factors, such as the neighborhood structure, the function to be minimized, and the starting solution. It has been experimentally shown that randomly generated solutions are of poor quality on average. On the other hand, greedy algorithms usually produce solutions of better quality than those of randomly generated solutions. Therefore, using greedy solutions as starting points for local search in a multistart procedure will

```
procedure LocalSearch(x, f(·))
1    Let N(x) be the neighborhood of x;
2    H:={y ∈ N(x) | f(y) < f(x)};
3    while (|H| > 0)→
4        x:=Select(H);
5        H:={y ∈ N(x) | f(y) < f(x)};
6    endwhile
7    return(x);
end LocalSearch
```

Fig. 4 Pseudo-code of a generic local search procedure

usually lead to good, though, most often, suboptimal solutions. This is because the amount of variability in greedy solutions is small and it is less likely that a greedy starting solution will be in the basin of attraction of a global optimum than a random solution. A greedy randomized construction as the one embedded in GRASP adds variability to the greedy algorithm.

In [110], besides analyzing the quality of the solution obtained by varying between randomness and greediness in the VB mechanisms of the GRASP construction procedure, Resende and Ribeiro also analyzed the quality of the solution output of the local search starting from solutions obtained by applying VB mechanisms with different values for the α parameter. As result of this analysis, the variance of the overall solution diversity, final solution quality, and running time increased with the variance of the solution values obtained in the construction phase. Moreover, it emerged that it is unlikely that GRASP finds an optimal solution if the average solution value is low, even if there is a large variance in the overall solution values, such as is the case for $\alpha = 0$. On the other hand, if there is little variance in the overall solution values, it is also unlikely that GRASP finds an optimal solution, even if the average solution is high, as is the case for $\alpha = 1$. Good solutions are usually obtained in the presence of relatively high average solution values and of a relatively large variance, such as is the case for $\alpha = 0.8$.

Enhancements

To improve the performance of the basic GRASP framework, most efforts have focused on construction mechanisms. Since Mockus et al. [95] pointed out that GRASP with a fixed nonzero RCL parameter α is not asymptotically convergent

to a global optimum (During construction, a fixed RCL parameter may rule out a candidate that is present in all optimal solutions.), several remedies have been proposed to get around this problem. They include reactive GRASP, cost perturbations in place of randomized selection, bias functions, memory and learning, and local search on partially constructed solutions.

Reactive GRASP

The results of the study conducted in [104, 106] involving variation of the value of the RCL parameter α motivated the proposal of the extension of the basic GRASP called reactive GRASP. Prais and Ribeiro in [106] have shown that using a single fixed value for the value of RCL parameter α very often hinders finding a high-quality solution, which eventually could be found if another value was used. Moreover, one drawback of the basic GRASP is the lack of *learning* from the history of solutions found in previous iterations. The basic algorithm discards information about any solution encountered that does not improve the incumbent. Instead, it is worth to use information gathered from good solutions leading to *memory-based* procedures. Information about the quality of previously generated solutions can influence the construction phase, by modifying the selection probabilities associated with each element of the RCL.

In this paragraph, we describe reactive GRASP, the first enhancement that incorporates a learning mechanism in the memoryless construction phase of the basic GRASP. In reactive GRASP, the value of the RCL parameter α is selected in each iteration from a discrete set of possible values with a probability that depends on the solution values found along the previous iterations. One way to accomplish this is to use the rule proposed in [106]. Let $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ be the set of possible values for α. At the first GRASP iteration, all m values have the same probability to be selected, i.e.,

$$p_i = \frac{1}{m}, \qquad i = 1, 2, \ldots, m. \tag{2}$$

At any subsequent iteration, let $\hat{z}$ be the incumbent solution, and let A_i be the average value of all solutions found using $\alpha = \alpha_i$, $i = 1, \ldots, m$. The selection probabilities are periodically reevaluated as follows:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}, \tag{3}$$

where $q_i = \hat{z}/A_i$, $i = 1, \ldots, m$. If values of $\alpha = \alpha_i$ ($i \in \{1, \ldots, m\}$) lead to the best solutions on average, then the value of q_i is increased, and larger values of q_i correspond to more suitable values for α. The probabilities associated with these more appropriate values will then increase when they are reevaluated.

Due to greater diversification and less reliance on parameter tuning, reactive GRASP has led to improvements over the basic GRASP in terms of robustness and solution quality. In fact, it has been successfully applied in power system transmission network planning [24] and in a capacitated location problem [36].

Cost Perturbations

Another step toward an improved and alternative solution construction mechanism is to allow *cost perturbations*. The idea is to introduce some "noise" in the original costs in a fashion that resembles the noising method of Charon and Hudry [31,32]. Cost perturbations are effective in all cases when the construction algorithm is not very sensitive to randomization, as, for example, in the case of the Steiner problem in graphs. To solve this problem, the hybrid GRASP procedure proposed by Ribeiro et al. in [118] was used as one of the main building blocks of the construction phase, the shortest path heuristic of Takahashi and Matsuyama [125].

Another situation where cost perturbations can be effective is when there is no greedy algorithm available for the problem to be solved, as, for example, in the case of the prize-collecting Steiner tree problem. To solve this problem, the hybrid GRASP procedure proposed by Canuto et al. in [29] used the primal-dual algorithm of Goemans and Williamson [68] to build initial solutions using perturbed costs. More in details, in [29], at each iteration a new solution for the prize-collecting Steiner tree problem is built using node prizes updated by a perturbation function, according to the structure of the current solution. Two different prize perturbation schemes are used to enforce search diversification, as described in the following.

Perturbation by eliminations: The primal-dual algorithm used in the construction phase is driven to build a new solution without some of the nodes appearing in the solution constructed in the previous iteration. This is done by changing to zero the prizes of some persistent nodes, which appeared in the last solution built and remained at the end of the local search. A parameter μ controls the fraction of the persistent nodes whose prizes are temporarily set to zero;

Perturbation by prize changes: Similarly to the noising method of Charon and Hudry [31, 32], some noise is introduced into the node prizes, resulting in a change of the objective function as well. For each node i, a perturbation factor $\beta(i)$ is randomly generated in the interval $[1 - a, 1 + a]$, where a is an implementation parameter. The original prize $\pi(i)$ associated with node i is temporarily changed to $\pi(i) = \pi(i) \cdot \beta(i)$.

Experimental results have shown that embedding a strategy of costs perturbation into a GRASP framework improves the best overall results. The hybrid GRASP with path-relinking proposed for the Steiner problem in graphs by Ribeiro et al. in [118] uses this cost perturbation strategy and is among the most effective heuristics currently available. Path-relinking will be described in detail in the subsequent section devoted to the description of hybrid GRASP with other heuristic frameworks.

Bias Functions

Another construction mechanism was proposed by Bresina [28]. Once the RCL is built, instead of choosing with equal probability one candidate among the RCL elements, Bresina introduced a family of probability distributions to bias the selection toward some particular candidates. A bias function is based on a rank $r(x)$ assigned to each candidate x according to its greedy function value and is evaluated only for the elements in RCL. Several different bias functions were introduced:

i. Random bias: $\text{bias}(r(x)) = 1$;
ii. Linear bias: $\text{bias}(r(x)) = 1/r(x)$;
iii. Log bias: $\text{bias}(r(x)) = \log^{-1}[r(x) + 1]$;
iv. Exponential bias: $\text{bias}(r(x)) = e^{-r}$;
v. Polynomial bias of order n: $\text{bias}(r(x)) = r^{-n}$.

Let $\text{bias}(r(x))$ be one of the bias functions defined above. Once these values have been evaluated for all elements of the RCL, the probability p_x of selecting element x is

$$p_x = \frac{\text{bias}(r(x))}{\sum_{y \in RCL} \text{bias}(r(y))}. \tag{4}$$

A successful application of Bresina's bias function can be found in [26], where experimental results show that, although valid on all candidates, the evaluation of bias functions may be restricted only to the elements of the RCL.

Reactive GRASP has been the first and very simple attempt to enhance the basic GRASP in order to save and use history from previous iterations. Another very simple attempt is due to Fleurent and Glover [58] who proposed improved constructive multistart strategies that besides defining a special bias function also maintains a pool of *elite solutions* to be used in the construction phase. To become an elite solution, a solution must be either better than the best member of the pool or better than its worst member and sufficiently different from the other solutions in the pool, in order to preserve not only solution quality but also the diversity of solutions. Fleurent and Glover defined: (1) a *strongly determined variable* as one that cannot be changed without eroding the objective or changing significantly other variables, (2) a *consistent variable* as one that receives a particular value in a large portion of the elite solution set, and (3) for each solution component i, a measure $I(i)$ of its strongly determined and consistent features that becomes larger as i appears more often in the pool of elite solutions. The intensity function $I(i)$ is used in the construction phase as follows. Recall that $g(i)$ is the greedy function, i.e., the incremental cost associated with the insertion of element i into the solution under construction. Let $K(i) = F(g(i), I(i))$ be a function of the greedy and the intensification functions. The idea of Fleurent and Glover is to define a special bias function that depends on $K(\cdot)$. In fact, the intensification scheme biases selection

from the RCL to those elements i with a high value of $K(i)$ by setting its selection probability to be

$$p_i = \frac{K(i)}{\sum_{y \in RCL} K(y)}. \tag{5}$$

They suggested $K(i) = \lambda g(i) + I(i)$, with $K(i)$ varying with time by changing the value of λ, e.g., initially λ may be set to a large value that is decreased when diversification is called for. Rules and procedures for changing the value of λ are given by Fleurent and Glover [58] and Binato et al. [26].

POP in Construction

The intuitive idea behind the *proximate optimality principle* (POP) is that "good solutions at one level (stage of the algorithm) are likely to be found 'close' to good solutions at an adjacent level." Given the combinatorial character of the problem to be solved, Fleurent and Glover [58] proposed a GRASP for the quadratic assignment problem that applies local search not only at the end of each construction phase but also during the construction itself on a subset of components of the solution under construction. This further application of local search aims to "iron out" from the current solution its "bad" components. Nevertheless, experimental investigation conducted in the literature has shown that applying the POP idea at each construction iteration is excessively running and time consuming. One possibility to implement the idea in a more efficient way is to apply local search during a few points in the construction phase and not during each construction iteration. In Binato et al. [26], local search is applied after 40 % and 80 % of the construction moves have been taken, as well as at the end of the construction phase.

Hybridizations

As enhancements to its basic framework, different hybridizations of GRASP with several other metaheuristics have been studied and proposed in the literature. In this section, some of them are surveyed and briefly described.

Laguna and Gonzalez-Velarde in 1991 [82] have first studied hybridization of GRASP with tabu search. Later, in 1999, Delmaire et al. [36] proposed a reactive GRASP whose local search have been strengthened by tabu search. In particular, they have proposed two approaches. In the first, GRASP is applied as a powerful diversification strategy in the context of a tabu search procedure. The second approach is an implementation of the reactive GRASP algorithm, in which the local search phase is strengthened by tabu search. Results reported for the capacitated location problem show that the hybrid approaches perform better than the pure methods previously used.

GRASP has been used also in conjunction with genetic algorithms. Basically, the feasible solution found by using a GRASP construction phase has been used as initial population by a genetic algorithm, as, for example, in [16] and in [3], where a greedy genetic algorithm is proposed for the quadratic assignment problem.

Another interesting hybridization of GRASP involves VNS (variable neighborhood search) and variable neighborhood descent (VND) proposed by Hansen and Mladenović [73, 94]. Almost all randomization effort in the basic GRASP algorithm involves the construction phase, while local search stops at the first local optimum. On the other hand, strategies such as VNS and VND rely almost entirely on the randomization of the local search to escape from local optima. With respect to this issue, probabilistic strategies such as GRASP and VNS may be considered as complementary and potentially capable of leading to effective hybrid methods.

VNS is based on the exploration of a dynamic neighborhood model. Contrary to other metaheuristics based on local search methods, VNS allows changes of the neighborhood structure along the search. It explores increasingly distant neighborhoods of the current best found solution x. Each step has three major phases: neighbor generation, local search, and jump. Let N_k, $k = 1, \ldots, k_{\max}$ be a set of predefined neighborhood structures, and let $N_k(x)$ be the set of solutions in the kth-order neighborhood of a solution x. In the first phase, a neighbor $x' \in N_k(x)$ of the current solution is applied. Next, a solution x'' is obtained by applying local search to x'. Finally, the current solution jumps from x to x'' in case the latter improved the former. Otherwise, the order of the neighborhood is increased by one, and the above steps are repeated until some stopping condition is satisfied.

A first attempt in the direction of integrating VNS into GRASP was done by Martins et al. [92]. The construction phase of their hybrid heuristic for the Steiner problem in graphs follows the greedy randomized strategy of GRASP, while the local search phase makes use of two different neighborhood structures as a VND strategy. Their heuristic was later improved by Ribeiro, Uchoa, and Werneck [118], one of the key components of the new algorithm being another strategy for the exploration of different neighborhoods. Ribeiro and Souza [116] also combined GRASP with VND in a hybrid heuristic for the degree-constrained minimum spanning tree problem. Festa et al. [55] studied different variants and combinations of GRASP and VNS for the MAX-CUT problem, finding and improving some of the solutions that at the time were the best known solutions for some open instances from the literature. In [47], the authors studied several hybridizations of GRASP, including VNS, for the far from most string problem.

At last, we devote the remainder of this section to the combination of the basic GRASP with path-relinking. Path-relinking was originally proposed by Glover [63] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [64–66]. It can be traced back to the pioneering work of Kernighan and Lin [77]. Starting from one or more elite solutions, paths in the solution space leading toward other guiding elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. At each

iteration, all moves that incorporate attributes of the guiding solution are analyzed, and the move that best improves (or least deteriorates) the initial solution is chosen.

The first proposal of a hybrid GRASP with path-relinking was in 1999 due to Laguna and Martí [83]. It was followed by several extensions, improvements, and successful applications [5,29,47,48,51,55,56]. Path-relinking is applied to a pair of solutions $\mathbf{x}$ and $\mathbf{y}$, where one can be the solution obtained from the current GRASP iteration and the other is a solution from an elite set of solutions. $\mathbf{x}$ is called the *initial solution* and $\mathbf{y}$ the *guiding solution*. The set $\mathcal{E}$ of elite solutions has usually a fixed size that does not exceed `MaxElite`. Given the pair $\mathbf{x}, \mathbf{y}$, their common elements are kept constant, and the space of solutions spanned by these elements is searched with the objective of finding a better solution. The size of the solution space grows exponentially with the distance between the *initial* and *guiding* solutions, and therefore only a small part of the space is explored by path-relinking. The procedure starts by computing the symmetric difference $\Delta(\mathbf{x}, \mathbf{y})$ between the two solutions, i.e., the set of moves needed to reach $\mathbf{y}$ (target solution) from $\mathbf{x}$ (initial solution). A path of solutions is generated linking $\mathbf{x}$ and $\mathbf{y}$. The best solution x^* in this path is returned by the algorithm. Since there is no guarantee that x^* is locally optimal, often local search is applied, starting from x^*, and the resulting locally optimal solution is returned.

Let us denote the set of solutions spanned by the common elements of the n-vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$S(\mathbf{x}, \mathbf{y}) := \{w \text{ feasible} \mid w_i = x_i = y_i, i \notin \Delta(\mathbf{x}, \mathbf{y})\} \setminus \{\mathbf{x}, \mathbf{y}\}. \tag{6}$$

Clearly, $|S(\mathbf{x}, \mathbf{y})| = 2^{n-d(\mathbf{x},\mathbf{y})} - 2$, where $d(\mathbf{x}, \mathbf{y}) = |\Delta(\mathbf{x}, \mathbf{y})|$. The underlying assumption of path-relinking is that there exist good-quality solutions in $S(\mathbf{x}, \mathbf{y})$, since this space consists of all solutions which contain the common elements of two good solutions $\mathbf{x}$ and $\mathbf{y}$. Since the size of this space is exponentially large, a greedy search is usually performed where a path of solutions

$$\mathbf{x} = \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{d(\mathbf{x},\mathbf{y})}, \mathbf{x}_{d(\mathbf{x},\mathbf{y})+1} = \mathbf{y}, \tag{7}$$

is built, such that $d(\mathbf{x}_i, \mathbf{x}_{i+1}) = 1, \quad i = 0, \ldots, d(\mathbf{x}, \mathbf{y})$, and the best solution from this path is chosen. Note that, since both $\mathbf{x}$ and $\mathbf{y}$ are, by construction, local optima in some neighborhood $N(\cdot)$ (Where the same metric $d(\mathbf{x}, \mathbf{y})$ is usually used.), then in order for $S(\mathbf{x}, \mathbf{y})$ to contain solutions which are not contained in the neighborhoods of $\mathbf{x}$ or $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{y}$ must be sufficiently distant from each other.

Figure 5 illustrates the pseudo-code of the path-relinking procedure applied to the pair of solutions $\mathbf{x}$ (starting solution) and $\mathbf{y}$ (target solution). In line 1, an initial solution $\mathbf{x}$ is selected at random among the elite set elements, and usually it differs sufficiently from the guiding solution $\mathbf{y}$. The loop in lines 6 through 14 computes a path of solutions $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{d(\mathbf{x},\mathbf{y})-2}$, local search is applied in line 15, and the solution x^* with the best objective function value is returned in line 16. This is achieved by advancing one solution at a time in a greedy manner. At each iteration, the procedure examines all moves $m \in \Delta(x, \mathbf{y})$ from the current solution x and

```
algorithm Path-relinking(f(·), x, E)
1   Choose, at random, a pool solution y ∈ E to relink with x;
2   Compute symmetric difference Δ(x, y);
3   f* := min{f(x), f(y)};
4   x* := arg min{f(x), f(y)};
5   x := x;
6   while (Δ(x, y) ≠ ∅) →
7       m* := arg min{f(x ⊕ m) | m ∈ Δ(x, y)};
8       Δ(x ⊕ m*, y) := Δ(x, y) \ {m*};
9       x := x ⊕ m*;
10      if (f(x) < f*) then
11          f* := f(x);
12          x* := x;
13      endif;
14  endwhile;
15  x* := LocalSearch(x*, f(·));
16  return (x*);
end Path-relinking
```

Fig. 5 Pseudo-code of a generic path-relinking for a minimization problem

selects the one which results in the least cost solution (line 7), i.e., the one which minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move m^* is made, producing solution $x \oplus m^*$ (line 9). The set of available moves is updated (line 8). If necessary, the best solution x^* is updated (lines 10–13). The procedure terminates when $\mathbf{y}$ is reached, i.e., when $\Delta(x, \mathbf{y}) = \emptyset$, returning the best solution found.

We now describe a possible way to hybridize with path-relinking the basic GRASP described in section "Basic Components". The integration of the path-relinking procedure with the basic GRASP is shown in Fig. 6. The pool $\mathcal{E}$ of elite solutions is initially empty, and until it reaches its maximum size, no path-relinking

```
procedure GRASP+PR(f(·), g(·), MaxIterations, Seed, MaxElite)
1    x_best:=∅; f(x_best):=+∞; E := ∅
2    for k = 1, 2, ...,MaxIterations→
3       x:=ConstructGreedyRandomizedSolution(Seed, g(·));
4       if (x not feasible) then
5          x:=repair(x);
6       endif
7       x:=LocalSearch(x, f(·));
8       if (k ≤MaxElite) then
9          E := E ∪ {x};
10         if (f(x) < f(x_best)) then
11            x_best:=x;
12         endif
13      else
14         x_p:=Path-relinking(f(·), x, E);
15         AddToElite(E, x_p);
16         if (f(x_p) < f(x_best)) then
17            x_best:=x_p;
18         endif
19      endif
20   endfor;
21   return(x_best);
end GRASP+PR
```

Fig. 6 Pseudo-code of a basic GRASP with path-relinking heuristic for a minimization problem

takes place. After a solution $\mathbf{x}$ is found by GRASP, it is passed to the path-relinking procedure to generate another solution. The procedure `AddToElite`$(\mathcal{E}, x_p)$ attempts to add to the elite set of solutions the solution that was just found. Since we wish to maintain a pool of good but diverse solutions, each solution obtained by path-relinking is considered as a candidate to be inserted into the pool if it is sufficiently different from every other solution currently in the pool. If the pool already has `MaxElite` solutions and the candidate is better than the worst of them, then a simple strategy is to have the former replace the latter. Another strategy, which tends to increase the diversity of the pool, is to replace the pool element most similar to the candidate among all pool elements with cost worse than the candidate's.

More formally, in several papers, a solution x_p is added to the elite set $\mathcal{E}$ if either one of the following conditions holds:

1. $f(x_p) < \min\{f(\mathbf{w}) : \mathbf{w} \in \mathcal{E}\}$,
2. $\min\{f(\mathbf{w}) : \mathbf{w} \in \mathcal{E}\} \leq f(x_p) < \max\{f(\mathbf{w}) : \mathbf{w} \in \mathcal{E}\}$ and $d(x_p, \mathbf{w}) > \beta n, \quad \forall \mathbf{w} \in \mathcal{E}$, where β is a parameter between 0 and 1 and n is the number of decision variables.

If x_p satisfies either of the above, it then replaces an elite solution $\mathbf{z}$ no better than x_p and most similar to x_p, i.e., $\mathbf{z} = \text{argmin}\{d(x_p, \mathbf{w}) : \mathbf{w} \in \mathcal{E}$ such that $f(\mathbf{w}) \geq f(x_p)\}$.

Figure 6 shows the simplest way to combine GRASP with path-relinking, which is applied as an intensification strategy to each local optimum obtained after the GRASP local search phase.

More generally, two basic strategies can be used:

i. Path-relinking is applied as a post-optimization step to all pairs of elite solutions;
ii. Path-relinking is applied as an intensification strategy to each local optimum obtained after the local search phase.

Applying path-relinking as an intensification strategy to each local optimum (strategy ii.) seems to be more effective than simply using it as a post-optimization step [111].

Automatic Tuning

An annoying drawback of heuristics is the large number of parameters that need to be tuned for good performance. The tuning phase can take several hundreds of experiments and can be a labor-intensive activity. Moreover, the performance of a heuristic depends on the instance being solved, so a tuned set of parameters obtained for one instance may not result in a good performing heuristic for another instance. When documenting a heuristic, a description of the tuning process is often left out, and therefore it is often difficult to reproduce computational results. These are some

of the factors that point to the need for an algorithmic approach to parameter tuning. Unfortunately, still nowadays very few attempts have been made in the design of efficient automatic tuning procedures for GRASP. In [57,97], the authors proposed a scheme for automatic tuning of GRASP with evolutionary path-relinking heuristics and tested for the generalized quadratic assignment problem and for the set covering, the maximum cut, and the node capacitated graph partitioning, respectively. Similarly to IRACE (Iterated Race for Automatic Algorithm Configuration) [98] and ParamILS [76], both the proposed scheme consist of two phases. In the first phase, a biased random-key genetic algorithm searches the space of parameters for a set of values that results in a good performance of the heuristic. In the second phase, the GRASP+PR heuristic is run using the parameters found in the first phase. For all the optimization problems selected, the authors conducted rigorous experiments whose results showed that the two-phase approach is a robust hybrid heuristic.

Conclusions

The goal of this chapter was to provide an overview of GRASP describing its basic components and enhancements to the basic procedure, including reactive GRASP, intensification strategies, and its hybridizations with different metaheuristics.

Cross-References

- Genetic Algorithms
- Multi-start Methods
- Variable Neighborhood Descent
- Variable Neighborhood Search

References

1. Abdinnour-Helm S, Hadley S (2000) Tabu search based heuristics for multi-floor facility layout. Int J Prod Res 38:365–383
2. Abello J, Pardalos P, Resende M (1999) On maximum clique problems in very large graphs. In: Abello J, Vitter J (eds) External memory algorithms and visualization. DIMACS series on discrete mathematics and theoretical computer science, vol 50. American Mathematical Society, Providence, pp 199–130
3. Ahuja R, Orlin J, Tiwari A (2000) A greedy genetic algorithm for the quadratic assignment problem. Comput Oper Res 27:917–934
4. Aiex R, Binato S, Resende M (2003) Parallel GRASP with path-relinking for job shop scheduling. Parallel Comput 29:393–430
5. Aiex R, Resende M, Pardalos P, Toraldo G (2005) GRASP with path relinking for three-index assignment. INFORMS J Comput 17(2):224–247

6. Alvarez-Valdes R, Parreño F, Tamarit J (2005) A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. J Oper Res Soc 56(4):414–425
7. Alvarez-Valdes R, Parreño F, Tamarit J (2008) Reactive GRASP for the strip-packing problem. Comput Oper Res 35(4):1065–1083
8. Amaldi E, Capone A, Malucelli F (2003) Planning UMTS base station location: optimization models with power control and algorithms. IEEE Trans Wirel Commun 2(5):939–952
9. Andrade D, Resende M (2006) A GRASP for PBX telephone migration scheduling. In: Eighth INFORMS telecommunication conference, Dallas
10. Andrade D, Resende M (2007) GRASP with path-relinking for network migration scheduling. In: Proceedings of the international network optimization conference (INOC 2007), Spa
11. Andres C, Miralles C, Pastor R (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. Eur J Oper Res 187(3):1212–1223
12. Areibi S (1999) GRASP: an effective constructive technique for VLSI circuit partitioning. In: Proceedings of the IEEE Canadian conference on electrical & computer engineering (CCECE'99), Edmonton
13. Areibi S, Vannelli A (1997) A GRASP clustering technique for circuit partitioning. In: Gu J, Pardalos P (eds) Satisfiability problems. DIMACS series on discrete mathematics and theoretical computer science, vol 35. American Mathematical Society, Providence, pp 711–724
14. Argüello M, Feo T, Goldschmidt O (1996) Randomized methods for the number partitioning problem. Comput Oper Res 23(2):103–111
15. Argüello M, Bard J, Yu G (1997) A GRASP for aircraft routing in response to groundings and delays. J Comb Optim 1:211–228
16. Armony M, Klincewicz J, Luss H, Rosenwein M (2000) Design of stacked self-healing rings using a genetic algorithm. J Heuristics 6:85–105
17. Arroyo J, Vieira P, Vianna D (2008) A GRASP algorithm for the multi-criteria minimum spanning tree problem. Ann Oper Res 159:125–133
18. Atkinson J (1998) A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. J Oper Res Soc 49:700–708
19. Bäck T, Fogel D, Michalewicz Z (1997) Handbook of evolutionary computation. Oxford University Press, New York
20. Bard J (1997) An analysis of a rail car unloading area for a consumer products manufacturer. J Oper Res Soc 48:873–883
21. Bard J, Feo T (1989) Operations sequencing in discrete parts manufacturing. Manag Sci 35:249–255
22. Bard J, Feo T (1991) An algorithm for the manufacturing equipment selection problem. IIE Trans 23:83–92
23. Bard J, Huang L, Jaillet P, Dror M (1998) A decomposition approach to the inventory routing problem with satellite facilities. Transp Sci 32:189–203
24. Binato S, Oliveira G (2002) A reactive GRASP for transmission network expansion planning. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 81–100
25. Binato S, Oliveira G, Araújo J (2001) A greedy randomized adaptive search procedure for transmission expansion planning. IEEE Trans Power Syst 16:247–253
26. Binato S, Hery W, Loewenstern D, Resende M (2002) A greedy randomized adaptive search procedure for job shop scheduling. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 58–79
27. Boudia M, Louly M, Prins C (2007) A reactive GRASP and path relinking for a combined production-distribution problem. Comput Oper Res 34:3402–3419
28. Bresina J (1996) Heuristic-biased stochastic sampling. In: Proceedings of the thirteenth national conference on artificial intelligence (AAAI-96), Portland, pp 271–278
29. Canuto S, Resende M, Ribeiro C (2001) Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks 38:50–58

30. Carreto C, Baker B (2002) A GRASP interactive approach to the vehicle routing problem with backhauls. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 185–200
31. Charon I, Hudry O (1993) The noising method: a new method for combinatorial optimization. Oper Res Lett 14:133–137
32. Charon I, Hudry O (2002) The noising methods: a survey. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 245–261
33. Commander C, Festa P, Oliveira C, Pardalos P, Resende M, Tsitselis M (2006) A greedy randomized algorithm for the cooperative communication problem on ad hoc networks. In: Eighth INFORMS telecommunications conference, Dallas
34. Contreras I, Díaz J (2008) Scatter search for the single source capacitated facility location problem. Ann Oper Res 157:73–89
35. Cravo G, Ribeiro G, Lorena LN (2008) A greedy randomized adaptive search procedure for the point-feature cartographic label placement. Comput Geosci 34(4):373–386
36. Delmaire H, Díaz J, Fernández E, Ortega M (1999) Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. INFOR 37:194–225
37. Deshpande A, Triantaphyllou E (1998) A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. Math Comput Model 27:75–99
38. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
39. Faria H, Binato S, Resende M, Falcão D (2005) Power transmission network design by a greedy randomized adaptive path relinking approach. IEEE Trans Power Syst 20(1):43–49
40. Feo T, Bard J (1989) Flight scheduling and maintenance base planning. Manag Sci 35:1415–1432
41. Feo T, González-Velarde J (1995) The intermodal trailer assignment problem: models, algorithms, and heuristics. Transp Sci 29:330–341
42. Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71
43. Feo T, Resende M (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
44. Feo T, Venkatraman K, Bard J (1991) A GRASP for a difficult single machine scheduling problem. Comput Oper Res 18:635–643
45. Feo T, Resende M, Smith S (1994) A greedy randomized adaptive search procedure for maximum independent set. Oper Res 42:860–878
46. Feo T, Sarathy K, McGahan J (1996) A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Comput Oper Res 23:881–895
47. Ferone D, Festa P, Resende M (2013) Hybrid metaheuristics for the far from most string problem. In: Proceedings of 8th international workshop on hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Berlin/New York, pp 174–188
48. Ferone D, Festa P, Resende M (2016) Hybridizations of GRASDP with path-relinking for the far from most string problem. Int Trans Oper Res 23(3):481–506
49. Festa P (2007) On some optimization problems in molecular biology. Math Biosci 207(2):219–234
50. Festa P, Resende M (2002) GRASP: an annotated bibliography. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 325–367
51. Festa P, Resende M (2013) Hybridizations of GRASP with path-relinking. In: Talbi EG (ed) Hybrid metaheuristics – studies in computational intelligence, vol 434. Springer, Berlin/New York, pp 135–155
52. Festa P, Resende MGC (2009) An annotated bibliography of grasp – part I: algorithms. Int Trans Oper Res 16(1):1–24
53. Festa P, Resende MGC (2009) An annotated bibliography of grasp – part II: applications. Int Trans Oper Res 16(2):131–172

54. Festa P, Pardalos P, Resende M (2001) Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. ACM Trans Math Softw 27:456–464
55. Festa P, Pardalos P, Resende M, Ribeiro C (2002) Randomized heuristics for the MAX-CUT problem. Optim Methods Softw 7:1033–1058
56. Festa P, Pardalos P, Pitsoulis L, Resende M (2006) GRASP with path-relinking for the weighted MAXSAT problem. ACM J Exp Algorithmics 11:1–16
57. Festa P, Gonçalves J, Resende M, Silva R (2010) Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm. In: Festa P (ed) Proceedings of 9th international symposium on experimental algorithms. Lecture notes in computer science, vol 6049. Springer, Berlin/New York, pp 338–349
58. Fleurent C, Glover F (1999) Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS J Comput 11:198–204
59. Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Company, New York
60. Ghosh J (1996) Computational aspects of the maximum diversity problem. Oper Res Lett 19:175–181
61. Glover F (1989) Tabu search – part I. ORSA J Comput 1:190–206
62. Glover F (1990) Tabu search – part II. ORSA J on Comput 2:4–32
63. Glover F (1996) Tabu search and adaptive memory programing – advances, applications and challenges. In: Barr R, Helgason R, Kennington J (eds) Interfaces in computer science and operations research. Kluwer Academic Publishers, Boston, pp 1–75
64. Glover F (2000) Multi-start and strategic oscillation methods – principles to exploit adaptive memory. In: Laguna M, Gonzáles-Velarde J (eds) Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research. Kluwer Academic Publishers, Boston, pp 1–24
65. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Boston
66. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 39:653–684
67. Goëffon A, Richer JM, Hao JK (2008) Progressive tree neighborhood applied to the maximum parsimony problem. IEEE/ACM Trans Comput Biol Bioinform 5(1):136–145
68. Goemans M, Williamson D (1996) The primal dual method for approximation algorithms and its application to network design problems. In: Hochbaum D (ed) Approximation algorithms for NP-hard problems. PWS Publishing Co., Boston, pp 144–191
69. Goldberg D (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
70. Gonçalves J, Resende M (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17(5):487–525
71. Hammer P, Rader D Jr (2001) Maximally disjoint solutions of the set covering problem. J Heuristics 7:131–144
72. Hansen P, Mladenović N (1998) An introduction to variable neighborhood search. In: Voss S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics, advances and trends in local search paradigms for optimization. Kluwer Academic Publishers, Boston, pp 433–458
73. Hansen P, Mladenović N (2002) Developments of variable neighborhood search. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer Academic Publishers, Boston, pp 415–439
74. Hart J, Shogan A (1987) Semi-greedy heuristics: an empirical study. Oper Res Lett 6:107–114
75. Hirsch M, Meneses C, Pardalos P, Ragle M, Resende M (2007) A continuous GRASP to determine the relationship between drugs and adverse reactions. In: Seref O, Kundakcioglu O, Pardalos P (eds) Data mining, systems analysis, and optimization in biomedicine. AIP conference proceedings, vol 953. Springer, Melville, pp 106–121
76. Hutter F, Hoos H, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. J Artif Intell Res 36:267–306

77. Kernighan B, Lin S (1970) An efficient heuristic procedure for partitioning problems. Bell Syst Tech J 49(2):291–307
78. Kirkpatrick S (1984) Optimization by simulated annealing: quantitative studies. J Stat Phys 34:975–986
79. Klincewicz J (1992) Avoiding local optima in the p-hub location problem using tabu search and GRASP. Ann Oper Res 40:283–302
80. Klincewicz J, Rajan A (1994) Using GRASP to solve the component grouping problem. Nav Res Logist 41:893–912
81. Kontoravdis G, Bard J (1995) A GRASP for the vehicle routing problem with time windows. ORSA J Comput 7:10–23
82. Laguna M, González-Velarde J (1991) A search heuristic for just-in-time scheduling in parallel machines. J Intell Manuf 2:253–260
83. Laguna M, Martí R (1999) GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J Comput 11:44–52
84. Laguna M, Martí R (2001) A GRASP for coloring sparse graphs. Comput Optim Appl 19:165–178
85. Laguna M, Feo T, Elrod H (1994) A greedy randomized adaptive search procedure for the two-partition problem. Oper Res 42:677–687
86. Li Y, Pardalos P, Resende M (1994) A greedy randomized adaptive search procedure for the quadratic assignment problem. In: Pardalos P, Wolkowicz H (eds) Quadratic assignment and related problems. DIMACS series on discrete mathematics and theoretical computer science, vol 16. American Mathematical Society, Providence, pp 237–261
87. Liu X, Pardalos P, Rajasekaran S, Resende M (2000) A GRASP for frequency assignment in mobile radio networks. In: Rajasekaran S, Pardalos P, Hsu F (eds) Mobile networks and computing. DIMACS series on discrete mathematics and theoretical computer science, vol 52. American Mathematical Society, Providence, pp 195–201
88. Lourenço HR, Paixão J, Portugal R (2001) Multiobjective metaheuristics for the bus-driver scheduling problem. Transp Sci 35:331–343
89. Martí R, Laguna M (2003) Heuristics and meta-heuristics for 2-layer straight line crossing minimization. Discret Appl Math 127(3):665–678
90. Martins S, Ribeiro C, Souza M (1998) A parallel GRASP for the Steiner problem in graphs. In: Ferreira A, Rolim J (eds) Proceedings of IRREGULAR'98 – 5th international symposium on solving irregularly structured problems in parallel. Lecture notes in computer science, vol 1457. Springer, Berlin/Heidelberg, pp 285–297
91. Martins S, Pardalos P, Resende M, Ribeiro C (1999) Greedy randomized adaptive search procedures for the Steiner problem in graphs. In: Pardalos P, Rajasekaran S, Rolim J (eds) Randomization methods in algorithmic design. DIMACS series on discrete mathematics and theoretical computer science, vol 43. American Mathematical Society, Providence, pp 133–145
92. Martins S, Resende M, Ribeiro C, Pardalos P (2000) A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. J Glob Optim 17:267–283
93. Mavridou T, Pardalos P, Pitsoulis L, Resende M (1998) A GRASP for the biquadratic assignment problem. Eur J Oper Res 105:613–621
94. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24: 1097–1100
95. Mockus J, Eddy E, Mockus A, Mockus L, Reklaitis G (1997) Bayesian discrete and global optimization. Kluwer Academic Publishers, Dordrecht/Boston
96. Monkman S, Morrice D, Bard J (2008) A production scheduling heuristic for an electronics manufacturer with sequence-dependent setup costs. Eur J Oper Res 187(3): 1100–1114
97. Morán-Mirabal L, González-Velarde J, Resende M (2013) Automatic tuning of GRASP with evolutionary path-relinking. In: Proceedings of 8th international workshop on hybrid metaheuristics, Ischia. Lecture notes in computer science, vol 7919, pp 62–77

98. nez MLI, Dubois-Lacoste J, Stützle T, Birattari M (2011) The IRACE package, iterated race for automatic algorithm configuration. Technical report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles
99. Osman I, Al-Ayoubi B, Barake M (2003) A greedy random adaptive search procedure for the weighted maximal planar graph problem. Comput Ind Eng 45(4):635–651
100. Pardalos P, Pitsoulis L, Resende M (1996) A parallel GRASP for MAX-SAT problems. Lect Notes Comput Sci 1184:575–585
101. Pardalos P, Pitsoulis L, Resende M (1997) Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. ACM Trans Math Softw 23:196–208
102. Pardalos P, Ramakrishnan K, Resende M, Li Y (1997) Implementation of a variance reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem. SIAM J Optim 7:280–294
103. Pinãna E, Plana I, Campos V, R Martì (2004) GRASP and path relinking for the matrix bandwidth minimization. Eur J Oper Res 153(1):200–210
104. Prais M, Ribeiro C (1999) Parameter variation in GRASP implementations. In: Extended abstracts of the third metaheuristics international conference, Porto, pp 375–380
105. Prais M, Ribeiro C (2000) Parameter variation in GRASP procedures. Investigación Operativa 9:1–20
106. Prais M, Ribeiro C (2000) Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J Comput 12:164–176
107. Pu G, Chong Z, Qiu Z, Lin Z, He J (2006) A hybrid heuristic algorithm for HW-SW partitioning within timed automata. In: Proceedings of knowledge-based intelligent information and engineering systems. Lecture notes in artificial intelligence, vol 4251. Springer, Berlin/Heidelberg, pp 459–466
108. Resende M, Feo T (1996) A GRASP for satisfiability. In: Johnson D, Trick M (eds) Cliques, coloring, and satisfiability: the second DIMACS implementation challenge. DIMACS series on discrete mathematics and theoretical computer science, vol 26. American Mathematical Society, Providence, pp 499–520
109. Resende M, Ribeiro C (1997) A GRASP for graph planarization. Networks 29:173–189
110. Resende M, Ribeiro C (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic Publishers, Boston, pp 219–249
111. Resende M, Ribeiro C (2005) GRASP with path-relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers. Springer, New York, pp 29–63
112. Resende M, Pardalos P, Li Y (1996) Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. ACM Trans Math Softw 22:104–118
113. Resende M, Pitsoulis L, Pardalos P (1997) Approximate solution of weighted MAX-SAT problems using GRASP. In: Gu J, Pardalos P (eds) Satisfiability problems. DIMACS series on discrete mathematics and theoretical computer science, vol 35. American Mathematical Society, Providence, pp 393–405
114. Resende M, Pitsoulis L, Pardalos P (2000) Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. Discret Appl Math 100:95–113
115. Ribeiro C, Resende M (1999) Fortran subroutines for approximate solution of graph planarization problems using GRASP. ACM Trans Math Softw 25:341–352
116. Ribeiro C, Souza M (2002) Variable neighborhood search for the degree constrained minimum spanning tree problem. Discret Appl Math 118:43–54
117. Ribeiro C, Urrutia S (2007) Heuristics for the mirrored traveling tournament problem. Eur J Oper Res 179:775–787
118. Ribeiro C, Uchoa E, Werneck R (2002) A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J Comput 14:228–246

119. Ríos-Mercado R, Bard J (1998) Heuristics for the flow line problem with setup costs. Eur J Oper Res 110(1):76–98
120. Ríos-Mercado R, Bard J (1999) An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup costs. J Heuristics 5:57–74
121. Rivera L (1998) Evaluation of parallel implementations of heuristics for the course scheduling problem. Master's thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, Monterrey
122. Robertson A (2001) A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. Comput Optim Appl 19: 145–164
123. Sosnowska D (2000) Optimization of a simplified fleet assignment problem with metaheuristics: simulated annealing and GRASP. In: Pardalos P (ed) Approximation and complexity in numerical optimization. Kluwer Academic Publishers, Boston
124. Srinivasan A, Ramakrishnan K, Kumaram K, Aravamudam M, Naqvi S (2000) Optimal design of signaling networks for Internet telephony. In: IEEE INFOCOM 2000, Tel-Aviv, vol 2, pp 707–716
125. Takahashi H, Matsuyama A (1980) An approximate solution for the Steiner problem in graphs. Math Jpn 24:573–577
126. Urban T (1998) Solution procedures for the dynamic facility layout problem. Ann Oper Res 76:323–342

Hyper-heuristics

17

Michael G. Epitropakis and Edmund K. Burke

Contents

Introduction . . . 490
Recent Advances in the Field of Hyper-heuristics . . . 493
Recent Advances on the Methodology of Hyper-heuristics . . . 494
Automatic Design of Algorithms . . . 497
Recent Theoretical Results in Hyper-heuristics . . . 498
Multi-objective Optimization . . . 499
Hyper-heuristics in Dynamic and Uncertain Environments . . . 500
Machine Learning Methodologies . . . 501
Automated Parameter Control and Tuning . . . 502
Hyper-heuristics for Scheduling Problems . . . 503
Recent Educational Timetabling Applications of Hyper-heuristics . . . 507
Games and Education . . . 508
Other Recent Developments . . . 508
Case Study: A Selection Hyper-heuristic-Based Iterated Local Search . . . 509
The HyFlex Framework . . . 509
Iterated Local Search with Adaptive Heuristic Selection . . . 510
Action Selection Models . . . 515
Computational Results . . . 518
Experimental Protocol . . . 518
Experimental Results . . . 520
Comparison to HyFlex State-of-the-Art Hyper-heuristics . . . 532
Conclusions . . . 534
Cross-References . . . 536
References . . . 537

M. G. Epitropakis (✉)
Data Science Institute, Department of Management Science, Lancaster University Management School, Lancaster University, Lancaster, UK
e-mail: m.epitropakis@lancaster.ac.uk

E. K. Burke
School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK
e-mail: vp-se@qmul.ac.uk

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_32

Abstract

This chapter presents a literature review of the main advances in the field of hyper-heuristics, since the publication of a survey paper in 2013. The chapter demonstrates the most recent advances in hyper-heuristic foundations, methodologies, theory, and application areas. In addition, a simple illustrative selection hyper-heuristic framework is developed as a case study. This is based on the well-known Iterated Local Search algorithm and is presented to provide a tutorial style introduction to some of the key basic issues. A brief discussion about the implementation process in addition to the decisions that had to be made during the implementation is presented. The framework implements an action selection model that operates on the perturbation stage of the Iterated Local Search algorithm to adaptively select among various low-level perturbation heuristics. The performance and efficiency of the developed framework is evaluated across six well-known real-world problem domains.

Keywords

Hyper-heuristics · Heuristics · Meta-heuristics · Evolutionary computation · Optimization · Search · Machine learning · Multi-objective optimization · Combinatorial optimization · Black box optimization · Dynamic optimization · Scheduling · Timetabling · Packing · Iterated local search

Introduction

Over the last 50 years or so, heuristic search methodologies have been successfully applied to address various real-world complex optimization problems. Computationally challenging optimization problems arise in various disciplines such as operational research, computer science, finance, bioinformatics, industrial informatics, engineering, and data sciences. Representative examples of optimization problems can be found in scheduling, timetabling, planning, resource and space allocation, cutting and packing, software design, hardware design, and engineering design problems. Such optimization problems can be addressed by exact or heuristic methodologies (or a hybridization). The main difficulty of solving such problems occurs from the often remarkably large and possibly heavily constrained search space. In such problems, or in noisy and dynamic real-world problem scenarios, finding optimal solutions by exact methods might be impossible. Therefore, in practice, heuristic search methodologies are often developed or applied that seek to find solutions of acceptable quality in reasonable time, but without having any guarantee of optimality.

Over the last few decades, a broad spectrum of different heuristics have been proposed and successfully applied to a wide variety of problems. However, heuristics are usually designed to solve specific optimization problems. Such heuristics are bespoke problem-specific methods that often demand significant time to design and

tune. They are also often expensive to implement and maintain. Hyper-heuristics have been motivated by the aim of raising the level of generality at which search algorithms can operate, as well as automating the design and tuning of heuristic algorithms [32–34,36,38,102,106,107]. The main characteristic of a hyper-heuristic method is that it operates on a space of heuristics rather than on a space of solutions. As such, a hyper-heuristic methodology attempts to find or generate the appropriate method or sequence of low-level heuristics instead of directly solving a particular problem. A hyper-heuristic can be characterized as a high-level methodology which automatically generates or combines low-level search components (heuristics) to effectively solve a given problem instance or class of problems.

The following definition has been provided in the literature:

> A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems [34,36].

The use of the term of *hyper-heuristics* goes back to 2001 [40]. However, the idea of automating the design and/or selection of heuristics has existed in the literature for more than 50 years [41, 52]. It is possible to characterize hyper-heuristic methodologies by considering two dimensions [34]: the characteristics of the heuristics' search space and the feedback information from the search process. The heuristic search space can be further divided into heuristic selection and heuristic generation. The first category represents methodologies or models that select among a set of available heuristics, while the latter category covers methodologies that generate new heuristics from existing heuristic components. A further refinement in both categories can be made based on the characteristics of the utilized search heuristic, i.e., constructive and perturbative search paradigms [60].

Hyper-heuristics can be further distinguished by looking at how they handle feedback information from the search process. Two main categories can be currently observed: methodologies that learn from the feedback and the ones that do not learn from it. The first category can be further refined based on the source of learning: *online* or *off-line* learning. When considering online learning hyper-heuristics, the learning phase takes place during the search procedure. Thus, timely feedback information can be exploited by the learning mechanism to study the trends and behavior of the underlying components. In the case of off-line learning (off-line learning hyper-heuristics), all information about the process is known a priori. As such, the learning mechanism is able to gather knowledge and learn from a training set of problem instances and then apply its knowledge on a test set of unseen instances. The aim of such a procedure is to generalize from the training information to unseen problem instances in an expected way. Methodologies that do not learn from the feedback information usually simply discard it.

The aforementioned classification reflects the majority of the past and the current research trends that can be widely found in the literature. Nevertheless, interesting example methodologies can be found that represent an interplay between the categories of heuristics, i.e., combinations of selection and generation heuristic search procedures [112,122].

Over the last few years, various introductory tutorials, review, and survey articles have been published on the area of hyper-heuristics. The first book chapter that appeared in 2003 [32] introduces the general idea of hyper-heuristics and provides a brief historical overview of the field at that time. In particular, the chapter focuses on one of the main objectives of hyper-heuristics, which is to increase the level of generality at which a search methodology can operate. The overview demonstrates the history as well as the developments and trends in the field at that time, through detailed descriptions of the representative applications and methodologies.

An introductory tutorial of the area was presented in [106] that effectively demonstrated the development process of a hyper-heuristic approach, through useful guidelines and characteristic examples. It additionally pinpoints potential application areas and discusses some issues and possible future research pointers in the area. A second-edition tutorial with enriched material and information was recently published in [107].

A classification of the field and a discussion of developments at that time were published in 2008 [38]. The chapter emphasizes the real-world applications that have been tackled by hyper-heuristic approaches and presents three criteria that characterize a hyper-heuristic methodology: operating at a higher level and managing low-level heuristics, searching the heuristic space rather than the solution space, and limited access to problem domain information, i.e., an increased level of generality. The authors identified the last criterion as a particularly important one for the general applicability and robustness of a hyper-heuristic approach.

An overview chapter was published in 2009 [33] that discusses hyper-heuristic methodologies that are able to generate new effective heuristics that are not known beforehand. The chapter addresses the concept of automatically designing heuristics and describes a methodology to generate new heuristics. Several representative case study examples are also provided. In addition, the presented methodology emphasizes the main algorithmic component of this class which is the genetic programming algorithm. Moreover, it briefly covers the related literature and discusses the characteristics and issues of this class of approaches.

A unified classification and definition of the field were presented in 2010 [34]. The chapter provides an overview of the previous categorizations of the area and determines a unified classification and definition of hyper-heuristics that captures the current and potential future directions of hyper-heuristics. Two main classes of hyper-heuristic approaches are recognized in the categorization, heuristic selection and generation approaches. Several characteristic examples are presented and discussed for both categories.

A recent survey of hyper-heuristics was published in 2013 [36]. The article presents the state of the art of the field up to 2012 by providing examples of the main and most representative methodologies and application areas. It also discusses the origins and intellectual roots of the field as well as provides a critical discussion of the literature. Various potential research trends are briefly discussed, and several future directions are highlighted. Finally, the most recent overview of hyper-heuristics was published in 2014 [102]. The article presents an overview

and critical analysis of hyper-heuristics specifically for educational timetabling problems.

Building upon the aforementioned overview articles that have been recently published, this chapter aims to provide a thorough literature review of the main very recent advances in the field of hyper-heuristics, since the publication of [36] in 2013. An additional objective is to provide a case study for the development process of a simple but highly efficient selection hyper-heuristic framework as a tutorial-style introduction to the field. The performance and efficiency of the developed framework is evaluated across six well-known real-world problem domains.

The remainder of the chapter is organized into three main parts. The first part will present a timely overview of the current developments in the field of hyper-heuristics (section "Recent Advances in the Field of Hyper-heuristics"). The second part presents the development of a simple selection hyper-heuristic framework as a case study (section "Case Study: A Selection Hyper-heuristic-Based Iterated Local Search"). The last part evaluates the generality and performance of the developed framework on six different problem domains (section "Computational Results"). The chapter concludes with a brief discussion that summarizes the presented work (section "Conclusions").

Recent Advances in the Field of Hyper-heuristics

This literature review covers and discusses the latest articles that have been published in the field. It aims to cover as many of the current advances and trends as possible, without presenting overlapping material with relatively recent overview and survey papers [32–34, 36, 38, 102, 106, 107]. The main aim of this review is to cover the period since the publication of [36] in 2013.

The latest trends and directions regarding foundational and methodological aspects of hyper-heuristics are firstly presented in the literature review (section "Recent Advances on the Methodology of Hyper-heuristics"). Then it proceeds with discussion on the automatic design of algorithms (section "Automatic Design of Algorithms") and the latest theoretical works in the area (section "Recent Theoretical Results in Hyper-heuristics"). Hyper-heuristic developments in multi-objective and dynamic problem formulations are then reviewed (sections "Multi-objective Optimization" and "Hyper-heuristics in Dynamic and Uncertain Environments"). The intersection between machine learning and hyper-heuristic methodologies is studied in section "Machine Learning Methodologies", while section "Automated Parameter Control and Tuning" presents the recent advancements in hyper-heuristics for automatic parameter control. Finally, the most recent developments in application areas of hyper-heuristics are presented that include scheduling, timetabling, games, and various other areas (sections "Hyper-heuristics for Scheduling Problems", "Recent Educational Timetabling Applications of Hyper-heuristics", "Games and Education", and "Other Recent Developments").

Recent Advances on the Methodology of Hyper-heuristics

Starting from the foundations of hyper-heuristics and by studying the classifications of the area, the authors of [128] reformulate the mathematical definition of a hyper-heuristic. It is specifically exhibited that the new presented formulation naturally leads to a recursive definition of a hyper-heuristic. The authors draw parallels between the new hyper-heuristic formulation and a blackboard architecture from artificial intelligence, in which heuristics are able to annotate a shared workspace with information that can be exploited by other heuristics. In such a formulation, heuristics can share information at any level, which suggests that the domain barrier can be relaxed at any level required, without the loss of generality. A detailed description of the proposed formulation and its developments can be found in [128]. A concrete example application on the 3-SAT problem domain has been presented as a case study, which exhibits the improvements of the proposed idea over a wide set of problem instances. Other extensions of the existing formulations of hyper-heuristics have been proposed in [108]. This study advocates the need to reconsider the existing approaches to be able to produce self-organizing heuristics that will be able to automatically self-adapt to the environment when the underlying problem domain changes.

The need for more general and cross-domain efficient methodologies [36, 103, 107] has led to novel studies that try either to introduce unified representations or to embed common domain knowledge across various problem domains [49, 129].

In particular, the authors in [49] do not employ a high-level barrier across domains that are used in much of the hyper-heuristic literature. Instead, they propose an evolutionary memetic computing paradigm that is capable of embedding domain knowledge in memes across different but related problem domains. To improve search efficiency, the memetic paradigm is capable of learning and evolving knowledge memes on two different problem domains that can share a common representation, the capacitated vehicle and arc routing problems. Thorough analyses and experimental results show that the evolutionary search can benefit from different relevant problem domains. The knowledge that a meme has is crucial for the search procedure. It was observed that low discrepancy in the common feature space, between the problem domains, is able to enhance the evolutionary search procedure.

Furthermore, a general-purpose hyper-heuristic approach has been proposed that has the ability to solve combinatorial problems with minimal effort [129]. A unified representation and a set of domain-independent low-level heuristics are essentially proposed under a hyper-heuristic framework. To demonstrate the generality of the approach, the framework has been applied on different problem domains that share common characteristics but vastly differ as applications. An analysis reveals that the unified framework is very competitive in performance with state-of-the-art tailor-made heuristics for each problem domain.

The aforementioned studies suggest that both high and low levels of abstraction in the problem domain can enhance the search efficiency of hyper-heuristics. Such approaches demonstrate the real power of hyper-heuristic techniques. It is possible

to develop approaches that have enough generality and efficiency to enable them to produce competitive performance against tailor-made search algorithms for either a given problem class or across problem domains.

Various recent developments include novel hyper-heuristics that operate across the main categories of hyper-heuristics. Such an approach was introduced in [112]. It combines both heuristic selection and generation approaches. Specifically, it incorporates a dynamic multi-armed bandit with extreme credit values to online select the most appropriate low-level heuristic for the problem on hand. In parallel, instead of using a fixed acceptance criterion, it gradually generates new acceptance criteria for each problem instance through a gene expression programming framework. The effectiveness and generality of the approach are evaluated on exam timetabling instances from the 2007 International Timetabling Competition and on dynamic vehicle routing problem instances from the literature. Empirical results suggest that the introduced method shows competitive and general performance across both domains, compared with several state-of-the-art algorithms.

An automatic design framework was introduced in [110]. It automatically generates high-level heuristics for hyper-heuristic methodologies. The framework utilizes a gene expression programming algorithm to create high-level heuristics during the searching process. It captures information from the current state of the given problem and determines the low-level heuristic selection rule, as well as the solution acceptance criterion. As such, possibly new high-level rules can be produced for each problem instance. Experimental analyses on six problem domains exhibit its generality and competitive performance with state-of-the-art hyper-heuristic methodologies across all problem domains.

Techniques have been developed that are capable of learning and self-adapting their behavior based on the learned knowledge. More specifically, a hyper-heuristic system that embraces the concept of lifelong learning has been considered in [57, 120–122]. The system is based on a self-reliant network of interacting components inspired by artificial immune systems. It has the ability to continuously learn over time how to solve combinatorial optimization problems. It is capable of generating new heuristics continuously as well as sample problems from its environment. The core of the system is adaptable to its environment and has the intelligence to save and learn characteristic problem instances and heuristics from it. These characteristics result in an intelligent platform that exploits existing knowledge and quickly generates promising solutions of a given problem, as well as generalizing and adapting to new unseen problems with different properties. A rigorous experimental analysis on the bin packing domain demonstrates its excellent performance and generalization capabilities across a wide range of different problem instances. The dynamic and adaptive nature of the system makes it computationally efficient and scalable.

Various studies have been proposed that carefully investigate intrinsic characteristics of hyper-heuristics, such as diversity in the heuristic space, or understanding the behavior of the low-level heuristics. In particular, in [55], the impact and effect of a diverse set of available heuristic algorithms on the performance of a selection hyper-heuristic is studied. The study firstly makes an attempt to define

and measure the diversity of heuristic spaces and then proceeds to investigate the impact in performance of a hyper-heuristic that utilizes various strategies to manage the measured diversity. The diversity concept is studied with a hyper-heuristic that employs as low-level heuristic various black box optimization methods that operate on continuous spaces (see also [54]). The evaluation of the proposed method on a wide range of benchmark functions suggests that the heuristic space diversity significantly impacts upon the performance of the method.

The performance of low-level heuristics plays a crucial role in the performance of a hyper-heuristic methodology. In [115], the authors try to understand the performance efficiency of each low-level heuristic when it operates individually or in combination with other heuristics. As such, the concepts of competence and affinity (in terms of heuristics) have been defined and qualitatively estimated on a problem-specific hyper-heuristic approach. The resulting information from this process is exploited and used to improve the performance of problem-specific hyper-heuristics for the hybrid flow-shop scheduling problem. The tailor-made hyper-heuristics exhibit promising performance on the studied instances.

In parallel, novel search approaches have recently been incorporated within hyper-heuristic approaches. A novel diversification method is introduced in [105]. The method perturbs the problem instance under investigation to create justifiable new search directions. As such, various low-level heuristics that act upon this idea are proposed under a hyper-heuristic framework. The low-level heuristics are guided through an expressive grammar high-level strategy that has the ability to easily satisfy the problem constraints and thus easily produce feasible solutions. An experimental evaluation and analyses on the Ising spin glass and the p-median problem demonstrate its efficiency and its competitive performance.

When considering selection hyper-heuristics, each of the available search low-level heuristics (to be selected) is scored by a credit assignment mechanism. The credit assignment mechanism is responsible for assigning a reward, or credit value, to the applied heuristic, based on measurable feedback from the search procedure. The feedback may represent a measurement or qualitative characterization of the applied heuristic effects on the current state of the search process. The role of the credit value is crucial for the performance of the hyper-heuristic search algorithm, since it is the main information that guides the search procedure. Recently, different new types of feedback information have been introduced, based on landscape analysis techniques.

In [124], the evolvability of a search operator has been considered. The evolvability metric uses local characteristics of the fitness landscape to approximate the potential of a search component to generate improved solutions, from the current state of the search. An experimental study on benchmark problems reveals that evolvability feedback has significant potential to improve the performance of a hyper-heuristic, when compared against simple fitness improvement feedback.

Additionally, a novel selection hyper-heuristic methodology has been introduced in [39] to address the Capacitated Arc Routing problem. In the higher level, an online learning algorithm has been considered, the Dynamic Weighted Majority

method, to predict and determine the most appropriate search mechanism. The learning mechanism selects a crossover operator from an available set. Each of the available choices does not rely on its ability to improve the fitness of the generated solution. Instead, each operator can be scored based on four different fitness landscape analysis techniques. The analysis takes into account dispersion and neutrality as well as evolvability measures. Extensive comparisons on a set of benchmark problems verify the efficiency of the proposed methodology against state-of-the-art techniques.

Automatic Design of Algorithms

A comparison of the fields of meta-learning and hyper-heuristics has been recently proposed in [100]. The study draws parallels between methodologies and concepts from the two fields and advocates the common objective of automating the algorithm design process. A brief historical overview of automated algorithm design is presented along with a discussion on the similarities and differences between supervised machine learning and hyper-heuristics. Various issues are discussed, including the different levels of automation and generality, the problem and algorithm space, and various measures of performance.

The automatic design process was used to generate simple, general, modular, and efficient Iterated Local Search-based methodologies in [5, 6]. The generated methodologies are very competitive with the state-of-the-art methodologies in domain-independent meta-heuristic search [6]. Other hyper-heuristics based on Iterated Local Search that have been recently proposed include [97] in addition to the case study presented in the sections below. In parallel, a novel grammar representation has been introduced in [90] to be used for the automatic design of heuristic algorithms for hard combinatorial optimization problems. The proposed representation of the grammar includes a sequence of categorical, integer, and real-valued parameters and incorporates an off-line parameter configuration tool to search for algorithms for a given problem. Extensive experimentation on one-dimensional bin packing and on permutation flow-shop problems demonstrates the efficiency of both the grammar and the parameter configuration tools against an established grammatical evolution algorithm for automatic design. An extension of the methodology was able to build more complex algorithmic schemes from a composition of problem-dependent and problem-independent grammars [85].

A unified selection hyper-heuristic framework that handles one- and two-dimensional regular packing as well as two-dimensional irregular packing and cutting problems has been proposed in [80]. The framework utilizes several different stages to generate fast algorithms with competitive performance against the best problem-specific heuristic for a given problem. The framework generates different hyper-heuristic algorithms, by representing them as variable-length chromosomes of low-level heuristics that are being evolved by a messy genetic algorithm. A data mining-based feature selection procedure is used to determine the most relevant

features in the problem-state representation, and an off-line analysis is employed to discover the most representative set of well-performing heuristics. The framework has been evaluated on a large set of problem instances, and the experimental results show that it is able to generate very competitive hyper-heuristics for the given problem. In addition, the performance of the produced hyper-heuristics is comparable or even better than the application of the single low-level heuristics on their own. Moreover, an automatic design process has been proposed to efficiently solve two-dimensional rectangular blocks for packing [132].

Recently, the commonalities of certain areas of memetic computing and hyper-heuristics have been reviewed in [48]. This has led to the development of a new framework that combines a multi-meme approach with an adaptive operator selection hyper-heuristic to address continuous optimization problems. The framework rewards and adaptively selects the most suitable heuristics/memes during the search procedure. Although it has a simple structure, it exhibits high-performance gains against state-of-the-art search algorithms in a wide range of benchmark problems.

To avoid stagnation to local optima solutions, a multi-objective formulation has been proposed to formulate the single-objective two-dimensional packing problem [116]. A new memetic algorithm along with a parallel hyper-heuristic approach has been introduced to search the new formulation of the problem. The behaviors of the proposed methodologies are analyzed, and experimental results on two-dimensional packing problem instances suggest that the parallel hyper-heuristic is very competitive since it has found new best solutions for some problem instances.

The effectiveness and generality of a well-designed hyper-heuristic approach that outperforms human-designed problem-specific algorithms are demonstrated in [125]. Specifically, an Iterated Local Search-based hyper-heuristic methodology that learns from feedback of the problem at hand and dynamically selects among the available search operators is developed. An off-line learning approach is compared and contrasted with an online learning model on a large set of real-world problem instances from the first and second international timetabling competition. The online learning approach demonstrates superior performance over the off-line (static) model and is competitive with the state-of-the-art algorithms in the field.

Recent Theoretical Results in Hyper-heuristics

Although there is a significant growth of research in the field of hyper-heuristics, most studies are empirical. The theoretical foundations of hyper-heuristics are largely unexplored. Prominent examples of recent theoretical work in the field include mostly runtime analysis. A rigorous runtime analysis of hyper-heuristics has been presented in [74]. The analysis considers selection hyper-heuristics that operate on a predefined set of low-level heuristics. It reveals that the combination of heuristics can lead to exponentially faster search than deterministically chosen heuristics for a given problem. However, the analysis shows that an appropriate mixing distribution should be carefully selected.

A runtime analysis on some very common learning mechanisms has been performed in [8]. In particular, the analysis considers the simple random, random gradient, greedy, and permutation learning mechanisms in the case of selection hyper-heuristics. The analysis shows that the learning mechanisms behave similarly, which suggests that they do not necessary improve the performance of the hyper-heuristics in the studied cases.

Multi-objective Optimization

A wide range of real-world problems require high-quality decisions for more than one objective. In general, an effective multi-objective algorithm should possess good search characteristics in terms of both solution diversity and convergence on the Pareto-front set. Several very effective multi-objective search algorithms have been proposed in the literature. They have different search dynamics, characteristics, advantages, and disadvantages. Hyper-heuristic approaches have also been explored within the context of multi-objective search (multi-objective hyper-heuristics).

The choice function approach has been used as a selection hyper-heuristic for a number of years. Interestingly, it still continues to be a promising and effective approach. The choice function along with the great deluge and late acceptance as nondeterministic move acceptance mechanisms has been investigated to address multi-objective optimization problems in [84]. The method utilizes the hyper-volume metric during the move acceptance process to effectively guide the search in promising areas and enhance both diversity and convergence characteristics of the method. The proposed method has been applied to traditional multi-objective benchmark problems in addition to the vehicle crashworthiness design problem [78]. Experimental analyses and results indicate that the proposed hyper-heuristic approach has the ability and potential to efficiently solve continuous multi-objective optimization problems.

Similarly, a hyper-heuristic method that combines the strengths of three well-known multi-objective algorithms has been introduced in [83]. The proposed approach implements an online learning choice function as a selection hyper-heuristic to combine the underlying algorithms. The choice function adaptively ranks the performance of the algorithms and determines which algorithm to apply at the next step. The introduced hyper-heuristic framework exhibits very effective performance on a wide range of multi-objective benchmark functions and on a real-world problem case, the vehicle crashworthiness design problem. The employed online learning mechanism greatly enhances the performance of the hyper-heuristic algorithm which is able to efficiently combine and exploit the advantages of the multiple low-level multi-objective algorithms.

In parallel, a multiple evolutionary algorithm portfolio has been developed in [139]. This approach utilizes simultaneously multiple algorithms for solving multi-objective optimization problems. The performance of each algorithm is evaluated by a score function, and the hyper-heuristic approach selects which to apply in the next step based on its score. The developed approach exhibits

promising results. Another example of a recently proposed hyper-heuristic has been presented in [93]. This method includes diversity-based techniques to solve the patrol scheduling problem.

The concept of adaptive operator selection in multi-objective algorithms was introduced in [76]. This utilizes state-of-the-art models for adaptive operator selection to select the most suitable operator at each stage of the algorithm. The methodology extends the well-known MOEA/D algorithm, which decomposes the multi-objective problems to single-objective problems and optimizes them simultaneously. Thorough experimental analyses indicate that the proposed methodology is robust, its performance is significantly better than other state-of-the-art multi-objective algorithms, and the operator selection process works effectively. Likewise, a hyper-heuristic approach has been recently incorporated into the Multi-Objective Particle Swarm Optimization algorithm [37]. This is a particle swarm optimization variant for multi-objective problems. Its main characteristic is that it has to employ a leader selection strategy and an archiving mechanism.

In [91], a genetic programming-based hyper-heuristic is proposed to solve the bi-objective aspect of the water distribution network design problem. The goal here is to find the optimal pipe sizes required by a network to provide best service with minimal costs. The objective of the proposed hyper-heuristic methodology is to generate efficient search operators (mutation operators) for multi-objective algorithms applied to solve the bi-objective formulation of the problem. Interestingly, the generated mutation operators reveal useful information for the designer. For example, an evolved operator incorporates domain-specific knowledge (pipe smoothing), while others are able to reproduce well-known operators from the literature. The methodology exhibits significant potential on various problem instances, while the evolved operators are general and can be used as new algorithmic components for future multi-objective algorithms. A further analysis of the strengths of various search operators along with the features of the problem is provided in [92].

Hyper-heuristics in Dynamic and Uncertain Environments

In the last few years, hyper-heuristic methodologies have been employed to address applications that are inherently dynamic and uncertain. This poses a major challenge for the research community since the majority of the developed methodologies that operate on static environments cannot easily cope with the dynamic and uncertain aspects of some applications. Various recent hyper-heuristic methodologies embrace adaptivity and are able to cope with dynamically changing problem scenarios.

A hybrid hyper-heuristic has been introduced to consider dynamic optimization problems [133]. This combines a selection hyper-heuristic and a memory/search algorithm. The hyper-heuristic is able to select between the available low-level heuristics for the problem instance in hand. To evaluate its performance efficiency, the following two problems have been considered: the dynamic generalized assignment problem and the moving peaks benchmark. Experimental results indicate that

the hyper-heuristic framework performs significantly better than the memory/search algorithm in the majority of the tested cases. From the considered selection heuristics, the choice function with the only improving acceptance criterion seems to have the most promising performance, since it is able to track changes in the environment and it can rapidly react to different types of changes.

An additional recent study on hyper-heuristics in dynamic environments includes [126]. This approach analyzes the performance of a hyper-heuristic methodology that utilizes specialized meta-heuristics across different types of dynamic environment. An interesting real-world problem with uncertainty has recently been efficiently addressed by a hyper-heuristic approach [7]. This is the online path planning for autonomous unmanned aerial vehicles. Online path planning is a challenging problem that includes problem scenarios in uncertain or unknown environments. A hyper-heuristic methodology has been implemented to navigate these unmanned vehicles using a 3-D online path planning model with sensing uncertainty. An experimental study on various terrains with different characteristics demonstrates the efficiency of the methodology to create efficient navigation path plans.

Machine Learning Methodologies

Machine learning methodologies play an important role in the field. Various methodologies have been incorporated mostly into the higher level of hyper-heuristics to learn the behavior of the underlying heuristics and to make effective decisions to guide the search procedure toward the most promising areas of the search space. In parallel, the automatic design concept in hyper-heuristics is widely applicable in the machine learning field to generate novel and more efficient machine learning methodologies for specific classes of problems or tasks. Recent advances include a wide range of methodologies.

A new machine learning approach based on tensor analysis was introduced in [18, 19]. It utilizes tensor analysis techniques and in particular tensor factorization to reveal latent relationships between the low-level heuristics and the hyper-heuristic. Intuitively, it models the history trace of the considered hyper-heuristic method and tries to identify the low-level heuristics that exhibit promising performance. Tensor analysis and factorization are employed to identify promising interactions and features between low-level heuristics as the search method operates, by observing feedback from the search history and performance. The method then exploits the knowledge learned to improve its overall search ability.

Inspired by Inverse Reinforcement Learning theory, the authors in [17, 21] explored apprenticeship learning for generalizing experts' behavior. Apprenticeship learning or imitation learning or learning by demonstration has been widely applied in control and robotic applications. The main goal of apprenticeship learning is to learn by observing experts while they are in action. Within the context of hyper-heuristics, apprenticeship learning has been applied to learn expert behavior in vehicle routing [17] and online bin packing problem domains [21]. The learning

procedure has been trained with diverse experts on small problem instances, and it has been tested on larger instances with different characteristics to test its generalization ability. The learning procedure successfully captures experts' different actions and generalizes them on unseen problem instances. Experimental results and analyses suggest that the novel learning mechanism demonstrates outstanding performance gains in both problem domains.

Monte Carlo Tree search has been used in [111] to search the space of low-level heuristics for various problem domains. In particular, it models sequences of low-level heuristics as trees and searches for optimal sequences to be applied at the corresponding state of the search procedure. Experimental results on six problem domains demonstrate the generality of the method as well as its competitive performance.

An automated design of classification algorithms tailored to deal with both balanced and imbalanced data has been proposed in [25]. This aims to automatically design decision-tree induction algorithms. A thorough analysis is firstly performed to determine the impact of different kinds of fitness functions on the performance of the proposed methodology for both balanced and imbalanced datasets. The best-performing fitness function is then used for the automatic design process. The evolved decision-tree induction algorithm is compared against traditional decision-tree induction algorithms on a wide selection of datasets. The experimental analysis suggests that the proposed method significantly outperforms the other methods.

A grammatical evolution-based hyper-heuristic has recently been introduced for designing automatically split criteria in decision trees for data classification tasks [27]. A hyper-heuristic evolutionary algorithm to automatically build Bayesian Network Classifiers tailored to a specific dataset [109] represents another recent addition to the literature.

Automated Parameter Control and Tuning

Automated parameter control represents one of the major applications of hyper-heuristics. A recent overview of the trends and future directions of the area can be found in [67].

Two interesting parameter control methods based on hyper-heuristics and fuzzy logic have been recently proposed in [118, 119]. They investigate the impact of solving single-objective optimization problems with multi-objective approaches. The utilized methodology is a bi-objective approach that uses diversity as a second objective. A probabilistic selection hyper-heuristic and a fuzzy logic method have been utilized to control the parameters of the bi-objective methodology. Experimental analysis on a wide range of problems reveals the efficiency of both parameter control methods against several static parameter settings and self-adaptive rules. Overall, the resulting method is computationally efficient and is able to outperform the single-objective scheme in most of the tested cases. A similar approach has been applied to address the frequency assignment problem [117]. Additionally,

fuzzy system methods have been investigated as selection hyper-heuristics to control parameters [62]. Off-line parameter tuning has been employed to fine-tune and successfully generate efficient heuristics for online bin packing [137].

A reinforcement learning-based parameter controller has recently been published in [64] to dynamically adapt the control parameters of evolutionary algorithms. The method is a generic and parameter-independent controller that can be easily adapted on any evolutionary algorithm. Experimental analysis that incorporates the proposed parameter control method on various state-of-the-art evolutionary algorithms, to address several continuous optimization problems, suggests that the method is capable of good performance. The controller is able to improve the quality of the solutions found by the algorithms with minimal effort and additional computational resources. Similar approaches include entropy-based controllers [9]. Furthermore, a recent study investigates the impact in performance of various feedback mechanisms that have been used for adaptive parameter control in evolutionary algorithms [10]. The study considers indicative feedback measures for both single- and multi-objective optimization problem formulations.

Time series prediction methodologies have recently been used [11] as parameter controllers to accurately predict suitable parameter values during the search process. In particular, various prediction methods have been utilized to predict future parameter values based on historical data from the search procedure. Evaluation of the considered methodologies on evolutionary algorithms suggests that the simple linear regression performs quite well, but without having a significant difference against the studied methods. The impact in performance of predictive parameter control methods is evident only when the performance data complies with the required assumptions of the prediction model, resulting in significant performance differences when compared against state-of-the-art parameter controllers. Otherwise, the prediction method does not exhibit any evident impact on the performance of the considered algorithm.

Hyper-heuristics for Scheduling Problems

As widely identified in previously published overview and survey articles [32–34, 36, 38, 106, 107], scheduling problems have played a major role in the development of hyper-heuristic approaches.

The automatic design of dispatching rules has been the subject of recent research attention because of the time-consuming process of manually designing such rules. An analysis of the representation of dispatching rules has been recently undertaken in [31]. The representation of dispatching rules essentially determines the search space and its complexity. As such, efficient representations highly impact upon the search process. Three different kinds of representations have been discussed in [31]: a linear combination of attributes, artificial neural networks, and a tree-based representation. Empirical analysis on the suitability of each representation in dynamic stochastic job-shop scenarios suggests that the tree representation, evolved with a hyper-heuristic genetic programming method, is the most efficient. Artificial

neural network representations operate well mostly on processes with a small computational budget, while linear representations operate competitively only on quite small computational budgets.

Dispatching rules for semiconductor manufacturing have been also automatically generated with genetic programming [58]. Grammatical evolution has been successfully applied as a hyper-heuristic methodology to address capacitated vehicle routing problems [87, 88]. Comparisons between adaptive and grammar-based hyper-heuristic approaches have been recently performed for various problem domains [86]. Moreover, a genetic programming-based hyper-heuristic has been considered in [101] to tackle order acceptance and scheduling problems in make-to-order manufacturing systems. This automatically generates dispatching rules that have stochastic behavior in order to improve the search procedure. The evolved rules exhibit high-performance gains compared to rules produced by either tailor-made heuristics or a simple version of the considered genetic programming method.

Several novel multi-objective genetic programming-based hyper-heuristics have recently been proposed that are able to generate scheduling policies in job-shop environments [96]. This paper proposes an automatic design approach to generate dispatching and due-date assignment rules for a dynamic version of the multi-objective job-shop scheduling problem. Having identified the complexity and the labor needed to manually design an effective scheduling policy, four multi-objective algorithms are developed to automate the process. All approaches are capable of handling multiple scheduling decisions in parallel. Thorough experiments and analyses demonstrate that the evolved Pareto sets represent effective scheduling policies that dominate policies from combinations of existing dispatching rules. These policies also have promising performance on unseen scenarios with various shop configurations. Generally all proposed methodologies exhibit not only effective but also very meaningful scheduling policies that help the decision-maker to understand their characteristics.

A two-phase search approach has been introduced in [95] to tackle the Workover Rigs Scheduling problem for maintenance services in onshore oil wells. The problem can be characterized as a parallel heterogeneous machine scheduling problem derived from the maintenance planning of heterogeneous wells. This problem is to find schedules for a small number of work-over rigs that minimize the production loss associated with the large number of wells waiting for service. The first phase of the proposed approach is a selection hyper-heuristic that searches for a promising initial solution to warm-start the second stage, which utilizes an exact branch, price, and cut approach. The considered hyper-heuristic uses learning mechanisms to learn from feedback during search and select the most effective low-level heuristic to apply in the next step. Having applied the abovementioned methodology to a variety of real-world problem instances from Petrobras, the Brazilian National Petroleum Corporation, the authors were able to identify classes of heuristics, which seem to be more efficient for solving the problem. In general, it was observed that the learning mechanisms were very effective and able to guide the search to promising

areas of the solution space. It is worth noting that the hyper-heuristic approach generated solutions that were very near to the optimal solutions found by the exact algorithm.

An interesting analysis has been performed in [94] that investigates the performance impact of heuristics while solving a problem with routing and rostering characteristics. The implemented hyper-heuristics act as analysis tools on the heuristic behavior. The aim is to analyze the behavior of the available heuristics and determine the requirements that a heuristic should consider to solve such problems. The following three different scheduling and routing problems have been considered: home care scheduling, routing and rostering of security guards, and maintenance personnel scheduling. The analysis revealed that some specific features of the problems under investigation significantly affect the performance of the heuristics. These features are the number of activities, the number of resources, and the planning horizon. The observed information of such analysis can be exploited and used in future search approaches to efficiently solve similar problems that share common characteristics. As such, hyper-heuristics can be employed as analysis tools for a particular problem class under investigation and may offer novel information on the characteristics of the problem class.

Nurse rostering is one of the applications where hyper-heuristics find wide applicability, with recent works including research in selection hyper-heuristics [14, 24]. A generic cooperative agent-based search framework has recently been introduced [89] that is able to incorporate the search strengths of various metaheuristics to efficiently solve nurse rostering problems. The framework promotes the asynchronous cooperation of effective agents using pattern matching and reinforcement learning techniques. This enables the exploration of different fairness objectives across several real-world rostering problem instances. A thorough experimental analysis demonstrates the efficiency of such frameworks to generate high-performance rostering solutions in minimal time.

Several hyper-heuristic methodologies draw upon swarm intelligence techniques, such as particle swarm and ant colony optimization. A particle swarm optimization-based hyper-heuristic approach has been introduced to address the resource-constrained project scheduling problem [72]. This operates on the heuristic space by representing its particles as ordered sequences of heuristics to apply for the given problem. Subsequently, a feasible solution is constructed by a serial scheduling construction mechanism, while the generated solution is improved by double justification local search. Several problem instances from the well-known PSPLIB library have been used to test and compare the hyper-heuristic algorithm with other algorithms. Computational results verify its efficiency and competitive performance.

The problem of intercell scheduling with single-processing machines and batch-processing machines has been tackled in [75] with an ant colony optimization hyper-heuristic search method. Generally, intercell transfers in cellular manufacturing systems are essential to substantially reduce production costs. The formulation of this scheduling problem considers the following three subproblems in parallel: an

assignment, a sequencing, and a batch subproblem. An ant colony hyper-heuristic has been adapted to search among different assignment heuristic rules for both parts and machines simultaneously. Subsequently, the discovered rules are applied to generate schedules. The efficiency of the proposed method is evaluated on randomly generated problem instances. The experimental results indicate that the method is significantly more efficient than CPLEX and genetic algorithms. Channel scheduling in multicell networks has also been addressed recently with hyper-heuristic methodologies [42].

A novel hyper-heuristic methodology has been proposed in [134] to solve task scheduling in cloud computing systems. Usually, rule-based scheduling algorithm is used for task scheduling in such systems. The method presented here is able to learn from the performance of several available scheduling algorithms and select the most suitable algorithm to apply. It incorporates a diversity and fitness detection strategies to efficiently balance the diversification and intensification behavior of the search procedure. This approach significantly outperforms several state-of-the-art scheduling algorithms on both simulation and real system scenarios. It produces solutions that have a significantly better makespan compared against other scheduling algorithms.

The problem of resource provisioning-based scheduling tasks in large-scale distributed environments, such as grids, has been addressed by exploring a hyper-heuristic scheduling method [16]. The main goal of this methodology is to efficiently schedule jobs on the available resources in order to simultaneously minimize execution time and increase security and reliability. The proposed hyper-heuristic algorithm is a particle swarm optimization-based algorithm that has been specifically developed to efficiently schedule jobs on available resources while addressing security requirements and constraints. A thorough experimental analysis reveals that the hyper-heuristic algorithm outperforms the state-of-the-art approaches, in terms of minimizing time, cost, and the makespan of the scheduling process.

Hyper-heuristics also have wide applicability in transportation scheduling. Motivated by a real railway-based disaster transportation relief scenario in the area of China, a selection hyper-heuristic search method [140] has been proposed to tackle emergency railway transportation problems. The evolutionary hyper-heuristic utilizes various search strategies as low-level heuristics and rewards each search strategy by assessing its ability to generate successful movements in the search space. The method has been compared with various state-of-the-art evolutionary algorithms on several problem instances, which have been created from disaster rescue operations data in China. Experimental results show high-performance gains across all instances against the considered evolutionary algorithms. The method has also been successfully applied on a real emergency during the 2013 Dingxi earthquake in China, producing a very satisfactory solution.

Another recent application of hyper-heuristic methodologies is the dial-a-ride transportation problem with time windows [135]. This is concerned with scheduling a set of available vehicles to pick up customers from an origin location

and deliver them to a destination. It considers a range of constraints. The proposed hyper-heuristic methodologies are efficient and result in competitive performance against state-of-the-art approaches from the literature. Moreover, cooperative hyper-heuristics have been successfully applied to bus driver scheduling problems [77].

Recent Educational Timetabling Applications of Hyper-heuristics

A recent overview and critical analysis of the literature on the hyper-heuristic methodologies applied to educational timetabling problems have been published in [102]. The article focuses on the hyper-heuristic research in three general educational timetabling problems, the university examination, the university course, and the school timetabling problems, as well as proposes several future research directions on the subject.

An estimation of distribution algorithm has recently been introduced [104] as a general hyper-heuristic approach to address exam timetabling problem instances. The objective was to develop a hyper-heuristic with increased generality over different problem instances. The hyper-heuristic operates on sequences of low-level exam-selection heuristics that construct a timetable, based only on non-domain-specific knowledge. The resulting algorithm is capable of guiding the search to very promising areas. It generates efficient sequences of timetabling heuristics. Experimental results on various real-world exam timetabling and graph coloring problem instances suggest that the hyper-heuristic is generic across all problem instances, with competitive performance against other hyper-heuristic approaches. In addition, the developed approach is capable of learning the most promising heuristic for the problem at hand.

A two-stage hybrid hyper-heuristic approach has recently been proposed to solve timetabling problems [35]. The two-stage hybrid approach utilizes a Kempe chain move heuristic and the time-slot swapping heuristic as search operators. Firstly, it generates and automatically analyzes random heuristic sequences. The analysis keeps the repeated heuristics of the best sequence and empties the remaining positions to be processed in the next stage. The second stage randomly assigns sequences if heuristics are in the empty positions in order to search for high-quality sequences of heuristics. The constructed ones are used for the given problem. The hybrid method is evaluated on two challenging sets of problems, the Toronto benchmark and the exam timetabling set of problems from the second International Timetabling Competition. The method exhibited competitive performance against the state-of-the-art methodologies for both problem sets.

New selection hyper-heuristic approaches that are based on simple stochastic search methods have exhibited outstanding performance in the recent high school timetabling competition [69]. Indeed, such approaches have provided exceptional results and new best solutions to very challenging problem instances. A hybrid hyper-heuristic which utilized sequences of low-level heuristics has also been applied recently in examination timetabling problems [15].

Games and Education

Hyper-heuristic methodologies have recently been proposed in the areas of games and education. A recent study focuses on solving the Jawbreaker puzzle [113]. The evolutionary process operates on the low-level heuristic space and produces sequences of low-level heuristics that can be applied to solve the given puzzle. The resulting methodology is able to efficiently solve instances of the puzzle with various sizes and levels of difficulty, including very difficult levels.

An educational game-based software tool has been introduced in [114] to teach the intrinsic operational characteristics of hyper-heuristics to engineering students. The tool is based on the Bubble Breaker puzzle, and it represents a strong synergy with the concept of selection hyper-heuristics. In addition, a hyper-heuristic method is used, under a context-aware ubiquitous learning environment, to maximize the learning efficacy of students [138]. The main goal of the search method is to locate quality learning paths for students by considering real-world context and constraints. The proposed approach is evaluated on real-world scenarios.

Other Recent Developments

The general character and the wide applicability of hyper-heuristics enable them to efficiently address a wide variety of applications. Here, the most recent developments in various scientific fields are presented, from linear algebra to aircraft structural design applications. A genetic programming-based hyper-heuristic is introduced in [71] to evolve the structure of the Sloan algorithm that aims to find a reduced envelope size of matrices. The envelope is the sum of the distances between each element of the matrix and its main diagonal. Its size has a major impact on the performance of large linear systems solvers. The best evolved algorithm (with the addition of a local search method) exhibits substantial performance gains and significantly outperforms state-of-the-art algorithms in the literature on classic problem sets.

The magic square problem has been efficiently tackled by a wide range of selection hyper-heuristic approaches [68] that utilize perturbative low-level heuristics.

A modified version of the well-known and widely used choice function has been successfully hybridized with a late acceptance strategy hyper-heuristic to tackle the multidimensional 0–1 knapsack problem [43]. In addition, a stochastic hyper-heuristic version of the choice function has been utilized to address the winner determination problem in [30, 73].

Another interesting application of hyper-heuristics is the real-world warehouse storage location assignment problem [136], in which a genetic programming methodology is employed to generate matching functions that guide the selection of subsets of items for a given problem. The methodology is able to locate high-quality solutions on the training problem cases. It is also able to perform well on unseen scenarios in the testing phase. This indicates that the evolved heuristics are reusable

and can be directly applied to similar problem scenarios. Furthermore, aircraft structural design optimization problems have recently been efficiently addressed by hyper-heuristic methodologies [12, 13].

The problem of distributing resources among multiple threads in a processor has been studied in [56]. A learning hyper-heuristic that predicts the best-performing heuristic between a set of available heuristics has successfully been applied to the problem. On average, its performance is comparable or significantly better than the state-of-the-art techniques in the field.

Case Study: A Selection Hyper-heuristic-Based Iterated Local Search

In this section, a simple selection hyper-heuristic framework is demonstrated to provide a tutorial-style introduction to hyper-heuristic development. Specifically, a hyper-heuristic framework is developed that is based on the well-known Iterated Local Search algorithm. The development process and the decisions that had to be made during implementation are briefly discussed (section "Iterated Local Search with Adaptive Heuristic Selection"). The framework implements an action selection model that operates on the Iterated Local Search algorithm to adaptively select among various search low-level perturbation heuristics (section "Action Selection Models"). It is based on the HyFlex platform which can be used to easily implement prototypes of hyper-heuristic search methods without having to be concerned about the implementation of problem domain low-level heuristics (which are available from the HyFlex site) (section "The HyFlex Framework").

The HyFlex Framework

HyFlex is a Java-based software framework that enables the easy development and testing of cross-domain hyper-heuristic search methodologies. The framework implements a common software interface to provide all the essential ingredients for easily developing a general-purpose search algorithm, without having to implement specific problem domain knowledge.

The framework implements six different combinatorial optimization problems, namely, maximum satisfiability (Max-SAT), one-dimensional bin packing, personnel scheduling, permutation flow-shop scheduling, the traveling salesman problem, and the vehicle routing with time windows problem. For each problem domain, there are 10 to 12 available problem instances, including both challenging benchmark and real-world industrial problem instances. The framework additionally exposes the user to a broad set of state-of-the-art low-level heuristic search operators with different search dynamics and characteristics. For all problem domains and all instances, appropriate evaluation functions have been implemented to easily evaluate generated solutions.

HyFlex currently has two main versions, 1.0 and 1.1. The first version includes the main functionality described above, while the second version adopts the concept of batch-mode hyper-heuristics, i.e., allowing the developed hyper-heuristic methodology to operate on a whole set of instances for each run, instead of using a single instance. A description of the APIs and documentation for both versions can be found in [20, 98] and in [1, 3, 4]. The HyFlex framework has been utilized in two cross-domain search challenges, CHeSC 2011 [1] and CHeSC 2014[3].

In this chapter, the first version of the framework has been adopted to develop the hyper-heuristic presented in the next section.

Iterated Local Search with Adaptive Heuristic Selection

This section discusses the motivation (and provides a detailed description) for designing a simple and general applicable hyper-heuristic search algorithm. However, the main purpose of this section is not to design a hyper-heuristic that has the best performance obtained in the literature so far. Instead, the goal of this section is to demonstrate how a simple search algorithm can be seen as a general hyper-heuristic approach. Moreover, it aims to show how the search algorithm can easily be applied to different problem domains without considering many unusual and made-to-measure design choices. The resulting approach can be easily fine-tuned on the application domains at hand and produce state-of-the-art performance gains [5, 6, 90].

The developed approach is based on a simple, general, and highly successful local search algorithm, the Iterated Local Search algorithm [29, 60, 82]. Iterated Local Search (ILS) is a single-point local search method that belongs to the wide category of stochastic local search methods [60]. The main idea behind its structure is to "iteratively create a sequence of potential solutions produced by a given heuristic search method, that leads to much better solution quality, than performing repeated random executions of it" [29, 60, 82]. In general, ILS includes four main stages in its structure. Firstly, ILS generates an initial solution for the problem at hand and then iteratively improves the current solution by applying a diversification (perturbation) and an intensification (local search) operation. Diversification is accomplished by applying a perturbation heuristic on the current solution, while intensification is achieved by employing a local search method on the perturbed solution. Having performed these two search operations, ILS determines whether the new solution is accepted to the next iteration, via an acceptance criterion. As such, in order to create an effective ILS variant, one should carefully determine the perturbation heuristic, the local search, and the acceptance criteria. A vast amount of available choices for various problem domains can be found in the literature [29, 60, 82].

Notice that the strength of perturbation heuristics varies among the available heuristics and greatly influences the behavior of the algorithm. For example, strong perturbations vastly increase the diversification of the search and might lead to a random restart-like behavior, i.e., there is a very low probability to find better

solutions. In contrast, perturbations with small strength might lead to very similar solutions that will not help the local search procedure find a better solution. Instead, the local search might revisit its previous solutions. Similarly, acceptance criteria can influence the balance between the intensification and diversification search abilities of the algorithm. For example, the acceptance of improving solutions will increase the intensification behavior of the algorithm, while accepting worsening moves will increase diversification.

Naturally, several ILS variants adapt their search strategies as well as their properties during the search procedure to react on the respective stage of the search. Representative reactive search principles can be found in [28, 29, 60]. Arguably this is the essence of a hyper-heuristic methodology [32–34, 36, 38, 102, 106, 107]. Motivated by these findings, this approach is adopted to develop a simple and general hyper-heuristic search framework and apply it to various problem domains. In particular, a simple framework of Iterated Local Search-based hyper-heuristic (HHILS) is implemented to exhibit the development process and the design choices that an interested researcher, or practitioner, might be faced with during the development phase of the hyper-heuristic search methodology.

In general, the HHILS framework follows the basic algorithmic structure of an Iterated Local Search methodology and enhances its reactive search ability with an action selection and a credit assignment module. The action selection module is devoted to efficiently predict and select the most suitable action to apply in the next step, while the credit assignment module is responsible for assigning a reward, or a credit value to the applied action, based on feedback from the search procedure. Arguably, the action selection module can be applied on any level or combination of levels within the search procedure, i.e., it can select among different perturbation heuristics, local search methodologies, and acceptance criteria, or any possible combination of them. This design choice is based on the level of adaptivity or reactivity that the researcher/practitioner is willing to incorporate in the search algorithm. In many cases, this decision will have a major impact on the performance of the resulting search methodology, since the adaptive procedure may rapidly change the dynamics of the search procedure, i.e., the diversification and intensification levels of the search. Here, a simple case is adopted in which the action selection module will operate only in the perturbation phase of the algorithm, i.e., it will be able to select from a set of available perturbation low-level heuristics.

The selected and applied action is scored based on the credit assignment module. The main role of the credit assignment module is to assign a reward, or credit value, to the applied action, based on feedback from the search procedure. The feedback can be seen as one or more measurements (or qualitative characterizations) of the applied action effects on the current state of the search process. The most common measurement for a search action is the quality improvement of its state (e.g., the difference in fitness value between the previous and the current state). However, various other qualitative different characterizations may be used, such as ranking successful movements [50], and evolvability [39, 124] of the search procedure. Such feedback is able to change the search characteristics of the algorithm and thus guide the search based upon different objectives. For example, the resulting

Algorithm 1: Pseudo code of the HHILS framework of ILS based hyper-heuristics.

1: Initialize the action selection, and the credit assignment module as well as create all data structures required by HHILS.
2: $s_{cur} \leftarrow$ GenerateInitialSolution() /* **Generate Initial Solution:** Initialize or construct a solution for the problem instance at hand. */
3: **while** termination criteria do not hold **do**
4: selected$_{llh} \leftarrow$ ActionSelection(str) /* **Action Selection:** Select a low-level heuristic with the str-th action selection model. */
5: $s_{tmp} \leftarrow$ ApplyAction(s_{cur}, selected$_{llh}$) /* **Perturbation:** Perturb the current solution (s_{cur}) with the selected low-level heuristic (selected$_{llh}$). */
6: $s_{tmp} \leftarrow$ ApplyLocalSearch(s_{tmp}) /* **Local Search:** Apply a local search procedure on the temporary solution s_{tmp}. */
7: $s_{cur} \leftarrow$ AcceptanceCriterion(s_{cur}, s_{tmp}) /* **Acceptance:** Accept which solution, between s_{cur}, and s_{tmp}, will survive to the next iteration based on an Acceptance criterion. */
8: CreditAssignment(selected$_{llh}$) /* **Credit Assignment:** Score the used action (selected$_{llh}$) based on feedback from the problem at hand. */
9: **end while**
10: **Return:** the best solution found so far s_{cur}.

algorithm will have more diversification characteristics or will avoid stagnating during the search procedure. Arguably, the quality of the feedback has a major impact on the performance and the success of the action selection model. As such, the literature includes various different attempts to investigate the impact of different feedback characterizations, like fitness improvement [46, 50, 53, 99], successful movements [47], rank based improvements [50], and evolvability [39, 124]. The selected feedback for the HHILS framework is based on the mean improvement of the applied perturbation heuristics and will be explained in detail below.

A family of different HHILS variants can be easily produced by adopting new action selection models and different credit assignment feedback functions. Algorithm 1 presents the general structure of the HHILS framework. Although some design choices have been made based on specific details related to the HyFlex framework (which was used for the implementation of HHILS), the algorithmic structure of the HHILS framework is relatively general and can be easily adapted to any other available framework.

HHILS firstly initializes all the necessary modules and data structures that are required by the algorithm to run properly (line 1 of Algorithm 1). This procedure includes the initialization of the credit assignment module with an empty or zero initial reward as well as the initialization of the action selection module that sets all actions to have equal chances to be selected. Next, HHILS initializes a solution (s_{cur}) for the problem instance at hand (line 2). The initial solution can be either generated randomly or constructed with a constructive heuristic. This depends on the problem domain and the available method to initialize, or construct, a feasible

solution. HyFlex provides such functionality for all problem domains which is convenient for easily constructing an initial solution independently of the problem domain.

Having found an initial solution, each iteration of HHILS performs the following five main steps. Let set $\mathcal{S}$ define the κ available perturbation low-level search heuristics $\mathcal{S} = \{llh_1, llh_2, \ldots, llh_\kappa\}$, for the problem domain in hand. In HyFlex, the developer has the option to select between greedy and non-greedy heuristics. Here, all available non-greedy heuristics for the current domain have been considered, which correspond mainly to *mutation*- and *ruin-and-recreate*-type-based heuristics. HHILS employs an action selection method (`ActionSelection(`str`)`, line 4) to predict and select the most appropriate perturbation low-level search heuristics to apply at the next step (selected$_{llh} \in \mathcal{S}$). The `ActionSelection` method might include various models, such as uniform selection and proportionate selection based on the reward values. The user is able to define which model to use with the str variable $str \in \mathcal{M}$, where $\mathcal{M}$ is the set of the available m action selection models, $\mathcal{M} = \{\mu_1, \mu_2, \ldots, \mu_m\}$. Section "Action Selection Models" briefly describes the models used in this chapter.

After selecting the selected$_{llh}$ perturbation low-level heuristic, HHILS proceeds with its application on the current solution s_{cur} to perturb its current position and create a new solution, s_{tmp} (line 5). The new position (s_{tmp}) is being immediately refined by the `ApplyLocalSearch(`s_{tmp}`)` procedure, which applies a greedy local search heuristic. In HyFlex, greedy local search heuristics belong in the *local search* category and have the property that they are not able to produce a worse solution than the given one. To avoid making a poor choice for the current problem instance, all of the available local search methods are applied in an iterative way [6]. More specifically, given a list $\mathcal{L}$ of the available λ local search heuristics, $\mathcal{L} = \{l_1, l_2, \ldots, l_\lambda\}$, at each iteration, a local search method is selected in a uniform random way and is applied to the current solution (s_{tmp}). If the application of the selected local search method does not lead to a better position (fitness improvement), it is excluded from the active list, until another local search heuristic finds an improved solution. If all the local search methods in the active list do not lead to an improved solution, then a local optimum has been reached. In this case, the iterative process will finish with the best solution found so far (s_{tmp}). The algorithmic scheme of the iterative local search procedure is illustrated in Algorithm 2.

Having applied the aforementioned transformations on s_{cur}, a new temporary solution s_{tmp} has been created. In the next step, HHILS employs an acceptance criterion `AcceptanceCriterion`(s_{cur}, s_{tmp}) (line 7 of Algorithm 1) procedure that is responsible to decide which of the two solutions will be accepted for the next iteration. In the literature, there are various choices available for the acceptance criterion [60], where the most common ones are to accept only improved solutions or to accept worsening solutions based on a probability distribution. These criteria will affect the diversification and intensification abilities of the current search method, for the given problem. Their impact depends on the landscape structure of the problem at hand. HHILS utilizes a Metropolis-based acceptance condition that accepts a worse solution based on a probability distribution. Intuitively, the greater

Algorithm 2: Pseudo code for the HHILS `ApplyLocalSearch(`s_{tmp}`)` method.

```
1: L_active ← L
2: while L_active is not empty do
3:    i ← uniformly random select a local search from L_active
4:    s_ls ← l_i(s_tmp) : Apply l_i local search on s_tmp
5:    if f(s_ls) < f(s_tmp) then
6:       s_tmp ← s_ls
7:    else
8:       L_active = L_active \ {i}
9:    end if
10: end while
11: Return: the best solution found so far s_tmp.
```

the fitness improvement, the greater the probability, or the greater the worsening fitness, the smaller the likelihood to accept the worse solution. A representative example of such an acceptance condition is the simulating annealing method [70].

Specifically, the probability density function of acceptance is defined as $p(f(s_{cur}),\ f(s_{tmp}), T, \mu_i) = e^{\frac{f(s_{cur})-f(s_{tmp})}{T\cdot\mu_i}}$, where $f(s_{cur})$ and $f(s_{tmp})$ are the objective value of the s_{cur} and s_{tmp} solutions, respectively, T is a constant temperature value with $T \in \mathbb{R}$, and μ_i is the mean improvement of the improving iterations [6]. The role of μ_i is to normalize the objective value difference by a quantity that does not depend on the problem at hand. Notice that the value of the temperature parameter is problem dependent and has to be fine-tuned for the considered problem in order to obtain optimal performance. Here, the temperature is fixed in all experiments to $T = 2$. A short and interesting study on the temperature behavior can be found in [6].

By this stage, HHILS has finished with the searching operations, and the applied low-level perturbation heuristic can be scored based on its search behavior. The final step of HHILS, before proceeding to the next iteration, is to employ the credit assignment module `CreditAssignment(`$selected_{llh}$`)` to score the selected perturbation heuristic, $selected_{llh}$ (line 8 of Algorithm 1). As such, HHILS defines the reward of an action as the improvement moves performed by the action, normalized by the total time it spends on them. Such a reward is a representative score for the improvement rate of each action over the time spend for each improvement. Specifically, let $\mathcal{A} = \{a_1, a_2, \ldots, a_\alpha\}$ be the available set of α actions (note that in HHILS $\mathcal{A} = \mathcal{S}$) and t_{α_i} be the time consumed by action α_i on searching the solution space (initially $t_{\alpha_i} = 0, \forall \alpha_i \in \mathcal{A}$). Then at each iteration, the time spent for the applied action, i, is calculated as $t_i = t_i + t_i^{spent}$, where t_i^{spent} is the time of action i spent on the current iteration. The reward value (r_i) of the applied action (i) can be calculated according to $r_i = \frac{1+improvement_i}{t_i}$, where $improvement_i = f(s_{cur}) - f(s_{tmp})$. In general, various different reward approaches can be used at this point, such as the instantaneous latest reward or the

average or a ranked-based reward [50]. However, some of them tend to be unstable and noisy estimations of credit due to the stochastic nature of the search process. To alleviate this drawback, the empirical quality of an action is estimated by utilizing the average value of a sliding window of its latest w rewards. More specifically, let W_i be a set of the latest w improvement rewards (r_i) achieved by the action a_i during the time step t of the hyper-heuristic. The final credit assignment value $r_{a_i}(t)$ is the average reward of the sliding window W_{a_i}, which can be determined by

$$r_{a_i}(t) = \frac{\sum_{j=1}^{|W_{a_i}|} W_{a_i}(j)}{|W_{a_i}|} \tag{1}$$

where $|W_{a_i}|$ indicates the cardinality of the set W_{a_i}.

Finally, the algorithm will iterate until one or more termination criteria hold. Here, the maximum allowed CPU wall clock time is used as termination criterion, $t_{allowed}$, which the algorithm may consume to solve the problem under investigation. It is worth noting that the HHILS framework does not consider whether the algorithm gets trapped in a particular position of the solution space for a long period of time. This is a very common situation in which either the algorithm is in a local optimum or the search operators are not able to find a better solution for a long period of time. In such cases, a restarting procedure is essential to help the searching capabilities of the algorithm to diversify the current solution. However, a restarting procedure requires specific design choices that for the sake of simplicity are overlooked in this tutorial demonstration.

Action Selection Models

As discussed previously, HHILS is a general framework that has the ability to adopt any action selection model. The literature includes various action selection models from different scientific fields such as statistics, filter theory, artificial intelligence, and machine learning. Each model has different advantages and disadvantages that depend on the properties of the underlying reward distribution that it is trying to predict. Representative examples of such models that have been used for this purpose include probability matching [130, 131], adaptive pursuit [130, 131], statistical-based models like the multinomial distribution with history forgetting [46, 47], evolutionary meta-learning methods [44, 45, 79, 123], and reinforcement learning approaches [50, 51, 127].

In this chapter, five different models have been utilized: the uniform selection model, the proportional selection model, the probability matching [130, 131], the adaptive pursuit [130, 131], and the upper confidence-bound multi-armed bandit methodology [50, 127].

The first two models are very simple and act as baseline models for the HHILS framework. The uniform selection model simply selects between the available actions randomly by following a uniform distribution. In this case, the scores

assigned to each action do not have any impact on the selection process. This is the baseline methodology that will provide insights into the usefulness and impact of applying an action selection model on HHILS. Notice that several studies have identified the usefulness of random variation in either action or parameter spaces [63, 65, 66]. The second model performs proportional selection among the available actions based on their assigned scores from the credit assignment module. Intuitively, this model provides proportional chances to each action based on their reward scores during the optimization phase, i.e., the most successful will have more chances to be selected again in the next step. The proportional selection is performed by applying a simple roulette wheel selection [23].

Probability matching (PM) [50, 130, 131] is a simple probabilistic model that updates the selection probability of each action proportionally to its empirical quality with respect to the other actions. Specifically, given a set $\mathcal{A} = \{a_1, a_2, \ldots, a_\alpha\}$ of the available α actions (here $\mathcal{A} = \mathcal{S}$), let $P(t) = \{p_{a_1}(t), p_{a_2}(t), \ldots, p_{a_\alpha}(t)\}$ be the selection probability vector of all actions, where initially $p_{a_i} = 1/\alpha, \forall a_i \in \mathcal{A}$. After the application of the perturbation, the local search, and the acceptance criterion stages, a reward $(r_{a_i}(t))$ is measured from the credit assignment module. Notice that in the general case, the reactive procedure operates on a nonstationary environment where the estimation of the empirical quality of an action might be more accurate and reliable if the more recent rewards influence the empirical quality more than the past ones. Therefore, in such cases, it is a common practice by various action selection models to estimate the empirical quality $q_{a_i}(t)$ of each action (a_i) in accordance with the following relaxation mechanism:

$$q_{a_i}(t+1) = q_{a_i}(t) + \gamma(r_{a_i}(t) - q_{a_i}(t)) \tag{2}$$

where $\gamma \in (0, 1]$ is the adaptation rate (here γ is fixed to 0.1) [50, 130, 131]. Having calculated the empirical quality of each action (a_i), PM adapts its selection probability (p_{a_i}) using the following rule:

$$p_{a_i}(t+1) = p_{\min} + (1 - \alpha \cdot p_{\min}) \frac{q_{a_i}(t+1)}{\sum_{i=1}^{\alpha} q_{a_i}(t+1)} \tag{3}$$

where $p_{\min} \in [0, 1]$ is the minimum probability value of each action. $p_{\min}$ is responsible for ensuring that each action will always have a small probability (here $p_{\min} = 0.05$) [130, 131]. Having estimated the probabilities of each action, PM utilizes a stochastic proportional selection mechanism, based on these probabilities, to select which action will be applied in the next step.

Notice that PM allows the inefficient actions to have at least $p_{\min}$ selection probability. Thus, the best action will be selected with probability $p_{\max} = (1 - (\alpha - 1) \cdot p_{\min})$ However, this hinders the efficiency and performance of PM, since all inefficient actions keep being selected. Therefore, the adaptive pursuit (AP) strategy [130, 131] addressed this drawback by embracing a winner-takes-all strategy. In detail, AP, instead of updating proportionally the action probabilities,

increases the probability of the corresponding best action ($a_{i^\star}$), while in parallel, it decreases the probabilities of the other actions according to the following equations:

$$a_{i^\star} = \arg \max_{i=1,2,\ldots,\alpha} \{q_{a_i}(t+1)\} \tag{4}$$

$$p_{a_i}(t+1) = \begin{cases} p_{a_i}(t) + \beta(p_{\max} - p_{a_i}(t)), \text{ if } a_i = a_{i^\star} \\ p_{a_i}(t) + \beta(p_{\min} - p_{a_i}(t)), \text{ otherwise} \end{cases} \tag{5}$$

where $\beta \in [0, 1]$ is a learning rate that manipulates the greediness of the winner-takes-all strategy (here $\beta = 0.8$). Intuitively, AP first finds the best action of the current step and then updates the probabilities by being in favor of the best action.

The last action selection model used in this chapter formulates the action selection model as a multi-armed bandit (MAB) problem and utilizes the upper confidence-bound (UCB) algorithm to solve it. The MAB problem considers κ one-arm slot machines and a player that has to decide which arm to pull, how many times to pull it, and in which order to choose between the available arms, having in mind that each machine provides a random reward. Intuitively, the MAB problem faces the well-known trade-off of *exploration versus exploitation*, where each time the player has to decide either to "exploit" its current knowledge, i.e., to pull the arm that gives the highest expected payoff, or to "explore" the available choices, i.e., to acquire more information about the expected payoff of the other arms.

More specifically, let us have κ arms (one-arm slot machines), where each arm (i-th) has a fixed unknown reward probability $p_i \in [0, 1], \forall i \in \{1, 2, \ldots, \kappa\}$. Assume that the arms are independent and that the associated rewards with each arm are independently and identically distributed. At each time step (t), the player selects an arm (j) and obtains a reward $r_t = 1$ with probability p_j or $r = 0$ otherwise. At any stage of the procedure, the performance of the MAB algorithm is estimated either by calculating the cumulative reward at the current stage or by employing a regret measure, i.e., the difference in performance between the current and the best arm. As such, the main objective of the selection algorithm at any stage is to either maximize the cumulative reward of the procedure or minimize its regret. In such a context, an "optimal" algorithm will have the best possible performance if at each time step it is able to select the best (unknown) arm, i.e., the arm with the maximum probability of obtaining a reward.

The upper confidence-bound (UCB) algorithm is implemented to solve the MAB problem [22, 50, 51]. It is worth mentioning that UCB achieves an optimal regret rate on the aforementioned formulation. Specifically, let $q_{a_i}(t)$ be an estimation of the empirical quality of action a_i on time step t, and let c_i be a confidence interval associated with the empirical quality of action a_i that depends on the amount of times $n_i(t)$ action i has been tried until the current step t. Then, at each time step, UCB deterministically selects the next action ($selected_{llh}$) with the optimal upper bound of the confidence interval c_i. The selection process can be calculated according to

$$selected_{llh} = \arg \max_{i=1,2,\ldots,\alpha} \left(q_{a_i}(t) + \mathbf{C} \cdot \sqrt{\frac{2 \log \sum_k n_k(t)}{n_i(t)}} \right) \quad (6)$$

$$n_i(t+1) = n_i(t) + 1 \quad (7)$$

$$q_{a_i}(t+1) = \frac{(n_i(t) - 1) \cdot q_{a_i}(t) + r_{a_i}(t)}{n_i(t)} \quad (8)$$

where $\mathbf{C}$ is a user-defined scaling factor that balance the trade-off between exploitation and exploration ability of the model (here $\mathbf{C} = 0.8$). Intuitively, the first term of Equation 6 favors the action with the best empirical quality (exploitation), while the second term benefits the trial of the other actions (exploration). UCB thus selects mostly the action that potentially provides the best reward, while giving the opportunity for infrequently tried actions to be applied regularly.

An analysis of the behavior and adaptability of the applied models should provide useful insights into the behavior of each model and its impact on the respective algorithm that incorporates it. However, this issue is beyond the scope of this tutorial chapter.

Computational Results

This section demonstrates a thorough analysis of the performance of the developed hyper-heuristics by evaluating them on a wide range of problem domains with different characteristics. The experimental protocol used in the experiments is firstly defined, and then two qualitative different analyses are presented. The first study provides comprehensive experimental results and statistical analyses of the developed hyper-heuristics on each of the considered problem domains separately, while the second study follows the experimental protocol used in the CHeSC 2011 [1] competition and compares the developed hyper-heuristics with the state-of-the-art approaches of the competition.

Experimental Protocol

Five different hyper-heuristic approaches have been developed that are based on the HHILS framework described previously in section "Iterated Local Search with Adaptive Heuristic Selection". The difference between the hyper-heuristics mostly lies in the action selection mechanism used to select among the available low-level perturbation heuristics. As such, comparisons between the following five hyper-heuristics are conducted:

- HHILS: The simplest HHILS variant that selects the available perturbation low-level heuristics randomly following a uniform distribution.

- HHILS-SA: An HHILS variant that proportionally selects which perturbation low-level heuristic to apply, based on their reward value.
- HHILS-PM: An HHILS variant that utilizes the probability matching selection method to select the perturbation low-level heuristics.
- HHILS-AP: An HHILS variant that utilizes the adaptive pursuit selection method to select the perturbation low-level heuristics.
- HHILS-MAB: An HHILS variant that utilizes the upper confidence-bound multi-armed bandit methodology to select the perturbation low-level heuristics.

All HHILS variants have been implemented in the Java programming language, based on the functionality provided by the HyFlex framework. Although their implementation details are specific for the HyFlex framework, their algorithmic design is general and can be easily adapted and developed in any other framework. To motivate the usage and further development of the HHILS variants, their source code is available at https://github.com/mikeagn/hhils.

To maintain a fair and reliable comparison, the same parameter configuration is used for the common parameters of all algorithms. In addition, for the different action selection models that have been incorporated in the HHILS variants, the default parameter settings are used as proposed in the literature. Specifically, all HHILS variants employ a temperature value of $T = 2.0$, while the action selection models employ the default values used in the literature as described in the previous section. Notice that it has not been performed any fine-tuning process to obtain high-quality parameter configurations. However, such parameter configuration values can be found by performing either manual or automatic tuning and sensitivity analyses. This tuning process might lead to superior performance gains; nevertheless, this is out of the scope of the current tutorial demonstration. Prominent examples of successful off-line automatic tuning tools among others are the `irace` [81], the `SPOT` [26], and the `SMAC` [61] tools.

Six different problem domains have been implemented within the HyFlex framework, namely, the Max-SAT (SAT), the bin packing (BP), the personnel scheduling (PS), the Flow Shop (FS), the traveling salesman problem (TSP), and the vehicle routing with time windows problem (VRP). For each problem domain, the HyFlex framework provides 10–12 problem instances. A problem instance corresponds to either a real-world realization or a well-known and challenging benchmark case of the considered problem domain. More information about the available instances can be found online in [2]. To evaluate the performance of an algorithm on a given problem instance, the best objective value achieved by the algorithm within a prespecified available time budget is used. Problem domains have been modeled as minimization problems. As such, the lower objective value indicates a better performance. The first part of the experimental analyses provides extensive experimental results for all available instances per problem domain.

For fair comparisons, a benchmarking program is provided on the CHeSC 2011 website [1]. This determines the allowed time limit ($t_{allowed}$) of each run under the specific hardware architecture $t_{allowed}$ which corresponds to 10 min of CPU time on the machine used during the competition. All experiments in this study have been

conducted on an AMD Opteron CPU of 2.2 GHz machine with 32 GB of RAM running GNU/Linux operating system. For this machine, the allowed time limit is 624 s ($t_{allowed} = 624$ s).

Finally, the second part of the analyses strictly follows the rules of the CHeSC 2011 competition in order to provide an easy and fair comparison between all hyper-heuristic search methodologies. A thorough description of the experimental setup used in section "Comparison to HyFlex State-of-the-Art Hyper-heuristics" can be found in [98].

Experimental Results

In this section, the HHILS variants are evaluated on the six available problem domains provided by the HyFlex framework. Specifically, the performances of the five developed HHILS variants are compared on all available instances per problem domain. Firstly the performance gains obtained by each algorithm are described separately, for each problem domain. Then, their overall performance is summarized, and the observed behavior is verified by statistical analysis.

Tables 1, 2, 3, 4, 5, and 6 show statistics on the performance of each algorithm per problem domain, in terms of the best objective value obtained over 31 independent runs. For each algorithm and each problem instance (I), the following statistics are provided: the best (min), the median (m), the mean objective value (μ), and the standard deviation (σ) from the mean objective value obtained by the algorithm at hand. The cases where either the best, the median, or the mean objective value is the smallest (i.e., best performance) across all algorithms for a problem instance in hand are highlighted with **boldface**. In general, it can be observed that, although each algorithm behaves differently across the various problem domains, some algorithms exhibit robust performance across the majority of the domains, such as the HHILS-AP and the HHILS-SA.

More specifically, it can be easily observed that HHILS-AP shows superior performance against the other algorithms in three different domains, BP, FS, and TSP. As presented in Tables 1, 2, and 5, for at least 60% of the problem instances, HHILS-AP shows superior mean and median performance, while it is able to find the best objective value against the other algorithms in 6, 8, and 7 instances for the BP, FS, and TSP, respectively. Regarding the VRP problem domain, although there are three instances where the observed median and mean value are the smallest, in the majority of the remaining cases, it is not able to show competitive performance against the first-ranking algorithms in the domain (HHILS-SA, HHILS-PM).

The next most promising hyper-heuristic is HHILS-SA, which is able to achieve best performance, in terms of both mean and median objective values, in three problem domains FS, PS, and VRP. HHILS-SA also demonstrates the best performance in several problem instances across the majority of problem domains, i.e., 11, 6, 4, 4, and 2 in SAT, VRP, FS, PS, and TSP, respectively. The remaining HHILS variants perform similarly for the majority of the considered domains. HHILS and HHILS-MAB do not exhibit observable performance differences for the majority

Table 1 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the bin packing domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	0.007	0.017	0.017	0.005	0.007	**0.007**	**0.007**	0.000	**0.006**	0.007	0.008	0.001	0.007	0.011	0.010	0.002	0.007	0.007	0.008	0.002
i1	0.012	0.017	0.017	0.004	**0.007**	**0.008**	**0.008**	0.000	0.007	0.008	0.008	0.001	0.008	0.008	0.009	0.002	0.007	0.008	0.008	0.001
i2	0.022	0.024	0.024	0.001	**0.021**	**0.022**	**0.022**	0.000	0.022	0.024	0.024	0.002	0.022	0.023	0.023	0.001	0.023	0.023	0.023	0.000
i3	0.023	0.026	0.025	0.001	**0.021**	**0.022**	**0.022**	0.001	0.024	0.026	0.026	0.002	0.023	0.024	0.024	0.000	0.023	0.024	0.024	0.000
i4	0.007	0.007	0.007	0.001	0.005	0.005	0.005	0.000	**0.000**	**0.005**	**0.005**	0.001	0.005	0.007	0.006	0.001	0.005	0.006	0.006	0.001
i5	0.004	0.009	0.008	0.002	0.004	0.004	0.004	0.000	**0.003**	**0.003**	**0.003**	0.000	0.004	0.004	0.005	0.002	0.004	0.004	0.004	0.001
i6	0.084	0.089	0.088	0.003	**0.011**	0.037	0.045	0.032	0.025	0.032	0.033	0.003	0.022	0.032	0.032	0.008	0.011	**0.016**	**0.018**	0.005
i7	0.107	0.112	0.112	0.003	**0.028**	0.088	0.068	0.029	0.061	0.076	0.074	0.007	0.043	0.069	0.070	0.012	0.033	**0.051**	**0.051**	0.013
i8	0.049	0.052	0.052	0.001	**0.039**	**0.047**	**0.046**	0.005	0.057	0.066	0.065	0.003	0.045	0.049	0.049	0.002	0.046	0.049	0.049	0.001
i9	0.009	0.014	0.013	0.002	0.008	0.010	0.010	0.001	0.016	0.019	0.019	0.001	**0.007**	0.009	0.010	0.001	0.008	**0.009**	**0.009**	0.001
i10	0.110	0.111	0.111	0.001	0.109	0.109	0.109	0.000	**0.108**	**0.108**	**0.108**	0.000	0.110	0.110	0.110	0.000	0.109	0.110	0.110	0.000
i11	0.029	0.031	0.031	0.002	0.017	0.020	0.020	0.002	0.033	0.037	0.037	0.002	**0.012**	**0.017**	**0.016**	0.002	0.018	0.023	0.023	0.002

Table 2 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the flow-shop domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	6328.0	6358.0	6356.1	12.8	**6308.0**	**6333.0**	**6331.0**	7.4	6327.0	6346.0	6346.5	9.4	6319.0	6343.0	6340.6	7.5	6324.0	6345.0	6344.2	6.1
i1	6282.0	6304.0	6302.4	8.4	6263.0	6288.0	**6285.6**	9.4	**6260.0**	6297.0	6295.7	9.0	6270.0	6294.0	6293.6	9.6	6270.0	**6287.0**	6288.0	6.4
i2	6370.0	6386.0	6384.9	7.3	6352.0	**6366.0**	**6363.8**	5.9	6365.0	6382.0	6382.5	7.8	6352.0	6372.0	6370.0	7.2	**6349.0**	6368.0	6367.3	7.5
i3	6361.0	6369.0	6369.2	3.3	**6323.0**	**6352.0**	**6352.4**	10.9	6350.0	6369.0	6367.2	6.0	6337.0	6367.0	6365.1	6.3	6330.0	6359.0	6357.7	11.0
i4	6424.0	6451.0	6447.1	10.7	**6402.0**	**6417.0**	**6416.3**	6.9	6409.0	6444.0	6442.0	11.9	6408.0	6426.0	6426.7	10.7	6407.0	6425.0	6424.2	7.2
i5	10501.0	10525.0	10525.6	9.9	**10495.0**	**10501.0**	**10500.9**	3.6	10517.0	10531.0	10529.5	5.9	10501.0	10516.0	10516.5	8.0	10497.0	10509.0	10508.2	6.0
i6	10948.0	10959.0	10960.4	4.4	**10923.0**	**10944.0**	**10943.6**	10.2	10938.0	10960.0	10959.7	6.6	10934.0	10957.0	10955.4	6.8	**10923.0**	10956.0	10950.7	10.2
i7	26290.0	26402.0	26394.8	36.3	**26249.0**	**26293.0**	**26293.0**	20.7	26339.0	26407.0	26408.3	26.4	26291.0	26343.0	26343.5	23.9	26266.0	26324.0	26322.8	27.7
i8	26859.0	26886.0	26895.1	28.1	26765.0	26831.0	26823.6	24.7	26852.0	26896.0	26895.5	21.2	26806.0	26866.0	26865.3	28.8	**26761.0**	**26815.0**	**26812.2**	20.4
i9	26663.0	26731.0	26725.3	27.9	**26580.0**	**26645.0**	**26641.6**	28.3	26667.0	26712.0	26717.9	25.1	26642.0	26704.0	26703.0	23.8	26616.0	26682.0	26679.5	23.2
i10	11420.0	11473.0	11471.4	20.5	**11398.0**	**11432.0**	**11430.2**	14.7	11442.0	11464.0	11466.9	11.8	11431.0	11458.0	11457.5	12.0	11404.0	11445.0	11444.4	12.4
i11	26629.0	26690.0	26694.2	27.3	26567.0	26640.0	26632.6	31.9	26611.0	26700.0	26694.2	27.2	26621.0	26678.0	26675.3	26.4	**26548.0**	**26638.0**	**26631.0**	28.1

Table 3 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the personnel scheduling domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	**3299.0**	**3317.0**	3319.6	12.7	**3299.0**	3318.0	**3318.5**	11.7	3312.0	3335.0	3335.9	14.9	3301.0	3325.0	3325.3	13.7	3301.0	3322.0	3321.5	11.8
i1	**1948.0**	2172.0	2155.8	101.3	2022.0	2175.0	2184.2	85.9	1950.0	2210.0	2208.5	88.3	1967.0	2137.0	2136.9	89.3	1955.0	**2103.0**	**2118.0**	98.1
i2	325.0	365.0	364.5	18.3	**305.0**	360.0	359.7	19.9	335.0	380.0	377.3	21.1	335.0	360.0	360.5	13.5	325.0	**355.0**	**354.8**	16.1
i3	13.0	18.0	18.3	2.1	13.0	19.0	19.0	2.4	17.0	21.0	21.4	2.6	**12.0**	**17.0**	**18.1**	3.5	**12.0**	18.0	**18.1**	3.0
i4	**14.0**	20.0	20.0	2.4	15.0	**19.0**	**18.9**	2.2	20.0	24.0	24.4	3.1	**14.0**	20.0	20.2	3.1	15.0	20.0	20.1	2.4
i5	16.0	21.0	21.3	3.0	**12.0**	**18.0**	**18.5**	2.6	17.0	25.0	24.8	3.2	14.0	20.0	20.6	2.8	16.0	21.0	22.0	3.1
i6	1108.0	1132.0	1148.4	40.2	1112.0	1132.0	1149.8	37.9	1108.0	1219.0	1213.6	66.2	**1103.0**	**1124.0**	**1132.2**	29.8	1107.0	1126.0	1141.1	38.6
i7	2170.0	2219.0	2230.1	38.1	2183.0	2262.0	2251.6	48.4	2195.0	2270.0	2262.6	51.7	2178.0	2225.0	2231.6	47.0	**2166.0**	**2211.0**	**2220.0**	34.0
i8	3159.0	3263.0	3263.3	65.5	3157.0	3256.0	3276.3	81.8	3158.0	3292.0	3308.6	76.8	3156.0	3238.0	3227.4	51.7	**3138.0**	**3181.0**	**3207.0**	59.2
i9	9420.0	9902.0	10006.1	344.6	9429.0	9910.0	10079.2	516.9	**9407.0**	**9678.0**	**9671.2**	177.8	9672.0	9904.0	10090.1	444.0	9422.0	9958.0	9999.1	394.9
i10	**1405.0**	1630.0	1613.7	100.3	1445.0	1580.0	1603.7	84.3	1450.0	1623.0	1639.3	97.1	1462.0	1610.0	1622.2	94.4	1435.0	**1565.0**	**1574.3**	75.0
i11	305.0	**335.0**	335.8	12.6	320.0	**335.0**	337.3	12.0	320.0	350.0	348.3	15.5	315.0	**335.0**	334.9	10.5	**300.0**	**335.0**	**334.4**	15.0

Table 4 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the SAT domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	2.00	**2.00**	2.00	0.00	2.00	**2.00**	2.00	0.00	2.00	**2.00**	2.00	0.00	2.00	**2.00**	2.00	0.00	**1.00**	**2.00**	**1.97**	0.18
i1	19.00	21.00	20.97	1.70	18.00	20.00	20.29	1.47	**17.00**	**19.00**	**18.97**	0.84	18.00	21.00	21.23	1.80	18.00	20.00	19.97	1.14
i2	**14.00**	16.00	16.52	1.61	**14.00**	16.00	16.10	1.04	**14.00**	16.00	15.68	1.11	**14.00**	16.00	16.03	1.28	**14.00**	**15.00**	**15.48**	1.03
i3	**0.00**	2.00	1.90	0.79	**0.00**	2.00	1.58	0.56	**0.00**	**1.00**	1.55	0.81	**0.00**	2.00	1.61	0.88	**0.00**	**1.00**	**1.52**	0.77
i4	1.00	2.00	1.71	0.78	1.00	**1.00**	1.32	0.48	**0.00**	**1.00**	1.55	0.81	**0.00**	**1.00**	1.42	0.56	**0.00**	**1.00**	**1.03**	0.55
i5	**1.00**	3.00	2.84	0.64	**1.00**	3.00	2.23	0.88	**1.00**	3.00	2.29	0.86	**1.00**	3.00	2.65	1.60	**1.00**	**2.00**	**1.97**	0.87
i6	**5.00**	**5.00**	**5.00**	0.00	**5.00**	**5.00**	5.19	0.48	**5.00**	6.00	5.81	0.87	**5.00**	**5.00**	5.03	0.18	**5.00**	**5.00**	5.45	0.62
i7	**5.00**	**5.00**	**5.06**	0.25	**5.00**	6.00	5.52	0.51	**5.00**	**5.00**	5.55	0.62	**5.00**	**5.00**	5.32	0.48	**5.00**	6.00	5.61	0.56
i8	**5.00**	**7.00**	**6.16**	1.19	**5.00**	8.00	7.13	1.31	**5.00**	8.00	7.45	1.21	**5.00**	8.00	7.29	1.22	**5.00**	8.00	7.26	1.37
i9	**209.00**	211.00	210.13	1.06	**209.00**	211.00	210.10	1.01	**209.00**	**209.00**	**209.26**	0.68	**209.00**	**209.00**	209.77	0.99	**209.00**	**209.00**	209.90	1.01
i10	**1.00**	2.00	2.16	1.19	**1.00**	**1.00**	1.42	0.76	**1.00**	**1.00**	1.32	0.83	**1.00**	**1.00**	1.16	0.37	**1.00**	**1.00**	**1.03**	0.18
i11	**7.00**	**7.00**	**7.39**	0.50	**7.00**	**7.00**	7.42	0.56	**7.00**	8.00	7.65	0.66	**7.00**	8.00	7.68	0.70	**7.00**	8.00	7.94	0.73

Table 5 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the TSP domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	48273.9	48392.2	48398.9	58.6	**48194.9**	**48214.9**	**48221.7**	28.7	48194.9	48225.7	48224.4	19.6	48209.7	48298.5	48291.2	30.6	48209.7	48270.2	48261.7	27.3
i1	108616.5	109392.8	109618.3	735.8	**107225.3**	109349.4	108800.4	962.6	108217.7	109167.8	109126.8	498.6	107442.3	108459.1	**108616.0**	817.0	107259.6	**108446.1**	108740.2	948.8
i2	6894.6	6923.7	6921.9	12.8	**6823.4**	**6844.5**	**6843.9**	10.7	6888.0	6915.9	6914.9	11.4	6863.2	6897.2	6896.6	14.2	6862.2	6895.6	6893.7	11.8
i3	42546.5	42709.8	42716.7	85.5	**42173.0**	**42279.8**	**42274.5**	60.8	42348.6	42614.6	42608.2	81.4	42467.0	42589.7	42576.6	63.1	42375.1	42512.2	42500.6	62.1
i4	9009.4	9048.3	9047.0	15.4	**8922.5**	**8943.0**	**8943.8**	11.2	8958.2	8973.4	8975.9	11.5	8985.8	9016.0	9012.9	14.4	8963.5	9004.1	9004.2	16.4
i5	58192.9	58612.5	58592.8	156.2	**57294.8**	**57560.0**	**57551.4**	141.8	58207.2	58766.1	58743.6	234.4	57975.3	58184.4	58197.8	137.1	57779.9	57977.5	58006.1	153.2
i6	55088.6	58971.0	58648.4	1749.7	53077.3	54605.6	54902.8	1189.3	53779.3	**54030.7**	**54031.2**	111.1	53096.6	56898.8	56873.8	1882.7	**52857.0**	54295.0	54257.8	781.0
i7	68711.3	69749.9	69723.7	474.9	**65789.9**	**66301.9**	**66296.2**	233.7	68339.4	68560.9	68593.5	158.3	67049.2	67786.3	67892.4	443.9	66088.5	67098.6	67047.9	399.6
i8	**21090984.4**	21380811.8	21396672.8	187926.6	21145557.5	21375242.0	21394858.1	165005.6	21164043.9	**21284114.6**	**21276739.0**	53894.4	21145557.5	21374787.3	21400271.2	160545.3	21210701.9	21364765.7	21377212.8	144263.6
i9	672053.3	676463.7	676895.2	2362.8	673105.2	676356.6	677073.4	2506.1	673688.4	675793.9	675873.1	1241.5	673583.9	676599.4	677243.7	2427.2	**671431.6**	**675118.2**	**675141.7**	1863.9

Table 6 Statistics on the best objective value achieved by the developed HHILS variants on all problem instances of the VRP domain

	HHILS				HHILS-AP				HHILS-MAB				HHILS-PM				HHILS-SA			
I.	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ	min	m	μ	σ
i0	5130.6	5165.7	5164.1	13.5	5115.0	5156.9	5155.6	13.9	**4250.7**	5156.7	**5124.0**	162.7	5114.0	**5156.5**	5158.0	14.4	5130.6	5165.4	5163.4	10.6
i1	20655.0	20657.4	20657.8	1.7	20652.2	**20654.0**	**20654.5**	3.1	20652.6	20656.5	20721.0	247.1	20652.5	20655.2	20655.0	0.6	**20652.2**	20654.7	20654.5	0.6
i2	12290.7	12323.8	12451.4	349.5	**12274.4**	**12294.6**	**12293.2**	15.6	12305.4	13343.0	13242.6	404.9	12282.3	12303.7	12305.0	16.0	12281.3	12307.1	12306.7	12.8
i3	5348.1	5373.7	**5374.6**	10.0	5351.4	6250.1	5939.5	435.2	**5321.4**	6245.4	6158.2	268.2	5326.7	**5366.1**	5453.2	277.0	5341.9	5367.3	5481.2	310.8
i4	13281.3	13293.6	13356.8	244.9	**13272.0**	**13286.9**	13695.2	493.3	13291.5	14280.2	14217.5	355.9	13277.3	13288.1	**13352.1**	245.8	13275.1	13287.5	13383.9	294.6
i5	197926.3	204256.8	204910.1	3616.1	142488.6	145038.2	**144824.7**	1402.2	210603.3	219067.4	219536.2	3525.1	143961.3	148171.3	148519.8	3054.0	**142479.0**	**144184.5**	144854.3	1676.7
i6	64490.5	67479.9	67417.3	1335.8	61316.4	64272.3	64115.7	1719.2	71101.8	72849.9	73027.1	1233.4	60121.0	62336.5	62360.7	988.9	**59730.3**	**61651.6**	**61552.1**	951.9
i7	173238.4	177164.0	176870.5	1251.6	157870.2	158386.5	158516.4	579.8	184077.7	188024.2	187791.9	1622.3	157881.5	158794.6	158839.0	545.1	**157770.8**	**158339.2**	**158343.0**	360.4
i8	165077.3	171211.4	171229.5	2639.7	143085.2	146652.3	147012.3	1496.2	188337.3	196181.7	196299.1	3476.2	143584.5	148048.2	148395.3	2280.6	**142928.5**	**144561.6**	**144347.1**	1222.1
i9	159486.8	162101.1	162205.1	1293.7	144086.8	145586.4	145595.4	668.4	165525.7	168651.6	168616.4	1493.4	144244.0	146098.4	146209.1	916.1	**143565.7**	**144827.5**	**144974.6**	862.5

of the studied cases, while HHILS-PM shows slight improvement gains against them for the BP, FS, and TSP domains (in terms of mean and median objective values). However, in general, the performance of HHILS-PM is more robust when compared with both HHILS and HHILS-MAB, since the mean objective values and their standard deviations are lower for the majority of the considered benchmarks.

It is worth noting that in the SAT problem domain, most of the HHILS variants behave equally well, since for the majority of the problem instances, they exhibit similar mean and median performance values. In addition, for most of the problem instances, they successfully reach the best objective value achieved in this study. Notice that observable performance gains have been shown mostly by HHILS-SA which reached the smallest (min) objective value for all problem instances.

To facilitate easy comparisons of the performance of each algorithm across all problem domains, instances, and independent runs, their objective values are normalized in a common range of values. As such, for a given problem instance, the obtained objective values of all algorithms defined on a range $O = [O_{\min}, O_{\max}]$ are transformed linearly to the normalized range $N = [N_{\min}, N_{\max}]$, where $O_{\min}$ and $O_{\max}$ are the smallest and the largest objective value observed by the considered algorithms, and $N_{\min} and N_{\max}$ are the lower and upper bound of the new normalized range. Here, the normalized range $N = [0, 1]$ has been used to keep calculations simple. Strictly speaking, let $y \in O \subset \mathbb{R}$ be an objective value obtained by an algorithm for a given instance, and let $f : O \rightarrow N$ be a linear function that transforms its input according to $\xi = f(y) = (y - O_{\min})/(O_{\max} - O_{\min})$. As such, the performance of each algorithm now can be computed by the set $Y_{\text{Alg}} = \{\xi_1, \xi_2, \ldots, \xi_n\}$ that consists of the normalized objective values across all problem domains, problem instances, and independent execution runs, where n indicates the total number of sample values and is equal to the number of problem instances for each domain, times the number of independent runs, summed up over all problem domains. Notice that this normalization enables a summarizing comparison of the algorithms across all problem domains and instances. Intuitively, the smallest normalized objective values indicate better performance.

To graphically demonstrate the distributions of the performance for each algorithm per problem domain, a summarizing illustration is also provided in Fig. 1. This demonstrates box-plot graphs of the normalized performance for the developed hyper-heuristics across all domains. In each box-plot graph, the mean value of the underlying distribution is additionally marked with a diamond.

Notice that the previously described behavior is clearly captured by the summarizing box plots illustrated in Fig. 1. On the whole, it can be easily identified that HHILS-AP exhibits great performance gains in three problem domains (BP, FS, and TSP), while HHILS-SA in two problem domains (VRP and PS). The next best-performing algorithm seems to be HHILS-PM which exhibits competitive performance in BP, PS, SAT, and VRP. HHILS and HHILS-MAB behave quite similarly, and their performance gains cannot be distinguished by the graphical illustration. Interestingly, it can be additionally observed that the performance distribution of HHILS is the most robust against the remaining HHILS variants on the SAT domain.

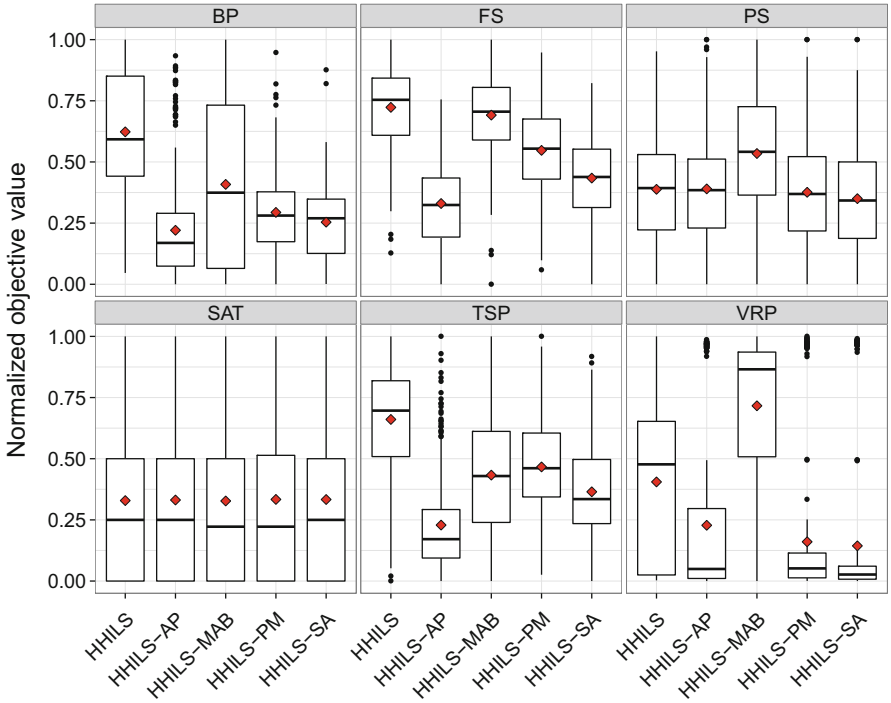


Fig. 1 Boxplot of the normalized objective values of all hyper-heuristics per problem domain

To assess the existence of statistically significant differences, of the observed performance values, between at least two hyper-heuristics across all considered problem domains and instances, the Friedman rank sum test is applied [59]. For each algorithm, the distribution sample of its mean normalized performance per problem instance across all instances and all domains considered in this study is used. The null hypothesis of the Friedman test states that the distributions of all samples are the same, against the alternative hypothesis which states that at least two samples are not the same [59]. Given the existence of significant differences in performance, a post hoc analysis is employed to determine which two algorithms exhibit significant differences in performance. As such, pairwise comparisons are conducted, and for each comparison, the p-values (p_w) calculated by the nonparametric Wilcoxon-signed rank test are reported, since the underlying sample distributions do not follow a normal distribution (tested with the Shapiro-Wilk normality test [59]). Furthermore, to alleviate from the problem of having type I errors in multiple comparisons with a higher probability, the Bonferroni correction method is applied, and the adjusted p-values (p_{bonf}) are reported.

The Friedman test strongly suggests the existence of statistically significant differences in the performances between the studied hyper-heuristics ($p = 0.0000$, with $\chi^2 = 83.071$). Therefore, a post hoc statistical analysis is employed to de-

termine which two algorithms significantly differ in performance. Table 7 presents summarizing statistics and p-values from pairwise statistical significance tests for all considered hyper-heuristics over all problem domains and instances. The left-hand side of Table 7 demonstrates the median (m_f), the mean (μ_f), and the standard deviation (σ_f) of the normalized objective values over all problem domains and instances, while the right-hand side presents the p-values obtained from the Wilcoxon-signed rank test (p_w) and the corresponding Bonferroni correction method (p_{bonf}). The presented statistics show that, on average, HHILS-SA and HHILS-AP are the most promising approaches. However, the statistical tests reveal that there are not statistically significant differences between their performances for the studied cases. Additionally, the pairwise comparisons indicate that HHILS and HHILS-MAB behave equally, while there exist statistically significant differences in performances for the remaining pairs of hyper-heuristics. This verifies the aforementioned reported performances observed in the analyses on each problem domain.

To further demonstrate the underlying performance distribution of the considered hyper-heuristics, Fig. 2 presents two graphical illustrations of their overall performances: a box-plot graph (left) and an empirical cumulative distribution function (ECDF) graph (right) on their normalized objective values over all the considered problem domains and instances. Such graphical illustrations are useful in displaying the characteristics of the distribution and the frequency of the observed performance values. Specifically, having measured a set of performance values (e.g., the normalized objective values) for an algorithm A, $M_A = \{\xi_1, \xi_2, \ldots, \xi_n\}$, the ECDF is defined as $F_n(\xi) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,\xi]}(\xi_i)$, where $I_{(-\infty,\xi]}(\cdot)$ is the indicator function which equals to one if $\xi_i \preceq \xi$ and to zero otherwise. Intuitively, ECDF captures the empirical probability of observing a value that is less than or equal to ξ. The larger a ECDF value corresponding to a given value ξ, the higher is the empirical probability of observing ξ in Y_A.

The box-plot graph (on the left-hand side of Fig. 2) clearly illustrates the differences between the performance distributions of the considered algorithms. Based on the illustrated distributions as well as the previous mentioned results and statistical analyses, it can be safely concluded that HHILS-AP and HHILS-SA demonstrate the most promising performance across all domains. Although, from the analyses conducted per problem domain, HHILS-PM has not exhibited any superior performance gains, it can be observed that its average performance is very promising. Finally, the remaining two hyper-heuristics HHILS and HHILS-MAB perform on average quite similarly. Moreover, the ECDF graph interestingly reveals that almost all algorithms are able to reach best-performing results with quite similar frequency across all problem domains and instances. However, it has to be noticed that HHILS-AP, HHILS-SA, and HHILS-PM have more chance to locate a better solution for the problem under investigation. For example, HHILS-AP has a chance of more than 50% to achieve a quite competitive performance value across all studied problem domains, i.e., there is a 50% chance that the obtained performance value will be in the first quartile of the observed performance values of all algorithms across all studied problem domains.

Table 7 Summarizing statistics of the normalized objective values of all hyper-heuristics over all problem domains and instances (left). Pairwise statistical significance tests of the normalized performances for all hyper-heuristics over all problem domains and instances (right)

Algorithm	m_f	μ_f	σ_f	HHILS		HHILS-AP		HHILS-MAB		HHILS-PM		
HHILS	0.542	0.521	0.293	p_w	p_{bonf}	p_w	p_{bonf}	p_w	p_{bonf}	p_w	p_{bonf}	
HHILS-AP	**0.222**	**0.292**	0.257	HHILS-AP	**0.000**	**0.000**	–	–	–	–	–	–
HHILS-MAB	0.533	0.515	0.309	HHILS-MAB	0.517	1.000	**0.000**	**0.000**	–	–	–	–
HHILS-PM	0.347	0.366	0.263	HHILS-PM	**0.000**	**0.000**	**0.001**	**0.005**	**0.000**	**0.001**	–	–
HHILS-SA	**0.297**	**0.317**	0.251	HHILS-SA	**0.000**	**0.000**	0.604	1.000	**0.000**	**0.000**	**0.000**	**0.000**

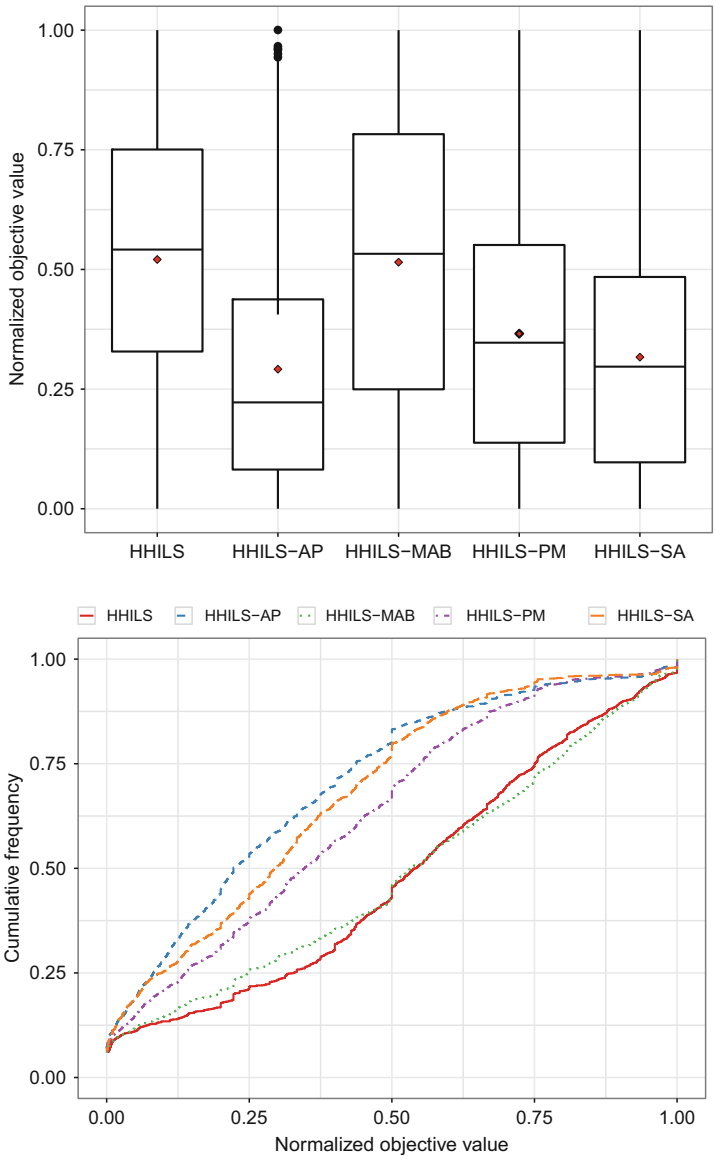


Fig. 2 Summarizing graphical illustrations: (Left) Box-plot graph, (Right) ECDF graph of the normalized objective values of all hyper-heuristics over all problem domains and instances

Comparison to HyFlex State-of-the-Art Hyper-heuristics

In this section, the performance of the HHILS variants is studied and compared against several state-of-the-art hyper-heuristics that participated in the CHeSC 2011 competition. To this end, the same experimental setup has been implemented as provided and described in the website of the CHeSC 2011 competition [1]. Specifically, five problem instances have been chosen (three tests and two hidden instances) for each of the six problem domains, resulting in total 30 different problem instances. Each hyper-heuristic is applied to solve the problem instance at hand for an allowed time period. To calculate the performance of each hyper-heuristic, 31 independent runs are conducted for each problem instance. The median objective value obtained out of these independent runs corresponds to the score of the hyper-heuristic for the specific problem instance at hand. As such, the competition entries are ranked based on their median objective value according to a Formula 1 pointing system. A thorough description of the methodology used for the CHeSC 2011 competition can be found in [98].

Table 8 lists the total and the per-problem domain ranking scores of the CHeSC 2011 competition entries along with the HHILS variants. Specifically, the rows of the table are sorted based on the total score (*total*) of each hyper-heuristic and rank them (*rank*) based on a descending order of their overall score. The remaining columns represent the obtained score of each hyper-heuristic across the different problem domains (for clarity, abbreviations SAT, BP, PS, FS, TSP, and VRP stand for Max-SAT, bin packing, personnel scheduling, flow shop, traveling salesman problem, and vehicle routing with time windows problem domain, respectively).

The scores in Table 8 suggest that the most competitive hyper-heuristics developed in this study, HHILS-SA and HHILS-AP, exhibit very competitive performance compared against CHeSC 2011 entries. HHILS-SA and HHILS-AP hold the second and third position, respectively, on the leaderboard. Notice that the score difference between HHILS-SA/HHILS-AP and the first ranked approach (AdaptHH) is small (9–11 units difference), while there is a large gap between them and the fourth entry (ML), (35–38 units difference). AdaptHH algorithm produces the best performance in two out of the six problem domains (BP and TSP), and it performs really well on the instances used for the FS domain. HHILS-SA exhibits superior performance in the PS and the VRP problem domains and high-performance gains in the SAT problem domain. Similarly, HHILS-AP shows superior performance in the SAT problem domain, while it shows quite competitive scores for the PS, the FS, and the VRP cases.

HHILS-PM is able to exhibit promising performance only in the SAT domain and high scores in the PS and VRP cases, while its contribution to the other domains is minimal. Notice that almost all HHILS variants show superior performance in the SAT domain, with HHILS-AP and HHILS-PM to share the first position in the ranking for this specific domain. In general, it can be observed that the HHILS variants were competitive entries for the SAT, PS, FS, and VRP domains. However,

Table 8 Total and per problem domain ranking scores of the developed hyper-heuristics as competitors on the CHeSC 2011 competition

Rank	*Algorithm*	*Total*	SAT	BP	PS	FS	TSP	VRP
1	AdaptHH	137.25	8.50	**45.00**	3.50	33.00	**38.25**	9.00
2	HHILS-SA	128.35	31.85	12.00	**37.50**	11.00	4.00	**32.00**
3	HHILS-AP	125.95	**32.95**	8.00	25.50	21.50	10.00	28.00
4	ML	90.50	1.00	6.00	23.50	**35.00**	12.00	13.00
5	VNS-TW	86.75	11.00	2.00	24.50	31.00	16.25	2.00
6	HHILS-PM	78.45	**32.95**	3.00	24.50	0.00	0.00	18.00
7	EPH	68.75	0.00	6.00	3.00	16.50	34.25	9.00
8	PHUNTER	68.75	1.00	3.00	7.50	8.00	24.25	25.00
9	NAHH	63.60	6.60	19.00	0.00	22.00	12.00	4.00
10	HHILS	50.45	29.45	0.00	17.00	0.00	0.00	4.00
11	ISEA	48.00	0.00	28.00	11.00	0.00	9.00	0.00
12	HAEA	36.33	0.00	2.00	0.00	5.33	11.00	18.00
13	HHILS-MAB	35.10	23.10	8.00	0.00	0.00	3.00	1.00
14	ACO-HH	32.33	0.00	18.00	0.00	8.33	6.00	0.00
15	HAHA	19.33	7.00	0.00	7.00	0.33	0.00	5.00
16	KSATS-HH	19.00	3.00	8.00	0.00	0.00	0.00	8.00
17	DynILS	18.00	0.00	7.00	0.00	0.00	11.00	0.00
18	GenHive	16.00	0.00	9.00	0.00	3.00	1.00	3.00
19	XCJ	15.00	0.00	11.00	0.00	0.00	0.00	4.00
20	GISS	10.00	0.00	0.00	4.00	0.00	0.00	6.00
21	AVEG-Nep	9.60	6.60	0.00	0.00	0.00	0.00	3.00
22	SA-ILS	9.50	0.00	0.00	6.50	0.00	0.00	3.00
23	SelfSearch	3.00	0.00	0.00	0.00	0.00	3.00	0.00
24	Ant-Q	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25	MCHH-S	0.00	0.00	0.00	0.00	0.00	0.00	0.00

they were not successful in producing a competitive performance, against the other entries, in the BP and TSP cases.

It is worth noting that most of the CHeSC 2011 competitors have a very complex structure that demands a lot of time, effort, and experimentation in the development process, which is a natural process for participating in a competition. However, as suggested by the ranking scores, simple approaches, such as the tutorial demonstration hyper-heuristics in this chapter, might lead to quite efficient search methodologies with competitive performance gains. Notice that several approaches in the class of generation hyper-heuristics try to (semi-)automate the design algorithmic phase in order to alleviate the researcher from having to fine-tune the considered search methodology for a specific (or not) class of problems. Representative examples of automatic algorithmic design can be found in [33, 34, 36].

Conclusions

Hyper-heuristics range over a broad class of search methodologies. Many hyper-heuristic methods can be characterized by their general applicability and robustness across different application domains. The main feature of a hyper-heuristic is that it operates on a space of heuristics rather than on a solution space. Although there has been a significant level of research on hyper-heuristics during the last 15 years or so, several important and challenging research directions in the field are in their infancy and demand further exploration.

This chapter presents a timely and thorough literature review of the main advances in the field of hyper-heuristics since the publication of [36] in 2013. In addition, a simple hyper-heuristic framework has been developed and evaluated as a tutorial-style introduction to the field. The chapter has been divided into three parts. The first part reviews a wide spectrum of advances in the field of hyper-heuristics since 2012 and discusses the current trends and ideas that have been considered by the community. In particular, various studies have been identified that investigate the foundational ideas of the field, consider novel methodologies for automatic design of algorithms and parameter control, provide theoretical analysis, introduce novel hyper-heuristic methodologies and algorithms, develop hyper-heuristic frameworks in multi-objective and dynamic problem formulations, and successfully tackle real-world applications in various fields.

In the second and third parts, a simple but efficient selection hyper-heuristic framework is developed, as a tutorial-style introduction to the field, and evaluates its performance on six problem domains. The hyper-heuristic framework incorporates into the iterated local search algorithm an action selection model to adaptively choose the most promising perturbation heuristic based on feedback during the search process. Various state-of-the-art action selection models have been employed and evaluated. Experimental results and statistical analyses on six different problem domains verify its efficiency and performance. Comparisons with state-of-the-art hyper-heuristics in the field demonstrate its strength despite its relative simplicity.

The literature review highlights various interesting advances and limitations of the current state of the art, which can be summarized as follows.

The current formulation and classification of hyper-heuristics have stimulated the interest of the community. In particular, reformulations and extensions of the definition of hyper-heuristics have been proposed (see, e.g., [128]). A productive criticism on the level of the domain barrier revealed that both high and low levels of abstraction in the problem domain can enhance the search efficiency of hyper-heuristics. Problem domain information is sometimes able to strengthen the search dynamics of a hyper-heuristic without limiting its generality and robustness. Recent advances include unified representations as well as the embedding of mechanisms of common knowledge across different problem domains (see, e.g., [49, 129]).

From the methodological point of view, a variety of novel algorithms and frameworks have recently been published. Heuristic generation methodologies have been developed not only as hyper-visors to generate low-level heuristics

but also as mechanisms to generate high-level models [110]. Moreover, hyper-heuristic frameworks that are able to learn the feature space of a problem class and its representative search strategies have been proposed [57, 120–122]. These frameworks are capable of generating new search strategies and self-adapting their search characteristics based on new problem instances. Furthermore, various analyses have been published that study either the characteristics of the heuristic space, such as its diversity, or the feedback information that can be obtained through the search process (e.g., see [39, 54, 55, 115, 124]). It has also been identified that current developments in the field are mostly empirical, while the theoretical foundations and analyses of hyper-heuristics represent significant open research challenges. A few representative examples of recent theoretical work include the runtime analysis of hyper-heuristic algorithms [8, 74].

A comparison between the fields of meta-learning and hyper-heuristics has been performed that advocates their common objective of automating the algorithm design process [100]. Recent advances in the automatic design of algorithms include various simple, general, and efficient frameworks based on stochastic local search algorithms and novel gramma-based representations of heuristic algorithms among others (e.g., see [6, 80, 85]). Additionally, the commonalities in memetic computing and hyper-heuristics have been reviewed, and new frameworks that combine ideas from both fields have been presented [48]. Various other recent developments in the automated design of algorithms generate efficient algorithms for specific classes of problems, including (among others) packing and cutting problems [116, 132].

A significant number of real-world applications demand high-quality decisions. However, their formulations might include multiple objectives, uncertainty, or dynamically changing environments. Recent trends in the field promote research on both multi-objective and dynamic or uncertain problems. An effective multi-objective algorithm should possess good search characteristics in terms of both solution diversity and convergence on the Pareto-front set. Current developments include multi-objective hyper-heuristic approaches that operate on search strategies to efficiently guide search to diverse Pareto sets that converge to the Pareto front (e.g., see [37, 76, 83, 84, 91, 92, 139]). Current methodologies mostly include selective mechanisms that combine the strengths of predefined search operators or algorithms, while heuristic generation approaches are in their infancy. Clearly, selection hyper-heuristics play a major role in this direction of research. However, the design of a multi-objective algorithm demands expert knowledge of the field, and it is a very demanding and time-consuming development process. Thus, automatic design processes that will efficiently address major issues will arguably have a tremendous impact on the multi-objective community. Hyper-heuristic methodologies for automatically designing and generating multi-objective algorithms are definitely a worthwhile (and growing) research direction.

Hyper-heuristic methodologies that employ adaptivity and cope with dynamically changing problem scenarios have been recently developed to address applications with uncertainty and dynamic changes in time. Most of the developed approaches utilize heuristic selection mechanisms of specialized meta-heuristics

that address dynamic problems across different types of dynamic environments [7, 126, 133]. Exploration of this area is only just beginning.

A major direction that draws upon the interdisciplinary scope of hyper-heuristics is the incorporation of machine learning methodologies to enhance hyper-heuristic efficiency and effectiveness. Machine learning methodologies have been incorporated mostly into the higher level of hyper-heuristics. The main aim of such methodologies is to learn the behavior of the underlying heuristics and be able to make effective decisions to guide the search toward the most promising areas of the search space. Recent advances include methodologies that consider tensor analysis, Inverse Reinforcement Learning theory, Monte Carlo Tree search, and grammatical evolution (among others, e.g., [17–19, 21, 25, 27, 109, 111]). Moreover, hyper-heuristics and the automatic design of algorithms have influenced the machine learning community too. Various developed methodologies include the automatic design of classification algorithms, such as decision trees, to generate more general and robust methodologies for specific class of problems. Clearly, established methodologies in machine learning can potentially boost the strength of hyper-heuristic approaches. As foreseen in [34], it can be recognized that research at the interface of machine learning and hyper-heuristic methodologies has significant potential.

Hyper-heuristics have been successfully applied in a wide variety of diverse application areas. Complex applications that can be modeled as combinatorial optimization problems such as scheduling, timetabling, cutting, and packing problems represent the early adopted application areas of hyper-heuristic approaches. Recent developments not only continue to study these application domains but are also successfully applied to a wide variety of other applications, such as games, engineering, informatics, and parallel and/or distributed applications.

Hyper-heuristic research lies at the interface between operational research and computer science, and it has an impact on a broad spectrum of interdisciplinary application areas. The interdisciplinary character of the field promotes research that combines advanced knowledge from a variety of research areas with the common objective of successfully solving complex real-life optimization problems.

Cross-References

- ▸ Adaptive and Multilevel Metaheuristics
- ▸ Cutting and Packing
- ▸ Data Mining in Stochastic Local Search
- ▸ Genetic Algorithms
- ▸ Iterated Local Search
- ▸ Matheuristics
- ▸ Memetic Algorithms
- ▸ Multi-objective Optimization
- ▸ Network Optimization
- ▸ Particle Swarm Methods

- Restart Strategies
- Tabu Search
- Theoretical Analysis of Stochastic Search Algorithms
- Variable Neighborhood Search

Acknowledgments Michael G. Epitropakis is supported by a grant from the Engineering and Physical Sciences Research Council (EPSRC Grant No. EP/J017515/1), by a Microsoft Azure Grant 2014, and by a Lancaster University Early Career Internal Grant (A100699). This support is gratefully acknowledged.

References

1. (2011) CHeSC 2011: cross-domain heuristic search challenge. http://www.asap.cs.nott.ac.uk/external/chesc2011/. Accessed 25 Mar 2015
2. (2011) HyFlex competition instance summary. http://www.asap.cs.nott.ac.uk/external/chesc2011/reports/CHeSCInstanceSummary.pdf. Accessed 25 Mar 2015
3. (2014) CHeSC 2014: the second cross-domain heuristic search challenge. http://www.hyflex.org/chesc2014/. Accessed 25 Mar 2015
4. (2014) HyFlex API: hyper-heuristics flexible framework API. http://www.hyflex.org/. Accessed 25 Mar 2015
5. Adriaensen S, Brys T, Nowe A (2014) Designing reusable metaheuristic methods: a semi-automated approach. In: 2014 IEEE congress on evolutionary computation (CEC), pp 2969–2976. https://doi.org/10.1109/CEC.2014.6900575
6. Adriaensen S, Brys T, Nowé A (2014) Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO'14). ACM, New York, pp 1303–1310. https://doi.org/10.1145/2576768.2598285
7. Akar E, Topcuoglu HR, Ermis M (2014) Hyper-heuristics for online UAV path planning under imperfect information. In: Esparcia-Alcázar AI, Mora AM (eds) Applications of evolutionary computation. Lecture notes in computer science. Springer, Berlin/Heidelberg, pp 741–752
8. Alanazi F, Lehre PK (2014) Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In: 2014 IEEE congress on evolutionary computation (CEC), pp 2515–2523. https://doi.org/10.1109/CEC.2014.6900602, 00000
9. Aleti A, Moser I (2013) Entropy-based adaptive range parameter control for evolutionary algorithms. In: Proceedings of the 15th annual conference on genetic and evolutionary computation (GECCO'13). ACM, New York, pp 1501–1508. https://doi.org/10.1145/2463372.2463560
10. Aleti A, Moser I (2013) Studying feedback mechanisms for adaptive parameter control in evolutionary algorithms. In: 2013 IEEE congress on evolutionary computation (CEC), pp 3117–3124. https://doi.org/10.1109/CEC.2013.6557950
11. Aleti A, Moser I, Meedeniya I, Grunske L (2013) Choosing the appropriate forecasting model for predictive parameter control. Evol Comput 22(2):319–349. https://doi.org/10.1162/EVCO_a_00113
12. Allen J (2014) A framework for hyper-heuristic optimisation of conceptual aircraft structural designs. Doctoral, Durham University
13. Allen JG, Coates G, Trevelyan J (2013) A hyper-heuristic approach to aircraft structural design optimization. Struct Multidiscip Optim 48(4):807–819. https://doi.org/10.1007/s00158-013-0928-3, 00001
14. Anwar K, Awadallah M, Khader A, Al-betar M (2014) Hyper-heuristic approach for solving nurse rostering problem. In: 2014 IEEE symposium on computational intelligence in ensemble learning (CIEL), pp 1–6. https://doi.org/10.1109/CIEL.2014.7015743

15. Anwar K, Khader AT, Al-Betar MA, Awadallah MA (2014) Development on harmony search hyper-heuristic framework for examination timetabling problem. In: Tan Y, Shi Y, Coello CAC (eds) Advances in swarm intelligence. Lecture notes in computer science, vol 8795. Springer International Publishing, Cham, pp 87–95
16. Aron R, Chana I, Abraham A (2015) A hyper-heuristic approach for resource provisioning-based scheduling in grid environment. J Supercomput 1–24. https://doi.org/10.1007/s11227-014-1373-9
17. Asta S, Özcan E (2014) An apprenticeship learning hyper-heuristic for vehicle routing in HyFlex. In: 2014 IEEE symposium on evolving and autonomous learning systems (EALS), pp 65–72. https://doi.org/10.1109/EALS.2014.7009505
18. Asta S, Özcan E (2014) A tensor-based approach to nurse rostering. In: 10th international conference on the practice and theory of automated timetabling (PATAT 2014), pp 442–445
19. Asta S, Özcan E (2015) A tensor-based selection hyper-heuristic for cross-domain heuristic search. Inf Sci 299:412–432. https://doi.org/10.1016/j.ins.2014.12.020
20. Asta S, Özcan E, Parkes AJ (2013) Batched mode hyper-heuristics. In: Nicosia G, Pardalos P (eds) Learning and intelligent optimization. Lecture notes in computer science. Springer, Berlin/Heidelberg, pp 404–409
21. Asta S, Özcan E, Parkes AJ, Etaner-Uyar S A (2013) Generalizing hyper-heuristics via apprenticeship learning. In: Middendorf M, Blum C (eds) Evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 7832. Springer, Berlin/Heidelberg, pp 169–178
22. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2–3):235–256. https://doi.org/10.1023/A:1013689704352, 01559
23. Bäck T, Fogel DB, Michalewicz Z (eds) (1997) Handbook of evolutionary computation. Oxford University Press, New York
24. Banerjea-Brodeur M (2013) Selection hyper-heuristics for healthcare scheduling. PhD thesis, University of Nottingham
25. Barros RC, Basgalupp MP, Carvalho ACPLFd (2014) Investigating fitness functions for a hyper-heuristic evolutionary algorithm in the context of balanced and imbalanced data classification. Genet Program Evolvable Mach 1–41. https://doi.org/10.1007/s10710-014-9235-z
26. Bartz-Beielstein T, Lasarczyk C, Preuss M (2010) The sequential parameter optimization toolbox. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Experimental methods for the analysis of optimization algorithms. Springer, Berlin/Heidelberg, pp 337–362, 00031
27. Basgalupp MP, Barros RC, Barabasz T (2014) A grammatical evolution based hyper-heuristic for the automatic design of split criteria. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO'14). ACM, New York, pp 1311–1318. https://doi.org/10.1145/2576768.2598327
28. Battiti R, Protasi M (2001) Reactive local search for the maximum clique problem. Algorithmica 29(4):610–637
29. Battiti R, Brunato M, Mascia F (2009) Reactive search and intelligent optimization. Operations research/computer science interfaces series, vol 45. Springer, Boston, 00000
30. Boughaci D, Lassouaoui M (2014) Stochastic hyper-heuristic for the winner determination problem in combinatorial auctions. In: Proceedings of the 6th international conference on management of emergent digital EcoSystems (MEDES'14). ACM, New York, pp 11:62–11:66. https://doi.org/10.1145/2668260.2668268
31. Branke J, Hildebrandt T, Scholz-Reiter B (2014) Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. Evol Comput 1–29. https://doi.org/10.1162/EVCO_a_00131
32. Burke EK, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger GA (eds)

Handbook of metaheuristics. International series in operations research & management science, vol 57. Springer, Boston, pp 457–474
33. Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford CL, Jain LC (eds) Computational intelligence. Intelligent systems reference library, vol 1. Springer, Berlin/Heidelberg, pp 177–201
34. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science, vol 146. Springer, Boston, pp 449–468
35. Burke EK, Qu R, Soghier A (2012) Adaptive selection of heuristics for improving exam timetables. Ann Oper Res 218(1):129–145. https://doi.org/10.1007/s10479-012-1140-3
36. Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724. https://doi.org/10.1057/jors.2013.71
37. Castro OR, Pozo A (2014) A MOPSO based on hyper-heuristic to optimize many-objective problems. In: 2014 IEEE symposium on swarm intelligence (SIS), pp 1–8. https://doi.org/10.1109/SIS.2014.7011803
38. Chakhlevitch K, Cowling P (2008) Hyperheuristics: recent developments. In: Cotta C, Sevaux M, Sorensen K (eds) Adaptive and multilevel metaheuristics. Studies in computational intelligence, vol 136. Springer, Berlin/Heidelberg, pp 3–29
39. Consoli PA, Minku LL, Yao X (2014) Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics. In: Dick G, Browne WN, Whigham P, Zhang M, Bui LT, Ishibuchi H, Jin Y, Li X, Shi Y, Singh P, Tan KC, Tang K (eds) Simulated evolution and learning. Lecture notes in computer science, vol 8886. Springer International Publishing, Cham, pp 359–370, 00000
40. Cowling P, Kendall G, Soubeiga E (2001) A hyperheuristic approach to scheduling a sales summit. In: Burke EK, Erben W (eds) Practice and theory of automated timetabling III. Lecture notes in computer science, vol 2079. Springer, Berlin/Heidelberg, pp 176–190
41. Crowston WBS (1963) Probabilistic and parametric learning combinations of local job shop scheduling rules. Carnegie Institute of Technology and Graduate School of Industrial Administration, Pittsburgh
42. Dong B, Jiao L, Wu J (2015) Graph-based hybrid hyper-heuristic channel scheduling algorithm in multicell networks. Trans Emerg Telecommun Tech n/a–n/a. https://doi.org/10.1002/ett.2923
43. Drake JH, Özcan E, Burke EK (2015) Modified choice function heuristic selection for the multidimensional knapsack problem. In: Sun H, Yang CY, Lin CW, Pan JS, Snasel V, Abraham A (eds) Genetic and evolutionary computing. Advances in intelligent systems and computing, vol 329. Springer International Publishing, Cham, pp 225–234
44. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31
45. Epitropakis MG, Plagianakos VP, Vrahatis MN (2009) Evolutionary adaptation of the differential evolution control parameters. In: IEEE congress on evolutionary computation (CEC'09), pp 1359–1366
46. Epitropakis MG, Tasoulis DK, Pavlidis NG, Plagianakos VP, Vrahatis MN (2012) Tracking differential evolution algorithms: an adaptive approach through multinomial distribution tracking with exponential forgetting. In: Maglogiannis I, Plagianakos V, Vlahavas I (eds) Artificial intelligence: theories and applications. Lecture notes in computer science, vol 7297. Springer, Berlin/Heidelberg, pp 214–222
47. Epitropakis MG, Tasoulis DK, Pavlidis NG, Plagianakos VP, Vrahatis MN (2012) Tracking particle swarm optimizers: an adaptive approach through multinomial distribution tracking with exponential forgetting. In: 2012 IEEE congress on evolutionary computation (CEC), pp 1–8

48. Epitropakis MG, Caraffini F, Neri F, Burke EK (2014) A Separability prototype for automatic memes with adaptive operator selection. In: 2014 IEEE symposium on foundations of computational intelligence (FOCI), pp 70–77. https://doi.org/10.1109/FOCI.2014.7007809
49. Feng L, Ong Y, Lim M, Tsang I (2014) Memetic search with inter-domain learning: a realization between CVRP and CARP. IEEE Trans Evol Comput PP(99):1–1. https://doi.org/10.1109/TEVC.2014.2362558
50. Fialho A (2010) Adaptive operator selection for optimization. Ph.D. thesis, Université Paris-Sud XI, Orsay
51. Fialho A, Costa LD, Schoenauer M, Sebag M (2010) Analyzing bandit-based adaptive operator selection mechanisms. Ann Math Artif Intell 60(1-2):25–64. https://doi.org/10.1007/s10472-010-9213-y, 00032
52. Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall, Englewood Cliffs, pp 225–251
53. Gong W, Fialho A, Cai Z (2010) Adaptive strategy selection in differential evolution. In: Proceedings of the 12th annual conference on genetic and evolutionary computation (GECCO'10). ACM, New York, pp 409–416
54. Grobler J, Engelbrecht A, Kendall G, Yadavalli VSS (2014) The entity-to-algorithm allocation problem: extending the analysis. In: 2014 IEEE symposium on computational intelligence in ensemble learning (CIEL), pp 1–8. https://doi.org/10.1109/CIEL.2014.7015744
55. Grobler J, Engelbrecht AP, Kendall G, Yadavalli VSS (2015) Heuristic space diversity control for improved meta-hyper-heuristic performance. Inf Sci 300:49–62. https://doi.org/10.1016/j.ins.2014.11.012
56. Güney IA, Küçük G, Özcan E (2013) Hyper-heuristics for performance optimization of simultaneous multithreaded processors. In: Gelenbe E, Lent R (eds) Information sciences and systems 2013. Lecture notes in electrical engineering, vol 264. Springer International Publishing, Cham, pp 97–106, 00001
57. Hart E, Sim K (2014) On the life-long learning capabilities of a NELLI*: a hyper-heuristic optimisation system. In: Bartz-Beielstein T, Branke J, Filipč B, Smith J (eds) Parallel problem solving from nature – PPSN XIII. Lecture notes in computer science, vol 8672. Springer International Publishing, Cham, pp 282–291
58. Hildebrandt T, Goswami D, Freitag M (2014) Large-scale simulation-based optimization of semiconductor dispatching rules. In: Proceedings of the 2014 winter simulation conference (WSC'14). IEEE Press, Piscataway, pp 2580–2590, 00000
59. Hollander M, Wolfe DA, Chicken E (2013) Nonparametric statistical methods, 3rd edn. Wiley, Hoboken
60. Hoos H, Stützle T (2004) Stochastic local search: foundations & applications. Morgan Kaufmann Publishers Inc., San Francisco, 01275
61. Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello CAC (ed) Learning and intelligent optimization. Lecture notes in computer science, vol 6683. Springer, Berlin/Heidelberg, pp 507–523, 00149
62. Jackson WG, Özcan E, John RI (2014) Fuzzy adaptive parameter control of a late acceptance hyper-heuristic. In: 2014 14th UK workshop on computational intelligence (UKCI), pp 1–8. https://doi.org/10.1109/UKCI.2014.6930167, 00000
63. Karafotias G, Hoogendoorn M, Eiben AE (2013) Why parameter control mechanisms should be benchmarked against random variation. In: 2013 IEEE congress on evolutionary computation (CEC), pp 349–355. https://doi.org/10.1109/CEC.2013.6557590
64. Karafotias G, Eiben AE, Hoogendoorn M (2014) Generic parameter control with reinforcement learning. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO'14). ACM, New York, pp 1319–1326. https://doi.org/10.1145/2576768.2598360
65. Karafotias G, Eiben E, Hoogendoorn M (2014) Generic parameter control with reinforcement learning. In: Genetic and evolutionary computation conference (GECCO'14), Vancouver, 12–16 July 2014, pp 1319–1326

66. Karafotias G, Hoogendoorn M, Eiben A (2014) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput PP(99):1–1
67. Karafotias G, Hoogendoorn M, Eiben AE (2014) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput PP(99):1–1. https://doi.org/10.1109/TEVC.2014.2308294
68. Kheiri A, Özcan E (2014) Constructing constrained-version of magic squares using selection hyper-heuristics. Comput J 57(3):469–479. https://doi.org/10.1093/comjnl/bxt130, 00001
69. Kheiri A, Özcan E, Parkes AJ (2014) A stochastic local search algorithm with adaptive acceptance for high-school timetabling. Ann Oper Res https://doi.org/10.1007/s10479-014-1660-0
70. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. https://doi.org/10.1126/science.220.4598.671, 00003
71. Koohestani B, Poli R (2014) Evolving an improved algorithm for envelope reduction using a hyper-heuristic approach. IEEE Trans Evol Comput 18(4):543–558. https://doi.org/10.1109/TEVC.2013.2281512, 00000
72. Koulinas G, Kotsikas L, Anagnostopoulos K (2014) A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Inf Sci 277:680–693. https://doi.org/10.1016/j.ins.2014.02.155, 00008
73. Lassouaoui M, Boughaci D (2014) A choice function hyper-heuristic for the winner determination problem. In: Terrazas G, Otero FEB, Masegosa AD (eds) Nature inspired cooperative strategies for optimization (NICSO 2013). Studies in computational intelligence, vol 512. Springer International Publishing, Cham, pp 303–314
74. Lehre PK, Özcan E (2013) A runtime analysis of simple hyper-heuristics: to mix or not to mix operators. In: Proceedings of the twelfth workshop on foundations of genetic algorithms XII (FOGA XII'13). ACM, New York, pp 97–104. https://doi.org/10.1145/2460239.2460249, 00008
75. Li D, Li M, Meng X, Tian Y (2015) A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. IEEE Trans Syst Man Cybern Syst 45(2):315–325. https://doi.org/10.1109/TSMC.2014.2332443
76. Li K, Fialho A, Kwong S, Zhang Q (2014) Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 18(1):114–130. https://doi.org/10.1109/TEVC.2013.2239648
77. Li S (2013) Hyper-heuristic cooperation based approach for bus driver scheduling. Ph.D. thesis, Université de Technologie de Belfort-Montbeliard
78. Liao X, Li Q, Yang X, Zhang W, Li W (2007) Multiobjective optimization for crash safety design of vehicles using stepwise regression model. Struct Multidiscip Optim 35(6):561–569. https://doi.org/10.1007/s00158-007-0163-x
79. Lobo F, Lima C, Michalewicz Z (eds) (2007) Parameter setting in evolutionary algorithms. Studies in computational intelligence, vol 54. Springer, Berlin/Heidelberg
80. López-Camacho E, Terashima-Marin H, Ross P, Ochoa G (2014) A unified hyper-heuristic framework for solving bin packing problems. Expert Syst Appl 41(15):6876–6889. https://doi.org/10.1016/j.eswa.2014.04.043, 00002
81. López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package, iterated race for automatic algorithm configuration. Technical report. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles
82. Lourenço HR, Martin O, Stützle T (2003) Iterated local search, handbook of meta-heuristics. Springer, Berlin/Heidelberg
83. Maashi M, Özcan E, Kendall G (2014) A multi-objective hyper-heuristic based on choice function. Expert Syst Appl 41(9):4475–4493. https://doi.org/10.1016/j.eswa.2013.12.050, 00008
84. Maashi M, Kendall G, Özcan E (2015) Choice function based hyper-heuristics for multi-objective optimization. Appl Soft Comput 28:312–326. https://doi.org/10.1016/j.asoc.2014.12.012

85. Marmion ME, Mascia F, López-Ibáñez M, Stützle T (2013) Automatic design of hybrid stochastic local search algorithms. In: Blesa MJ, Blum C, Festa P, Roli A, Sampels M (eds) Hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Berlin/Heidelberg, pp 144–158
86. Marshall RJ, Johnston M, Zhang M (2014) A comparison between two evolutionary hyper-heuristics for combinatorial optimisation. In: Dick G, Browne WN, Whigham P, Zhang M, Bui LT, Ishibuchi H, Jin Y, Li X, Shi Y, Singh P, Tan KC, Tang K (eds) Simulated evolution and learning, Lecture notes in computer science, vol 8886. Springer International Publishing, Cham, pp 618–630
87. Marshall RJ, Johnston M, Zhang M (2014) Developing a hyper-heuristic using grammatical evolution and the capacitated vehicle routing problem. In: Dick G, Browne WN, Whigham P, Zhang M, Bui LT, Ishibuchi H, Jin Y, Li X, Shi Y, Singh P, Tan KC, Tang K (eds) Simulated evolution and learning, Lecture notes in computer science, vol 8886. Springer International Publishing, Cham, pp 668–679
88. Marshall RJ, Johnston M, Zhang M (2014) Hyper-heuristics, grammatical evolution and the capacitated vehicle routing problem. In: Proceedings of the 2014 conference companion on genetic and evolutionary computation companion (GECCOComp'14). ACM, New York, pp 71–72. https://doi.org/10.1145/2598394.2598407
89. Martin S, Ouelhadj D, Smet P, Vanden Berghe G, Özcan E (2013) Cooperative search for fair nurse rosters. Expert Syst Appl 40(16):6674–6683. https://doi.org/10.1016/j.eswa.2013.06.019
90. Mascia F, López-Ibáñez M, Dubois-Lacoste J, Stützle T (2014) Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Comput Oper Res 51:190–199. https://doi.org/10.1016/j.cor.2014.05.020
91. McClymont K, Keedwell EC, Savić D, Randall-Smith M (2014) Automated construction of evolutionary algorithm operators for the bi-objective water distribution network design problem using a genetic programming based hyper-heuristic approach. J Hydroinf 16(2):302. https://doi.org/10.2166/hydro.2013.226, 00001
92. McClymont K, Keedwell E, Savic D (2015) An analysis of the interface between evolutionary algorithm operators and problem features for water resources problems. A case study in water distribution network design. Environ Model Softw. https://doi.org/10.1016/j.envsoft.2014.12.023
93. Misir M, Lau HC (2014) Diversity-oriented bi-objective hyper-heuristics for patrol scheduling. In: 10th international conference on the practice and theory of automated timetabling (PATAT 2014)
94. Misir M, Smet P, Vanden Berghe G (2014) An analysis of generalised heuristics for vehicle routing and personnel rostering problems. J Oper Res Soc https://doi.org/10.1057/jors.2014.11
95. Neamatian Monemi R, Danach K, Khalil W, Gelareh S, Lima Jr FC, Aloise DJ (2015) Solution methods for scheduling of heterogeneous parallel machines applied to the workover rig problem. Expert Syst Appl 42(9):4493–4505. https://doi.org/10.1016/j.eswa.2015.01.046
96. Nguyen S, Zhang M, Johnston M, Tan KC (2014) Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Trans Evol Comput 18(2):193–208. https://doi.org/10.1109/TEVC.2013.2248159, 00013
97. Ochoa G, Burke EK (2014) Hyperils: an effective iterated local search hyper-heuristic for combinatorial optimisation. In: 10th international conference on the practice and theory of automated timetabling (PATAT 2014)
98. Ochoa G, Hyde M, Curtois T, Vazquez-Rodriguez JA, Walker J, Gendreau M, Kendall G, McCollum B, Parkes AJ, Petrovic S, Burke EK (2012) HyFlex: a benchmark framework for cross-domain heuristic search. In: Hao JK, Middendorf M (eds) Evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 7245. Springer, Berlin/Heidelberg, pp 136–147

99. Ong YS, Keane AJ (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Trans Evol Comput 8(2):99–110. https://doi.org/10.1109/TEVC.2003.819944, 00460
100. Pappa GL, Ochoa G, Hyde MR, Freitas AA, Woodward J, Swan J (2013) Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. Genet Program Evolvable Mach 15(1):3–35. https://doi.org/10.1007/s10710-013-9186-9, 00000
101. Park J, Nguyen S, Johnston M, Zhang M (2013) Evolving stochastic dispatching rules for order acceptance and scheduling via genetic programming. In: Cranefield S, Nayak A (eds) AI 2013: advances in artificial intelligence. Lecture notes in computer science, vol 8272. Springer International Publishing, Berlin, pp 478–489, 00001
102. Pillay N (2014) A review of hyper-heuristics for educational timetabling. Ann Oper Res 1–36. https://doi.org/10.1007/s10479-014-1688-1
103. Poli R, Graff M (2009) There is a free lunch for hyper-heuristics, genetic programming and computer scientists. Springer, Berlin/Heidelberg, pp 195–207
104. Qu R, Pham N, Bai R, Kendall G (2014) Hybridising heuristics within an estimation distribution algorithm for examination timetabling. Appl Intell 1–15. https://doi.org/10.1007/s10489-014-0615-0
105. Ren Z, Jiang H, Xuan J, Hu Y, Luo Z (2014) New insights into diversification of hyper-heuristics. IEEE Trans Cybern 44(10):1747–1761. https://doi.org/10.1109/TCYB.2013.2294185, 00004
106. Ross P (2005) Hyper-heuristics. In: Burke EK, Kendall G (eds) Search methodologies, 1st edn. Springer, New York, pp 529–556
107. Ross P (2014) Hyper-heuristics. In: Burke EK, Kendall G (eds) Search methodologies, 2nd edn. Springer, New York, pp 611–638
108. Ryser-Welch P, Miller JF (2014) Plug-and-play hyper-heuristics: an extended formulation. In: 2014 IEEE eighth international conference on self-adaptive and self-organizing systems (SASO), pp 179–180. https://doi.org/10.1109/SASO.2014.33, 00000
109. Sá AGCd, Pappa GL (2014) A hyper-heuristic evolutionary algorithm for learning Bayesian network classifiers. In: Bazzan ALC, Pichara K (eds) Advances in artificial intelligence – IBERAMIA 2014. Lecture notes in computer science. Springer International Publishing, Cham, pp 430–442
110. Sabar N, Ayob M, Kendall G, Qu R (2014) The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems. IEEE Trans Evol Comput PP(99):1–1. https://doi.org/10.1109/TEVC.2014.2319051
111. Sabar NR, Kendall G (2015) Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. Inf Sci https://doi.org/10.1016/j.ins.2014.10.045
112. Sabar NR, Ayob M, Kendall G, Qu R (2015) A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. IEEE Trans Cybern 45(2):217–228. https://doi.org/10.1109/TCYB.2014.2323936
113. Salcedo-Sanz S, Matías-Román JM, Jiménez-Fernández S, Portilla-Figueras A, Cuadra L (2013) An evolutionary-based hyper-heuristic approach for the Jawbreaker puzzle. Appl Intell 40(3):404–414. https://doi.org/10.1007/s10489-013-0470-4, 00000
114. Salcedo-Sanz S, Jiménez-Fernández S, Matías-Román JM, Portilla-Figueras JA (2014) An educational software tool to teach hyper-heuristics to engineering students based on the Bubble breaker puzzle. Comput Appl Eng Educ n/a–n/a. https://doi.org/10.1002/cae.21597, 00000
115. Salhi A, Rodríguez JAV (2013) Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. Memetic Comput 6(2):77–84. https://doi.org/10.1007/s12293-013-0121-7, 00000
116. Segredo E, Segura C, León C (2013) Memetic algorithms and hyperheuristics applied to a multiobjectivised two-dimensional packing problem. J Glob Optim 58(4):769–794. https://doi.org/10.1007/s10898-013-0088-4, 00000

117. Segredo E, Segura C, Leon C (2014) Fuzzy logic-controlled diversity-based multi-objective memetic algorithm applied to a frequency assignment problem. Eng Appl Artif Intell 30:199–212. https://doi.org/10.1016/j.engappai.2014.01.005
118. Segredo E, Segura C, Leon C, Hart E (2014) A fuzzy logic controller applied to a diversity-based multi-objective evolutionary algorithm for single-objective optimisation. Soft Comput 1–19. https://doi.org/10.1007/s00500-014-1454-y
119. Segredo E, Segura C, Leon C (2014) Control of numeric and symbolic parameters with a hybrid scheme based on fuzzy logic and hyper-heuristics. In: 2014 IEEE congress on evolutionary computation (CEC), pp 1890–1897. https://doi.org/10.1109/CEC.2014.6900538, 00000
120. Sim K, Hart E (2014) An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO'14). ACM, New York, pp 121–128. https://doi.org/10.1145/2576768.2598241
121. Sim K, Hart E, Paechter B (2013) Learning to solve bin packing problems with an immune inspired hyper-heuristic. MIT Press, pp 856–863. https://doi.org/10.7551/978-0-262-31709-2-ch126
122. Sim K, Hart E, Paechter B (2014) A lifelong learning hyper-heuristic method for bin packing. Evol Comput 1–31. https://doi.org/10.1162/EVCO_a_00121
123. Smit SK, Eiben AE (2009) Comparing parameter tuning methods for evolutionary algorithms. In: IEEE congress on evolutionary computation (CEC'09), pp 399–406
124. Soria Alcaraz JA, Ochoa G, Carpio M, Puga H (2014) Evolvability metrics in adaptive operator selection. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO'14). ACM, New York, pp 1327–1334. https://doi.org/10.1145/2576768.2598220, 00001
125. Soria-Alcaraz JA, Ochoa G, Swan J, Carpio M, Puga H, Burke EK (2014) Effective learning hyper-heuristics for the course timetabling problem. Eur J Oper Res 238(1):77–86. https://doi.org/10.1016/j.ejor.2014.03.046, 00006
126. van der Stockt S, Engelbrecht AP (2014) Analysis of hyper-heuristic performance in different dynamic environments. In: 2014 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE), pp 1–8. https://doi.org/10.1109/CIDUE.2014.7007860
127. Sutton RS, Barto AG (1998) Introduction to reinforcement learning, 1st edn. MIT Press, Cambridge, 02767
128. Swan J, Woodward J, Özcan E, Kendall G, Burke EK (2013) Searching the hyper-heuristic design space. Cogn Comput 6(1):66–73. https://doi.org/10.1007/s12559-013-9201-8, 00000
129. Swiercz A, Burke EK, Cichenski M, Pawlak G, Petrovic S, Zurkowski T, Blazewicz J (2013) Unified encoding for hyper-heuristics with application to bioinformatics. CEJOR 22(3):567–589. https://doi.org/10.1007/s10100-013-0321-8, 00000
130. Thierens D (2005) An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th annual conference on genetic and evolutionary computation (GECCO'05). ACM, New York, pp 1539–1546
131. Thierens D (2007) Adaptive strategies for operator allocation. In: Lobo F, Lima C, Michalewicz Z (eds) Parameter setting in evolutionary algorithms. Studies in computational intelligence, vol 54. Springer, Berlin/Heidelberg, pp 77–90, 00042
132. Thomas J, Chaudhari NS (2014) Design of efficient packing system using genetic algorithm based on hyper heuristic approach. Adv Eng Softw 73:45–52. https://doi.org/10.1016/j.advengsoft.2014.03.003, 00000
133. Topcuoglu HR, Ucar A, Altin L (2014) A hyper-heuristic based framework for dynamic optimization problems. Appl Soft Comput 19:236–251. https://doi.org/10.1016/j.asoc.2014.01.037, 00003
134. Tsai CW, Huang WC, Chiang MH, Chiang MC, Yang CS (2014) A hyper-heuristic scheduling algorithm for cloud. IEEE Trans Cloud Comput 2(2):236–250. https://doi.org/10.1109/TCC.2014.2315797, 00003

135. Urra E, Cubillos C, Cabrera-Paniagua D (2015) A hyperheuristic for the dial-a-ride problem with time windows. Math Probl Eng 2015:e707056. https://doi.org/10.1155/2015/707056, 00000
136. Xie J, Mei Y, Ernst AT, Li X, Song A (2014) A genetic programming-based hyper-heuristic approach for storage location assignment problem. In: 2014 IEEE congress on evolutionary computation (CEC), pp 3000–3007. https://doi.org/10.1109/CEC.2014.6900604
137. Yarimcam A, Asta S, Özcan E, Parkes AJ (2014) Heuristic generation via parameter tuning for online bin packing. In: 2014 IEEE symposium on evolving and autonomous learning systems (EALS), pp 102–108. https://doi.org/10.1109/EALS.2014.7009510
138. Yin PY, Chuang KH, Hwang GJ (2014) Developing a context-aware ubiquitous learning system based on a hyper-heuristic approach by taking real-world constraints into account. Univ Access Inf Soc 1–14. https://doi.org/10.1007/s10209-014-0390-z
139. Yuen SY, Zhang X (2014) Multiobjective evolutionary algorithm portfolio: choosing suitable algorithm for multiobjective optimization problem. In: 2014 IEEE congress on evolutionary computation (CEC), pp 1967–1973. https://doi.org/10.1109/CEC.2014.6900470, 00000
140. Zheng YJ, Zhang MX, Ling HF, Chen SY (2015) Emergency railway transportation planning using a hyper-heuristic approach. IEEE Trans Intell Transp Syst 16(1):321–329. https://doi.org/10.1109/TITS.2014.2331239

Iterated Greedy

18

Thomas Stützle and Rubén Ruiz

Contents

Introduction 548
Iterated Greedy 550
  Greedy Construction Heuristics 550
  Iterated Greedy Framework 551
Some Simple Examples of Iterated Greedy Algorithms 555
  TSP Example 555
  SCP Example 556
  PFSP Example 557
Case Study: Iterated Greedy for Flow Shop Scheduling 558
  Results of the Simple Iterated Greedy Without Local Search 561
  Results of the Simple Iterated Greedy with a Local Search Step 562
IG Applications: Historical Development 563
Relationship to Other Approaches 566
  Repeated (Greedy) Construction Algorithms 566
  (Perturbative) Local Search Techniques 567
  Tree Search Algorithms 568
Applications 569
  Iterated Greedy for Scheduling Problems 569
  Iterated Greedy for Routing Problems 570
  Iterated Greedy for Other Problems 570
Conclusions 571
References 572

T. Stützle (✉)
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

R. Ruiz
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politécnica de València, València, Spain
e-mail: rruiz@eio.upv.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_10

Abstract

Iterated greedy is a search method that iterates through applications of construction heuristics using the repeated execution of two main phases, the partial destruction of a complete candidate solution and a subsequent reconstruction of a complete candidate solution. Iterated greedy is based on a simple principle, and methods based on this principle have been proposed and published several times in the literature under different names such as simulated annealing, iterative flattening, ruin-and-recreate, large neighborhood search, and others. Despite its simplicity, iterated greedy has led to rather high-performing algorithms. In combination with other heuristic optimization techniques such as a local search, it has given place to state-of-the-art algorithms for various problems. This paper reviews the main principles of iterated greedy algorithms, relates the basic technique to the various proposals based on this principle, discusses its relationship with other optimization techniques, and gives an overview of problems to which iterated greedy has been successfully applied.

Keywords

Stochastic local search · Metaheuristics · Iterated greedy · Greedy methods · Local search · Constructive search

Introduction

Many effective algorithms for $\mathcal{NP}$-hard combinatorial optimization problems rely on the efficient, repeated execution of some simple, underlying mechanisms. A common example is perturbative search methods that iterate over neighborhood searches or iteratively apply underlying iterative improvement procedures. Examples of such methods include simulated annealing, tabu search, iterated local search, memetic algorithms, or dynamic local search [41,48]. In fact, simulated annealing, tabu search, or dynamic local search at each step explore the neighborhood of a current solution, while iterated local search and memetic algorithms can be seen as iterating across repeated applications of improvement algorithms. Less frequently, stochastic local search (SLS) methods make use of the iterative application of solution construction algorithms. Examples for these latter methods are ant colony optimization (ACO) [23, 24], greedy randomized adaptive search procedures (GRASP) [30, 31], or the pilot and rollout method [11, 27].

In this chapter, we review another SLS method that relies on the iterative application of solution construction procedures: iterated greedy. The method builds a sequence of solutions by iterating through phases of (partial) solution destruction and subsequent reconstruction of a complete candidate solution. A first complete solution in this sequence is generated by some constructive method. Then the following three steps are iteratively executed. First, components of a complete candidate solution s are removed, resulting in a partial solution s_p. Second, starting

from s_p, a complete candidate solution s' is rebuilt. Third, an acceptance criterion decides whether to continue this process from s or s'.

Iterated greedy has a clear underlying principle, and it is generally applicable to any problem for which constructive methods can be conceived. As such, iterated greedy is clearly a general-purpose method. Iterated greedy is a rather simple method that needs typically only short development times, especially if already a constructive heuristic is available. Iterated greedy provides also a rather simple way of improving over the single application of a constructive method, and for various problems very high-quality solutions are generated. Additionally, basic versions of iterated greedy do only incur few main parameters, and their impact on the search process is rather intuitive to understand. All these reasons make iterated greedy a desirable technique for developers of heuristic algorithms.

Given the simple underlying principle, it is maybe also not surprising that the method that we here call iterated greedy has been (re-)discovered and applied a large number of times under different names by different authors (including ourselves). Algorithms that rely to a significant extent on the same underlying principle have been given names such as simulated annealing [50], evolutionary heuristic [62], iterative flattening [17], ruin-and-recreate [95], iterative construction heuristic [86], large neighborhood search [96], or, as here, iterated greedy [48, 92]. We will review these different developments and other related procedures in section "IG Applications: Historical Development."

Despite the possible confusion that may arise for the reader due to the different names and the in part different views on the method, we want to stress that the really important aspect is the principle that underlies all these algorithms. In fact, in all these proposals a repeated usage of constructive methods is made that start from some intermediate, partial candidate solutions. This principle is a generic one of a potential large utility, and it should be understood as one of the basic principles that can be used to develop optimization algorithms. This basic principle may also be only one of the principles that is used in the development of a hybrid optimization algorithm that combines elements from different techniques. For example, several algorithms that make use of the iterated greedy principle include also a local search phase that may improve the solutions generated by the constructive mechanisms [92].

The structure of the chapter is as follows. In section "Iterated Greedy" we review the basic principles of iterated greedy algorithms. Next, in section "Some Simple Examples of Iterated Greedy Algorithms," we give some concrete examples of iterated greedy algorithms. In section "Case Study: Iterated Greedy for Flow Shop Scheduling" we give some results of an experimental study that discusses the main trade-offs in the design of an iterated greedy algorithm for the permutation flow shop problem. We then present other algorithms that make use of the same principle as iterated greedy in section "IG Applications: Historical Development." The relationship of iterated greedy to some other methods is discussed in section "Relationship to Other Approaches." References to some noteworthy applications of iterated greedy are given in section "Applications," and we conclude in section "Conclusions."

Iterated Greedy

Greedy Construction Heuristics

Constructive algorithms build candidate solutions to optimization or decision problems step by step, starting from an empty solution. At each step, they add a solution component to the current partial solution and repeat these steps until a complete candidate solution is obtained. Commonly, constructive algorithms use a heuristic function that estimates for each solution component the benefit of including it into a partial candidate solution. A baseline construction algorithm is formed by so-called greedy (constructive) algorithms that at each step add a solution component for which the value of the heuristic function is the best (see also Fig. 1). If more than one solution component has the same best heuristic value, a tiebreaking criterion is used to decide which solution component is actually added; in the simplest case, this tiebreaking is done uniformly at random, but it also may be done by a secondary heuristic function.

Greedy construction heuristics are frequently used when tackling combinatorial optimization problems due to a number of reasons. First, greedy construction heuristics are rather fast, and at the same time they generate solutions that are usually much better than those generated uniformly at random or by a randomized but heuristically biased construction. Second, these algorithms are often used to seed (perturbative) local search methods such as iterative improvement algorithms; more sophisticated SLS methods such as tabu search and simulated annealing; or population-based methods such as memetic algorithms. In the latter case, typically some members of the population are generated by greedy constructive methods, while others may be randomly generated. Seeding perturbative local search methods with solutions from greedy construction algorithms can incur advantages such as improved quality of local optima, faster identification of local optima, and a better trade-off between computation times and solution quality, that is, better anytime behavior [114]. Third, sometimes one can prove guarantees on the quality of the solutions that are generated in the worst case, leading to so-called approximation algorithms. Often, the best provable guarantees that can be obtained even for more complex SLS algorithms are the guarantees that directly stem from those of the initial greedy construction. Fourth, for various polynomially solvable problems, greedy algorithms are also guaranteed to generate optimal solutions, the Kruskal algorithm for minimum spanning trees being a well-known example. However, for $\mathcal{NP}$-hard problems, this is not the case. Finally, they build the basis for a

Fig. 1 Algorithmic outline of a greedy heuristic

```
procedure Greedy Constructive Heuristic:
    s = ∅
    while s is not a complete solution do
        choose a best rated solution component c
        s = s + c
    end while
end
```

number of other methods such as GRASP [30, 31], ACO [23], or squeaky wheel optimization [52].

One straightforward way to improve over the generation of a single greedy solution is in some cases the repeated application of a greedy heuristic to generate a variety of different candidate solutions and then to choose the best one. Obviously, repetition in this sense is only reasonable if in the construction process different solutions can be generated. For example, for the well-known nearest neighbor heuristic for the traveling salesperson problem (TSP), n distinct nearest neighbor tours may result (assuming no random tiebreaking is done); for each of the n possible cities that may be chosen (randomly) as the initial city for the solution construction, a different tour may result. However, in other cases where the greedy construction is fully deterministic, additional randomization of the construction process, as proposed in the semi-greedy heuristics [45] and in GRASP [30, 84], may be required to generate different solutions.

Repeated construction of solutions also has inherent disadvantages. Constructing a full solution is relatively time-consuming as especially the initial construction steps require a large amount of computation when compared to later construction steps. Furthermore, no information is taken from one solution construction to another one, and thus such a repeated construction does not exploit knowledge gained from previous solutions. A method that alleviates these problems and that allows to invest, in principle, arbitrary computing times to generate different solutions by constructive heuristics is iterated greedy.

Iterated Greedy Framework

The main principle of iterated greedy is to iterate over (greedy) construction methods by first generating a complete candidate solution and then cycling through a main loop that consists of two main steps. In the first step, some solution components are removed from the current complete candidate solution s to result in some intermediate partial candidate solution s_p. We call this the *destruction step*. In the next step, starting from s_p, a construction heuristic is used to generate a new complete solution s'. We call this step the *construction step*. An acceptance test then decides from which of the two solutions, s or s', the next destruction step applies. While in the simplest case, the acceptance test may accept only improved solutions, other choices may lead to more search diversification and, thus, to possibly better results when many iterations of the iterated greedy algorithm are done.

An algorithmic outline of an IG algorithm is given in Fig. 2. It starts by first generating an initial candidate solution using a procedure GenerateInitialSolution and then iterates through a main loop that consists of the application of the three procedures Destruction, Construction, and AcceptanceCriterion. Note that the construction procedures used in GenerateInitialSolution and Construction may be different and, hence, may also use different greedy heuristics. In the simplest case, however, they may be the same procedures.

Fig. 2 Algorithmic outline of Iterated Greedy (IG)

```
procedure Iterated Greedy
    s_0 = GenerateInitialSolution
    repeat
        s_p = Destruction(s*)
        s' = Construction(s_p)
        s* = AcceptanceCriterion(s*, s')
    until termination condition met
end
```

A simple default version of an iterated greedy algorithm could use the following choices. As constructive heuristic one may take one that is either already available or implement a known state-of-the-art constructive heuristic. The solution destruction may delete some randomly chosen solution components, where the number d of solution components to be removed is a parameter of the algorithm. As acceptance criterion, one may force the cost to decrease by only accepting improved or equal quality candidate solutions.

The iterative process around construction heuristics gives IG some specific advantages when compared to the repeated construction of complete candidate solution from scratch. In fact, by starting from a partial solution, one may reduce significantly the time necessary to generate a new candidate solution as less constructive steps need to be done, and the time per construction decision is also reduced as the number of available solution components to choose from is smaller the larger the partial solutions. In addition, through the potential bias exerted by the acceptance criterion, the search process can more easily intensify around the best solutions found in the search process.

When trying to develop a more performing version of an iterated greedy algorithm, many different options for each of the specific operators may be taken into account. Some of the main relevant issues to be considered will be discussed in the following. Other ideas will be discussed also later when considering the relationship of iterated greedy to other methods or when discussing applications of iterated greedy algorithms.

Destruction There are a number of different possible choices for the solution destruction. A first consideration concerns the number of solution components that should be removed, as defined by a parameter d. The extreme settings would correspond to removing only a single component, that is, $d = 1$ or all of them. Even if one may argue that the resulting algorithms should be considered as iterated greedy algorithms, these parameter settings would not correspond to the main ideas underlying iterated greedy. The first case would be akin to a randomized local search, while the latter be akin to a repeated application of a construction heuristic. (We discuss these relationships in more depth in section "Relationship to Other Approaches.") Intermediate values of d result in a trade-off between search intensification and diversification: removing a large number of solution components allows to jump to rather distant solutions in the construction phase, while removing few components leads to a more localized search.

Another choice is whether to leave the number of solution components to be removed fixed or variable during the algorithm run. In the case d is left variable, a scheme of how to adapt its value is required. If the value of d is left fixed, it requires proper tuning. Possibilities for varying the value of d could be to modify this value randomly within some interval every few iterations, to choose the value according to a scheme similar as those introduced for variable neighborhood search [44] or to adapt the value of d at computation time, exploiting ideas of reactive search [9].

Once it is known how many components are to be removed, which ones should be chosen? There are a number of possible answers to this question. Intuitively, a randomized destruction is preferable over deterministic choices to reduce the danger of cycling. If a stochastic destruction is used, the simplest case is to choose components uniformly at random. More involved could be choices that take into account cost measures on the components or the partial solutions that remain, thus introducing a bias in the choice. In that case, solution components that have a strong contribution to the cost of a solution would then have a larger probability of being removed than low cost components. Alternatively, one may use lower bounds on the partial solutions resulting after having removed a component: the smaller the lower bound, the higher the probability.

Construction One of the crucial ideas of IG is that restoring a fully specified solution is done by some form of (greedily biased) constructive mechanism. Hence, the usage of a constructive mechanism is essential. Using a random perturbation (corresponding to a move to a random solution in a large neighborhood) is more in the spirit of reduced variable neighborhood search than in the spirit of IG.

The naming iterated greedy suggests that the construction heuristics used in the algorithm are greedy construction heuristics and typically deterministic (modulo random tiebreaking). In fact, in many implementations of iterated greedy algorithms, this is also the case. However, we want to emphasize here that this need not necessarily be the case as, in principle, any suitable constructive mechanism that starting from some partial candidate solution s_p can generate a complete candidate solution – be it deterministically greedy, probabilistically greedy, or else – may be used as the underlying construction mechanism in an iterated greedy algorithm.

Among the construction rules, one may distinguish between *adaptive* heuristics and *static* ones. In the first case, the heuristic value assigned to a particular choice in the solution construction depends on the partial solution. Typically, adaptive construction heuristics will result in better quality solutions than static ones; however, this improved solution quality is often reached at the cost of higher computation times.

In the simplest case, the solution construction is done following a deterministic construction heuristic – deterministic except of maybe random tiebreaking. Depending on the construction mechanism, it is possible to apply a deterministic rule: by applying stochastic destruction, either the order in which solution components are added is modified or, if adaptive construction heuristics are used, their heuristic evaluation.

Such basic considerations about solution construction in iterated greedy algorithms can be extended in many natural ways by adopting techniques that have been proposed in other methods. One may take inspiration from other constructive methods and use randomized selection in candidate lists built at each construction step as in GRASP algorithms, use biased constructive decisions exploiting past experience such as the pheromone trails in ACO, or use prohibitions by avoiding adding again solution components that have been removed in the solution destruction, taking ideas from tabu search. Another option may be to choose among different possible construction rules and use ideas from the hyperheuristics community for this task [15]. In fact, all kind of methods and techniques that are compatible with the idea of constructing solutions may be adopted within iterated greedy algorithms if this seems promising, making iterated greedy in this sense also a very flexible technique.

Acceptance criterion. The acceptance criterion has a strong influence on the diversification/intensification behavior of an IG algorithm. On the extreme cases are the possibilities of accepting any new solution independent of its solution quality or to only accept solutions that improve over the previous one. There are many intermediate choices such as occasionally accepting worse solutions or allowing backtracks to previously seen solutions. A popular choice when accepting worse solutions is to use acceptance criteria from simulated annealing such as the Metropolis criterion: if a new solution is better or equal, it is accepted; otherwise it is accepted with a probability $\exp\{(f(s) - f(s'))/T\}$, where T is a parameter called temperature. More elaborated approaches might probably make use of short-term memory as in tabu search. The acceptance criterion, whatever it might be, does not need to be applied at every iteration, i.e., a given incumbent solution can be destructed and reconstructed a number of times before deciding on its value. For the choice of the acceptance criterion, very much the same issues arise as in iterated local search [82], and an appropriate choice may be crucial to an IG algorithm's performance.

Hybridization with local search. Iterated greedy algorithms can form directly the basis of hybrid algorithms that combine various search mechanisms. In fact, for constructive heuristics a natural extension is to improve the generated solutions by the application of a (perturbative) local search method, in the simplest case this being an iterative improvement algorithm. Such an extension is also straightforward to be adopted in an iterated greedy algorithm and actually has been done in a number of such approaches. This extension results in an outline of iterated greedy as given in Fig. 3. With this additional local search phase, the IG algorithm also strongly resembles iterated local search (ILS) algorithms [82]. In fact, the destruction and construction phases implement a solution perturbation in the ILS sense. However, a minor difference is that ILS often makes use of perturbations that are randomly chosen from some large neighborhood, while in IG algorithms an underlying constructive method is exploited. More importantly, IG can also reach very high performance when used without the local search phase, which is not necessarily

Fig. 3 Algorithmic outline of an IG with an additional local search step

```
procedure Iterated Greedy with local search
    s_0 = GenerateInitialSolution
    s* = LocalSearch(s_0)
    repeat
        s'_p = Destruction(s*)
        s' = Construction(s'_p)
        s' = LocalSearch(s')
        s* = AcceptanceCriterion(s*, s')
    until termination condition met
end
```

true for ILS algorithms. More on relations of IG to other methods is given in section "Relationship to Other Approaches."

In any case, for an algorithm to qualify as an iterated greedy algorithm, it is necessary that one can distinguish between a clearly available construction mechanism and a clear destruction mechanism that are repeatedly applied in an alternating order. An acceptance criterion may be used in an explicit way or in an implicit way; the latter is the case, for example, if every new solution is accepted and no mentioning of an acceptance criterion is done. In that case, the acceptance criterion would correspond to a "random walk-type" acceptance criterion used sometimes in ILS algorithms [82].

Some Simple Examples of Iterated Greedy Algorithms

Let us consider a few examples of constructive heuristics for basic combinatorial problems that will serve throughout the chapter for illustrating various details of IG algorithms. We consider here three examples for the traveling salesman problem (TSP), the set covering problem (SCP), and the permutation flow shop problem (PFSP). We first shortly introduce the problem and then describe a basic constructive heuristic and basic IG algorithms.

TSP Example

Traveling salesman problem (TSP). The TSP is given a graph $G = (N, E)$ where N is the set of $n = |N|$ nodes and E is the set of edges that fully connects the nodes. To each edge (i, j) is associated a distance d_{ij}. Here we assume that the distance matrix is symmetric, that is, we have $d_{ij} = d_{ji}$ for all $(i, j) \in E$; this type of TSP instances are called symmetric and are among the most widely studied types of TSP instances. The objective in the TSP is to determine a Hamiltonian cycle of minimal length. Such a cycle can be represented by a permutation $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$, where $\pi(i)$ is the node index at position i. The objective function to be minimized is

$$\min_{\pi \in \mathcal{S}} \; d_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} \tag{1}$$

where $\mathcal{S}$ is the search space consisting of the set of all permutations.

Probably the best known constructive heuristic for the TSP is the nearest neighbor heuristic. It chooses randomly a first node to start from to obtain a partial tour $\langle \pi(1) \rangle$. Then at each step it appends to the partial tour $\langle \pi(1) \dots \pi(l) \rangle$, a still unvisited node that has minimum distance to $\pi(l)$. Once all nodes are visited, the tour is completed by closing the tour and going back to $\pi(1)$. The nearest neighbor heuristic usually contains subtours that may be close to optimal ones but also contains long edges that are added often toward the end of the construction. Empirically, for Euclidean two-dimensional TSP instances, the nearest neighbor heuristic generates tours that are about 20–40% above optimal (see [51]).

One possibility to build an iterated greedy algorithm on top of the nearest neighbor heuristic would be to remove a subtour of d consecutive cities of a tour, resulting in one subtour of $n - d$ cities and to restart the nearest neighbor heuristic from it. Another possibility would be to remove randomly cities, reconnect the remaining subtours, and then restart the nearest neighbor heuristic. However, it is unclear what the performance of such an algorithm would be.

Another simple constructive heuristic for the TSP is the random insertion heuristic (RIH). It starts by choosing randomly two nodes that are ordered arbitrarily. At each construction step, a next node is chosen randomly and inserted into a position in the current path such that the cost increase is minimum. The RIH is one of simplest heuristics for the TSP; however, it is among the best performing constructive algorithms for the TSP [51].

IG can be used on top of the RIH as follows. The first solution is constructed by RIH. The procedure Destruct would remove l nodes that are chosen uniformly at random. Next, Construct adds the removed nodes using again the same rules as in the RIH. Finally, AcceptanceCriterion could be chosen to only accept improved solutions. This means that the solution destruction would be applied to the best tour found so far. Different choices for iterated greedy algorithms based on such steps have also been experimentally examined by [95].

SCP Example

Set covering problem (SCP). The set covering problem (SCP) is given a set $A = \{a_1, \dots, a_n\}$ of items and a set $B = \{B_1, \dots, B_m\}$ of subsets of elements of A that covers A; in other words, we have that for each $B_i \subseteq A$ and $\bigcup_{i=1}^{m} = A$. A set B_i covers an item a_j if $a_j \in B_i$. Each set B_i has a cost of c_i. The objective in the SCP is to find a subset C of the sets in B that covers each element in A and that is of minimal total cost.

For the SCP, constructive heuristics differ in the choice of the heuristic function that is used. A standard heuristic function is to compute the cover value of the sets in B, which is defined as $\gamma_i = c_i / b_i$, where b_i is the number of items that would be covered when adding subset B_i to a current partial cover. Note that when starting with an empty solution, that is, none of the subsets B_i is chosen to be in the cover C, we have that $b_i = |B_i|$. However, each time a subset is chosen, b_i needs to be updated taking into account the items already covered in a partial cover C_p. The cover value is an example of an adaptive heuristic, where the heuristic values depend on the partial solution already generated. Adaptive heuristics require higher

computation times than their corresponding static heuristics, which do not update heuristic values in dependence of the partial solution, but typically lead also to better quality solutions. Once a complete cover is obtained, some of the subsets may have become redundant if all the items they cover are also covered by other subsets; removing such redundant subsets then improves the solution cost of the generated cover. Note that the removal of redundant subsets is not yet a destruction step as through the removal of redundant subsets the candidate solution remains a complete cover. The destruction step then may remove a number of subsets from a complete solution, resulting in a partial cover C_p, from which again the construction heuristic starts.

A first IG algorithm for the SCP has been proposed by [50]. They construct the first solution using a simple greedy construction heuristic [7] that at each step first selects a random, not covered item and adds the least cost subset that covers that item. Once a complete cover is found, redundant columns are removed. The solution destruction removes $k_1 \cdot |C|$ subsets that are chosen uniformly at random; here $0 < k_1 < 1$ is a parameter, and $|C|$ is the number of subsets on the cover and the cover size. The solution construction uses the heuristic based on cover values explained above. (The iterated greedy algorithm by [50] is, hence, a first example where the constructive heuristics for generating the initial solution and for extending partial solutions in the main loop differ.) However, the subsets to be considered are limited to a candidate set that comprises all those subsets that have a cost less than $k_2 \cdot \max\{c_i | i \in C\}$, where $k_2 > 0$ is a parameter that influences the size of the candidate set and C is the current cover. At each step of the construction, a subset with a minimum cover value is added to the current partial solution, breaking ties uniformly at random. Once a complete cover is obtained, redundant columns are removed as explained above. The acceptance criterion of a new cover C' is based on the Metropolis condition that is frequently used in simulated annealing algorithms. This simple IG algorithm for SCP obtained very high performance improving over several earlier proposed SCP heuristics.

PFSP Example

Permutation flow shop scheduling problem (PFSP). In the PFSP, n jobs have to be scheduled on m machines. All jobs visit the machines in the same order, each job having an operation at each machine. Time p_{ij} denotes the nonnegative, known, and deterministic processing time that job j needs on machine i. In the PFSP the same processing sequence of the jobs is maintained throughout all machines, and hence the processing sequence is obtained as a permutation of the jobs. The standard objective is to minimize the completion time of the last job in this order, which is also known as makespan ($C_{\max}$). The PFSP arises in many practical situations as it is common to have production lines where machines are disposed in series. The PFSP is a thoroughly studied problem in the scheduling literature with literally hundreds of papers published each year. The minimization of the makespan is the most used criterion in the literature, but not so in practice [36]. Some reviews on the PFSP are [35, 47, 91].

A popular and high performing constructive heuristic for the PFSP is the NEH heuristic [68], named after the initials of the last names of the paper's authors. NEH

is an insertion heuristic, a kind of heuristics that iteratively extends a permutation by inserting at each step one new element into the current partial solution. The NEH heuristic first computes for each job j its sum of the processing time $P_j = \sum_{i=1}^{n} p_{ij}$ and then orders them according to nonincreasing P_j values, resulting in a sequence $\phi(1) \ldots \phi(n)$. The first two jobs according to that order are taken, their two possible sequences $\langle \phi(1)\phi(2) \rangle$ and $\langle \phi(2)\phi(1) \rangle$ are evaluated, and the better of the two is adopted. Then at each step l, the job $\phi(l)$ is considered and tentatively inserted in all possible l positions in the current partial solution $\pi(1) \ldots \pi(l-1)$. Among these tentative insertions, the one resulting in the least increase of the makespan is taken. These steps are repeated until all jobs are inserted. Note that there might be two levels of ties, both at the ordering of the P_j values and at the insertion phase. The NEH heuristic has a computational complexity of $O(n^3m)$ which is lowered to $O(n^2m)$ using the efficient implementation of [97]. There is a rich literature of methods that propose variants of the NEH heuristic, mainly proposing mechanisms to break ties or reinsertions. As shown by [91] and more recently by [32], the NEH is a state-of-the-art constructive heuristic for the PFSP.

A simple IG algorithm for the PFSP has been proposed by [92]. It is based on insertion heuristics such as NEH. In fact, this iterated greedy algorithm uses NEH to construct an initial solution. At each destruction step, it removes a number of jobs that are chosen uniformly at random from the current permutation. In the construction step, these jobs are then reinserted in the same order in which they have been removed. For the insertion of the jobs, the same procedure as described for the NEH heuristic is followed. The acceptance criterion accepts a new candidate solution using the Metropolis condition known from simulated annealing algorithms but with the temperature T set to a constant value. The performance of this simple IG algorithm was very good, outperforming many metaheuristic algorithms for the PFSP. When combined with a local search phase, the proposed IG algorithm was shown to be a new state-of-the-art algorithm for the PFSP [92]. The IG implementation of [32] has given some further improvements of the above presented iterated greedy algorithm. Currently, the top-performing iterated greedy algorithm for the PFSP is the variant of [26], which additionally adds a local search on the partial solution that is obtained after the destruction step. As a matter of fact, this IG method has shown to outperform other more recent, and arguably much more complex, approaches.

Case Study: Iterated Greedy for Flow Shop Scheduling

The development of an effective iterated greedy algorithm requires the algorithm designer to choose which of the various possible implementation choices to take. In this section, we exemplify the development of an iterated greedy algorithm for the PFSP and study the impact specific alternative choices have on iterated greedy performance. Here we examine the impact of various alternative choices for the iterated greedy algorithm by [92] that we presented in the previous section.

In particular, we consider the following design choices in our experimental analysis.

Initial solution. It may seem advisable to start from an as good initial solution as possible. For the PFSP, this would be the NEH heuristic [32, 91] using the accelerations of [97]. However, at least for some instances, it is not clear whether a random or heuristic initialization of iterated greedy provides the best results. In fact, [81] study cases in which only for hard instances and in short CPU times it is advisable to use NEH (or extensions of the NEH) to obtain competitive results.

Destruction strength. A first and foremost decision to be taken is how many elements of a complete candidate solution should be removed. Here, we consider different fixed values for this parameter d, although it may also be interesting to consider schemes of how to vary the value of d at run-time.

Destruction type. The type of destruction determines which components of the incumbent solution are removed. In the simplest case, one may remove components of a solution uniformly at random. Alternatively, one may remove blocks of consecutive elements. In this case, a random block of d jobs could be removed from the sequence. Note that for a same solution π and a block-based destruction, there are only $n - d + 1$ possible choices for the destruction move. These two latter possibilities will be examined here.

However, one may consider a biased destruction, where components of the incumbent solution may be chosen randomly but in a biased way. For the PFSP, for example, jobs generating a large idle time at machines before or after their processing might be disrupting the sequence and are therefore likely to have been misplaced. We leave the study of such a destruction operator for future work.

Construction type. Similarly to the type of destruction move, different ways of how to construct a solution could be examined. The default choice in the iterated greedy algorithm is to insert jobs using the NEH heuristic. As an alternative, we consider a random insertion of the removed jobs, which could be used if no efficient greedy heuristic is available. However, in this case, the difference between iterated greedy and ILS becomes somewhat blurry, especially when additional local search is used.

Acceptance criterion. The acceptance criterion has a direct impact on the balance between intensification and diversification of the search. A simple idea is to accept only better quality solutions, while alternatively one may use the Metropolis condition that occasionally also accepts worse candidate solutions.

Acceptance iterations. Instead of applying the acceptance criterion at each iteration, one may apply an acceptance criterion only each l iterations. This would correspond to accepting for a few iterations every new candidate solution that is generated and only applying the acceptance test after each sequence of l steps.

Local search. Finally, it is well known that local search can have a tremendous impact on the quality of the results achieved, even though iterated greedy algorithms may reach high-quality solutions even without local search. Hence, it may be worthwhile to test the impact of an additional local search phase.

Table 1 Summary of the factors and the levels studied in the experimental analysis

Factor	Abbreviation	Level one	Level two
Initial solution	*Initialization*	NEH	Random
Destruction strength	*Destruct*	4	6
Destruction type	*Destruction_T*	Random	Block
Construction	*Reconstruction_T*	NEH	Random
Acceptance criterion	*Acceptance_C*	SA	Descent
Acceptance iterations	*Iterations_Acc*	1	5
Local search	*LS*	No	Yes

In order to study the different design and implementation alternatives, we have carried out a design of experiments (DOE) approach [67] where the previous factors are analyzed. Seven factors are coded and tested at two levels each, as summarized in Table 1. As a result, a total of $2^7 = 128$ combinations are to be tested in a full factorial experimental design. However, we use a half fractional design, which has most of the power of a full factorial one but requires only half of the runs. We use a 2^{7-1}_{VII} design, which has a high resolution VII: interactions between four factors are aliased (confounded) with interactions between three factors, interactions between five factors aliased with interactions between two factors, and so on. As a result, we can safely study the experimental data since it is highly unlikely that two high-level interactions might be significant. More elaborated techniques for the analysis and calibration of algorithms than the simple exploratory analysis we perform here have been published elsewhere [8]. Each studied combination has been tested on 30 of the hardest instances from Taillard's benchmark (99). This benchmark is composed of 12 groups of 10 instances, each ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. The instances of one particular group are denoted as $n \times m$. For the tests we pick three of the hardest groups, namely, 50×20, 100×20, and 200×20. We are interested in these instances since for most of the remaining problems, the optimum solution is already known, and today's state-of-the-art methods are capable of obtaining near optimum solutions.

Each combination is run five independent times (replicates) with each instance. We also control the number of jobs (n) as a blocking factor with three levels in the experiment. Therefore, the experimental design contains $2^{7-1} \cdot 3 = 192$ treatments, $192 \cdot 10 = 1\,920$ experimental units (for ten instances), and $1\,920 \cdot 5 = 9\,600$ experiments (for five replicates).

The basic iterated greedy algorithm along with all studied alternatives has been implemented in Delphi language and run during a predefined CPU time which is fixed to 30, 60, and 120 s for the instances of 50, 100, and 200 jobs, respectively. The machine used for the tests is a Pentium IV PC/AT computer running at 3.2 GHz with 2 GBytes of RAM memory. The performance measure and response variable of the experimental design are the so-called relative percentage deviation (*RPD*) over the optimum or best known solution (upper bound) for each tested instance:

$$\text{Relative percentage deviation } (RPD) = \frac{Heu_{\text{sol}} - Best_{\text{sol}}}{Best_{\text{sol}}} \times 100 \tag{2}$$

where Heu_{sol} is the solution given by any of the replicates of any iterated greedy variation for a given instance and $Best_{\text{sol}}$ is the optimum solution or the lowest known upper bound for that specific Taillard's instance as of November 2015. The results are analyzed by means of the parametric ANOVA technique. Notice that in this case, the three assumptions of this parametric test (normality, homoscedasticity, and independence of the residual) have to be satisfied. In our experiment, the factor *LS* results to be extremely significant, and it creates large differences in the response variable. As expected, applying the local search results in a statistically significant difference in the average *RPD* of 1.46 considering all results, compared with the average *RPD* of 3.73 without local search. Such a large difference generates normality problems. In order to avoid this situation, we study two separate ANOVAs, one for each level of the *LS* factor.

Results of the Simple Iterated Greedy Without Local Search

All remaining factors after fixing the local search have p-values very close to zero in the resulting ANOVA table. As a result, we focus on the F-Ratio, which is the ratio between the variance generated by a given factor and the residual variance in the studied two-level interaction linear model. The higher this ratio, the more significant the factor or interaction is. Figure 4 shows the means plots for the remaining six factors in order of importance.

From Fig. 4 we can observe how all six studied factors have levels and variants that result in statistically significant differences. Observed differences in the average performance of the iterated greedy when a factor has been set to a given level or variant are depicted with nonoverlapping confidence intervals in the plots. The importance of the F-Ratio is shown in Fig. 4 with the most significant factors to the

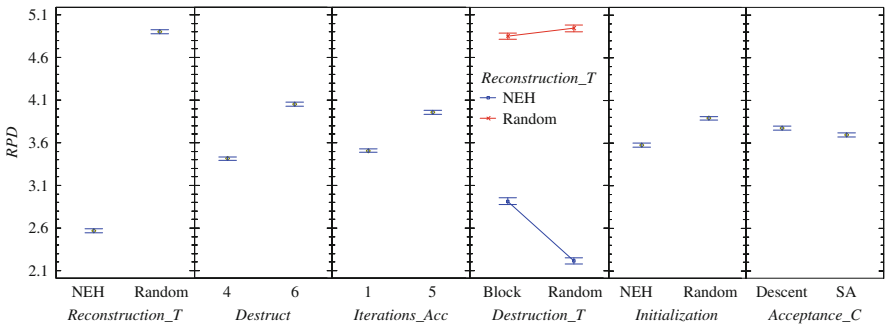


Fig. 4 Means plot of the average relative percentage deviation (*RPD*) and 99% Tukey HSD confidence intervals for the studied factors. Simple iterated greedy without local search

left of the figure and the least significant to the right. As can be seen, without local search, it is clearly preferable to greedily reconstruct the solution as the difference in performance is extremely large. This is one of the main keys behind the iterated greedy algorithm. A random reconstruction of jobs means that first the jobs are randomly extracted and then randomly inserted. This is more or less a sequence of insertion moves. Too many moves (and no local search) result in an algorithm that is not performing well.

The second factor in importance is *Destruct* or d, the number of jobs to be removed in the destruction phase. Following the previous discussion, six jobs impose a rather large overall disruption, and removing only four jobs gives much better results. Notice that a similar conclusion was reached by [92] in their calibration of the iterated greedy for the PFSP. Although not shown here, removing four jobs is better for all the studied instances with 50, 100, and 200 jobs.

The third most important factor results to be *Iterations_Acc*. Among the two tested levels, applying the acceptance criterion at every iteration gives substantially better results. The analysis is continued until all factors have been fixed through the most important effects in the response variable. For example, the third most significant effect in the ANOVA is not a single factor but the interaction between factors *Destruction_T* and *Reconstruction_T*. It is shown in the fourth plot from the left in Fig. 4. We see that both factors interact greatly. When *Reconstruction_T* is set to random, the block destruction appears to be slightly better. However, when *Reconstruction_T* is set to NEH, as in the original NEH method, the destruction has to be done randomly as much better results are obtained. *Initialization* is best set at NEH, but we observe that the difference in performance with the random initialization is rather small. Finally, the acceptance criterion factor (*Acceptance_C*) is also slightly better at SA, but the difference with descent is very small, even though it is statistically significant. More specifically, the average *RPD* given by the simple iterated greedy across all factors with a SA-like acceptance criterion is 3.69%, whereas for the descent criterion it is 3.77%. It is interesting to note that most results confirm the choices made in previous studies as in [92].

Results of the Simple Iterated Greedy with a Local Search Step

Now we proceed with the analysis of the experimental data after focusing on the results of the simple iterated greedy with the local search enabled. In this experiment, the first interesting result is that the observed differences as regards *RPD* among the levels of the factors are much smaller than in the previous one. It is safe to say that the local search is capturing much of the variability of the experiment. Figure 5 contains the means plots for the six factors, in order of importance according to their F-Ratios.

As can be seen, the relative importance of the studied factor is not the same when compared to the previous experiment without local search. The most important factor is now *Iterations_Acc*. Similar to the previous experiment, applying the acceptance criterion at each iteration improves results significantly. The next factor in importance is *Reconstruction_T*. The plot is similar to the previous experiment

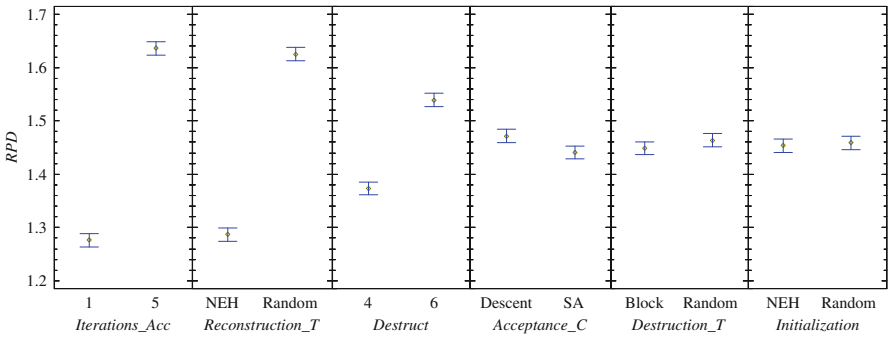


Fig. 5 Means plot of the average relative percentage deviation (*RPD*) and 99% Tukey HSD confidence intervals for the studied factors. Simple iterated greedy with local search enabled

and represents an important result. Considering that the algorithm already carries out a local search step after reconstruction, it could be argued that a random reconstruction might suffice. However, this is not the case. Carrying out a local search step from a randomly reconstructed solution yields, on average, much worse results than if the local search is carried out on a greedily (and therefore, of higher quality) reconstructed solution. This supports the point that iterating over greedy heuristics gives very good results. The remaining factors per importance are *Destruct* and *Acceptance_C*. The last two remaining factors, *Destruction_T* and *Initialization*, are not statistically significant at a 99% confidence level as it can be seen by the fact that their corresponding means plots overlap. Hence, among the two tested types of destruction, it is not important how the jobs are removed. This is probably due to the fact that even though jobs are removed in blocks, they are reinserted one by one. As for the initialization, the advantage gained by an NEH initialization is nullified during the run of the iterated greedy algorithm, confirming the observation of [81].

As a conclusion from this study, the most significant factor is the local search and the NEH reconstruction. Most other factors have less importance. Iterated greedy can even do without a full NEH initialization for the hardest instances of Taillard in the PFSP problem.

IG Applications: Historical Development

Among the examples of iterated greedy algorithms in section "Some Simple Examples of Iterated Greedy Algorithms," we had described an algorithm for the SCP by [50] who have actually called their algorithm a *simulated annealing* algorithm. This naming is probably due to the use of the Metropolis condition as an acceptance criterion, which was also used in the first proposals of simulated annealing [56]. This algorithm has later been extended by the same authors [14]. Somehow related is the more complex algorithm by Marchiori and Steenbeek [62] for the SCP; in

this algorithm a construction starts from a partial solution following the steps of the iterated greedy method; however, instead of obtaining the partial solution by choosing the subsets to be removed, they use a mechanism to choose the subsets of the best solution found so far that are maintained in the partial solution. Maybe surprisingly, in that paper the proposed algorithm is viewed as an evolutionary algorithm. This algorithm is preceded by another algorithm for the unweighted SCP [61], which more directly follows the ideas of the iterated greedy method.

Interestingly, the abovementioned algorithms for the SCP are not the only ones that are directly based on the same destruction–construction process that is the basis of iterated greedy algorithms. In fact, a large number of other algorithms are based on the same principle. However, different names have been used for the underlying method or no specific name at all, which complicates the transfer of knowledge to other researchers.

Probably the earliest techniques that use some mechanism akin to the destructive–constructive moves of iterated greedy are found in the VLSI design automation community and, in particular, in the routing step, where components are connected by wires trying to obey design rules for integrated circuits. In the rip-up and reroute approach [20, 90], some routes related to bottlenecks are removed, and components are reconnected in a different order. The rip-up and reroute technique is commonly used in the design automation community, but the principle underlying this method seems not to have been generalized to the level of metaheuristic for tackling a wide variety of other optimization problems.

The possible use of destructive–constructive moves has also been mentioned earlier in the context of the type of techniques that have become known as strategic oscillation [38–40]. These types of approaches are often embedded into algorithms that make use of tabu search features to provide additional guidance to the search. Some examples of such implementations that have also a clear component related to the iterated greedy methodology are [42, 43, 59].

Among the first researchers to identify the potential of the principles underlying iterated greedy and to formulate these as a general-purpose SLS method are [95]. They called their method *ruin-and-recreate*, where the *ruin* stands for the solution destruction and the *recreate* for the reconstruction of a complete solution.[1] In their seminal article, they applied ruin-and-recreate to the symmetric TSP, the vehicle routing problem with time windows, and to a network design problem, reporting good overall performance of the method. Following this work, a number of other papers have been published using the name ruin-and-recreate; however, not all papers that use the name ruin-and-recreate should be considered as iterated greedy algorithms; for example, [65, 66] essentially uses a random mutation of a current solution instead of a clearly separable destruction phase and greedy solution reconstruction.

[1] Ruin-and-recreate is protected by US patent *Optimization with ruin recreate* No. 6418398; see http://www.patentstorm.us/patents/6418398-fulltext.html.

Ahmadi and Osman [3] study the capacitated clustering $p-$median problem. Among different proposed methods and heuristics, the authors introduce a *periodic construction–deconstruction* procedure, which is basically a simple form of iterated greedy. After a given number of clusters have been built in the constructive method, some of them are randomly deconstructed, and the constructive method is reapplied.

Richmond and Beasley [86] have presented an *iterative construction heuristic* for a problem arising in ore selection. In this problem, processing options needs to be chosen for a number of mining blocks. The destruction operator deletes the processing option for some mining blocks; the size of the destruction is chosen randomly following a uniform distribution in some interval. After the reconstruction of a complete solution, the algorithm accepts only better quality solutions as new incumbent ones. It should be clear from the description that the algorithm follows directly the very same principles of iterated greedy.

Cesta et al. [17] describe an algorithm for a capacitated scheduling problem that makes usage of the ideas underlying the concept of iterated greedy algorithms. They have developed a method that uses a greedy constructive heuristic to generate feasible schedules by introducing precedence constraints. In the destruction phase (called retraction in the chapter), they remove some of the introduced precedence relations to obtain a partial solution and then reconstruct a complete feasible one. They call their method *iterative flattening*, where flattening refers to the reduction of resource conflicts, which is an effect of the introduced precedence constraints. The term iterative flattening has gained some popularity, and several follow-up articles have been published on improvements of the algorithm [63], studies of combinations of basic algorithm components [69], or applications to other problems [70, 71].

The authors of iterative flattening characterize the method in their initial paper as a local search method. Few years before, [96] has proposed the *large neighborhood search* method. The initial proposal consists in the removal of solution components and the reinsertion of solution components by using constraint programming techniques that exploit a tree search and constraint propagation techniques. Even though the method was proposed in a constraint programming framework, where a tree search is used to restore a complete solution instead of simple constructive heuristic, the latter is mentioned as one possible option. In fact, in many later articles on large neighborhood search, simple constructive heuristics are used [78], implementing, hence, directly an iterated greedy method.

Finally, the name *iterated greedy* has been used by [92] and also in [48] to denominate the type of SLS method we describe in this chapter. However, these are not the first publications that are using the name *iterated greedy*. A much earlier mentioning of the term iterated greedy is by Culberson [19]. His iterated greedy algorithm for graph coloring uses a greedy algorithm for generating a coloring, removing the color of all nodes and then reapplying the greedy algorithm using a specific ordering of the nodes that guarantees the next generated coloring to be not worse than the previous one but potentially better. This algorithm may be seen as an extreme case of iterated greedy where the destruction operator destroys the complete solution. However, our view is rather that it is a specific algorithm that works in the way it is proposed mainly for graph coloring.

Relationship to Other Approaches

Iterated greedy is a general-purpose SLS method that has many links to other SLS methods. In the following, we discuss similarities and differences to a number of SLS methods considering other constructive methods, local search methods, and tree search algorithms. These relationships also open up many possibilities for combining techniques proposed for different methods.

Repeated (Greedy) Construction Algorithms

Iterated greedy has natural connections to other methods that repeatedly construct complete candidate solutions. A key difference between iterated greedy and few other repeated construction methods is that the solution construction in iterated greedy usually starts from a (nonempty) partial solution, while the other methods repeatedly generate new candidate solutions starting from *empty* candidate solutions.

A well-known simple such method is the greedy randomized adaptive search procedures (GRASP) that combines a greedily biased but randomized solution construction with a subsequent improvement of the generated candidate solutions by a local search procedure [30, 84]. GRASP in turn extends on some initial ideas of how to generate different solutions by a randomized greedy heuristic [45]. While GRASP does not start the construction from partial solutions, it is an example of a (simple) randomization scheme during the solution construction, which could be included directly in iterated greedy algorithms. An appealing feature of GRASP is the usage of adaptive constructive procedures, and the large number of articles on GRASP [84] may be a source for randomized greedy heuristics to be used in iterated greedy algorithms.

Many useful ideas that underlie other constructive methods may be adopted into iterated greedy algorithms. One such possibility is ways of how to improve the constructive mechanisms, for example, through look-ahead methods or, when taking look-ahead to an extreme, as advocated in the rollout [11] and pilot method [27]. In the rollout and pilot methods for each constructive decision to be taken, a full solution is constructed (or at least well approximated), and the construction decision that results in the best complete solution is taken as the next one. This process is repeated for each construction decision, resulting in a relatively time-consuming constructive approach. However, starting the rollout/pilot method from partial solutions with the resulting reduction in computation time may make this approach promising inside an iterated greedy algorithm.

Squeaky wheel optimization (SWO) [52] uses the idea of biasing the solution construction by information that was gained by analyzing complete candidate solutions. This is done by assigning priorities to specific solution components. The solution reconstruction in SWO is done, differently from iterated greedy, from empty initial solutions corresponding to a complete destruction of the current candidate solution. Within the SWO framework, Aicklin et al. [5] proposed the idea of seeding the reconstruction of a full candidate solution by some partial solution

that was obtained by removing some of the solution components of a complete candidate solution, making it a variant of iterated greedy.

A rather different way of biasing the solution construction is underlying the ant colony optimization (ACO) metaheuristic [23, 24]. In ACO, (artificial) ants implement stochastic solution construction heuristics that make their constructive decisions based on so-called pheromone information and heuristic information associated to specific solution components. The pheromone information tries to bias solution construction toward the best solutions that have been found so far. Again, differently from iterated greedy algorithms, the ants start their solution construction from empty candidate solutions. However, there have been a number of proposals in the ACO area that suggest starting the solution construction from partial solutions that are either stored [1, 2] or obtained by deconstructing complete solutions [103, 104, 109] in a way akin to iterated greedy algorithms. These approaches generally try to transfer the advantages of iterated greedy to ACO algorithms, and for several of these approaches, positive results are reported.

(Perturbative) Local Search Techniques

Iterated greedy has tight links to local search algorithms and, in particular, to very large-scale neighborhood searches [4]. In fact, as also mentioned in section "IG Applications: Historical Development," one of the proposals of iterated greedy style algorithms called the method large neighborhood search Shaw [78]. The analogy stems from the fact that the removal of k solution components in the destruction step and the subsequent reconstruction of a complete candidate solution can be seen as a move in a large neighborhood that is implicitly defined by the number of components to be removed and/or added.[2] The (greedy) reconstruction of a complete candidate solution typical for iterated greedy algorithms can then be seen as a heuristic examination of the neighborhood either in a fully greedy way or using randomization steps–details that simply depend on the particular design and implementation of the construction step in an iterated greedy algorithm. By varying the value of k, the size of the implicitly defined neighborhood is varied. Such changes may be done either (i) randomly within some limited range $[k_{\min}, k_{\max}]$ resembling known strategies in local search methods such as robust tabu search [98], where one tries to avoid overcommitment to a single value for a parameter, (ii) in a more systematic way such as advocated in the strategies defined for variable neighborhood search [44], or (iii) by using feedback from the search performance such as advocated in reactive search methods [9]. An interesting possibility for exploring the neighborhoods is used in the adaptive large neighborhood algorithms

[2]Note that the number of solution components removed in a destruction step may be different from the number of solution components added in the construction step and so we refrain from talking of k-exchange neighborhoods here. A common example where this happens is subset problems such as the SCP we discussed earlier.

by [77, 89], who propose to consider different heuristics to be used in the solution reconstruction. In their method, they additionally adapt the probability with which specific heuristics are chosen over the run-time of the algorithm based on the feedback of the search process.

An alternative to the heuristic exploration of neighborhoods provides methods that try to determine the best possible solution that can be reached from the current one, akin to best-improvement neighborhoods. In the case of large neighborhood searches, this corresponds to an exact examination of all the possible solutions that can be reached from a current partial candidate solution [4, 28]. The original proposal of large neighborhood search by Shaw [96] actually considered such an exact exploration of the resulting neighborhood by means of an constraint programming approach embedded within a branch-and-bound scheme. Due to the large variation in computation time that such a scheme however incurs, alternatives have been tested such as pure insertion-based reconstruction or limited enumeration schemes.

If we consider other local search-based SLS methods, the closest related one is certainly iterated local search [83], especially when iterated greedy algorithms exploit additional (perturbative) local search methods to improve candidate solutions, as outlined in Fig. 3. In that case, the destruction–construction cycle corresponds to what in ILS terms would be a solution perturbation. In general, it seems to be commendable in ILS to apply problem-specific perturbations whenever possible, and the destruction–construction cycle in iterated greedy introduces such problem-specific information through the use of heuristic information. Basic variable neighborhood search is an ILS-type algorithm that systematically varies the strength of the perturbation, and thus the link to iterated greedy is also immediate.

Besides the fact that complete solutions may be improved by an additional local search phase in iterated greedy algorithms, it could be worthwhile to also consider the local re-optimization of the partial solutions that are obtained during the destruction and reconstruction process. It is noteworthy that such occasional local re-optimization of partial candidate solutions has shown to be useful in a number of other contexts such as vehicle routing [16].

Tree Search Algorithms

Constructive mechanisms can be extended to an exhaustive search method by adding a simple backtracking mechanism; adding further bounding schemes, one quickly comes to methods such as branch-and-bound or, more in general, tree search techniques [48]. Hence, it is clear that iterated greedy algorithms also share some relationships to such methods.

Some links have already been mentioned in the previous section, when exact methods are exploited to generate the best possible completion of a partial candidate solution. Such a completion may be generated using tree search methods, and various of these ideas have been explored [96]. Even if tree search is used, the search need not be necessarily complete. For example, Shaw [96] has considered the usage of limited discrepancy search [46], which consists in examining a limited number

of alternative choices in the decision points during the tree generation. Any other variants such as depth-bounded discrepancy search [108] could also be interesting. Another alternative that does not seem to have been applied so far would be to apply beam search [72].

Yet another connection to tree search techniques could rely on the exploitation of lower bound information to generate heuristic information or to prune choices that are guaranteed not to lead to improved solutions. So far, we are not aware of an exploitation of such ideas inside iterated greedy algorithms, although in other constructive SLS algorithms, such uses have been explored [12, 60].

Applications

In this section, we give a short overview of the applications of iterated greedy algorithms. Few applications have already been mentioned in section "IG Applications: Historical Development" when discussing other methods that use the same principle as iterated greedy. Here, we focus mainly on applications that identified their algorithm as an iterated greedy algorithm. In fact, depending on the class of problems for which the respective methods have been proposed, the usage of names to refer to iterated greedy-type methods differs. For example, in the scheduling area, the excellent results of the iterated greedy algorithms for the permutation flow shop scheduling problem have spawned a lot of follow-up work on similar problems, while the name large neighborhood search is frequently found in application to vehicle routing problems, given the prominent role these algorithms have played in that domain.

Iterated Greedy for Scheduling Problems

As mentioned, iterated greedy, as defined in this chapter, was initially applied to the permutation flow shop scheduling problem by [92], so many applications to other scheduling problems were published after that.[3] [93] extended their iterated greedy algorithm to tackle the permutation flow shop scheduling problem when considering additional sequence-dependent setup times and other objectives than makespan, namely, the total tardiness. Other variants of flow shop problems have been studied in [85, 101] (blocking), [75] and [22] (no-wait), [94, 100], and [74] who studied no-idle and mixed no-idle problems. Non-permutation flow shops were approached by [111] or [10]. Distributed flow shop scheduling problems have been solved with iterated greedy algorithms in [57] and [33]. Iterated greedy has also proven valuable in other optimization criteria apart from makespan. Tardiness is studied in [34] or total flowtime in [73] to name just a few. Multi-objective

[3] Various applications of iterative flattening to scheduling problems have been referenced in section "IG Applications: Historical Development".

extensions of iterated greedy have proven effective for Pareto flow shops without and with setups in [64] and in [18], respectively; [25] have embedded an iterated greedy algorithm into the two-phase local search framework to tackle various bi-objective flow shop problems. Parallel machine problems were successfully tackled with iterated greedy algorithms [6, 29, 88]. Iterated greedy algorithms are effective for various single-machine scheduling problems as shown in [21, 112]. Some other more complex problems have been studied, including job-shop scheduling with blocking constraints [80], job-shop scheduling with sequence-dependent setup times and job families [55], hybrid flow shops [110], and real-life problems [105–107].

Iterated Greedy for Routing Problems

As mentioned above, in the context of vehicle routing problems, several algorithms that follow the main structure of an iterated greedy algorithm have been applied but usually using branding under the name large neighborhood search. The article making these approaches popular has already been mentioned [96]. A major impact in that line of applications had the adaptive large neighborhood search approaches [77,89], which led to a large number of follow-up work mainly in the vehicle routing domain. Some overview is also given in [78]. In fact, the usage of such iterated greedy-type methods as local searches inside SLS algorithms for vehicle routing problems is increasingly widespread and can by now be considered a standard in this area. Apart from vehicle routing, also iterated greedy approaches have been proposed to few other routing-type problems such as scheduling and routing problems of freight trains [113] or in the context of TSP variants [54].

Iterated Greedy for Other Problems

There have been a number of other applications where explicitly iterated greedy algorithms have been devised as the main solution techniques. Lozano et al. [58] presented an iterated greedy approach to the maximum diversity problem, where from a set of elements a subset with maximum diversity has to be chosen. After proper tuning, the algorithm was shown to perform better than various competing algorithms and was established as a new state-of-the-art algorithm. García-Martínez et al. [37] have developed an iterated greedy algorithm enhanced by a short tabu list in the destructive phase for the quadratic multiple knapsack problem. Lozano et al. [59] have developed also another iterated greedy algorithm for the quadratic minimum spanning tree problem obtaining excellent results on large instances; further embedding the algorithm into a strategic oscillation approach by essentially extending it with tabu criteria led to further improvements on some problem instance classes. Early implementations of similar ideas for binary quadratic programming have been presented before [43]; more recently, Toyama et al. [102] apply iterated greedy to the same problem tackling also very large-scale instances successfully. Kang et al. [53] study the problem of allocating parallel tasks to processors

in computing systems that are distributed and heterogeneous. Population-based iterated greedy algorithms and iterated greedy algorithms that exploit further exact solutions to large neighborhood searches have been proposed for the maximal covering location problem [87], the goal of which is to cover clients such that the largest amount of demand possible is satisfied. A problem in market segmentation, where a company asks to partition a set of customers subject to some specific requirements related to homogeneity of customers and compactness of the areas is tackled by Huerta-Muñoz et al. [49]. A population-based iterated greedy algorithm has also been applied for delimiting and zoning rural settlements [79] and to the minimum weight vertex cover problem [13]. First applications of iterated greedy algorithms for machine learning tasks, in particular the generation of classification rules, have been explored [76].

Given the wide applicability, the flexibility, and the often high performance of iterated greedy, we would expect this list of applications to further grow significantly in the future.

Conclusions

The main principle of iterated greedy is to build a sequence of solutions by iterating over constructive algorithms through a loop of solution destructions and (re-)constructions. Deconstruction removes solution components resulting in partial solutions from which again full solutions are reconstructed. This loop may be extended by an additional local search phase, where the generated complete candidate solutions are further improved, and by many other techniques from other heuristic and exact search techniques, making iterated greedy a very flexible and malleable method.

We prefer to call this principle iterated greedy because (i) this name directly refers to what the principle implies, namely, making iterative use of construction methods, (ii) it does not obfuscate the name with natural or unnatural analogies, (iii) it is a short and punchy name describing the essence of the method, and (iv) and it has by now been used in a large number of publications. A bit unfortunate is that in the literature there have been proposed several methods under different names that make use of the same (or a very similar) principle including large neighborhood search [78,96], simulated annealing [50], evolutionary heuristic [62], ruin-and-recreate [95], iterative construction search [86], or iterative flattening [17]. The multitude of different names for the same kind of approach can probably be explained in two ways. First, different researchers have a different perspective on the same method, and, thus, the iterated greedy principle can be viewed and proposed, for example, from the perspective of a (perturbative) neighborhood search leading to the heuristic or exact exploration of large neighborhoods or from the perspective of a (constructive) algorithm background where the method is simply iterating through applications of constructive algorithms. Second, the method has been proposed in somewhat different communities such as ruin-and-recreate in a physics journal [95], large neighborhood search at a constraint programming conference [96], and

iterative flattening at an artificial intelligence conference [17], and publications using other names have appeared in operations research journals [50, 86, 92]. It apparently has taken a significant amount of time that these ideas transpired from one community to another.

Even if this leads to some confusion and we maybe contribute to this confusion, we think that what is really important is (i) the principle underlying these methods, (ii) the fact that the iterated greedy principle can lead to very powerful algorithms, and (iii) the fact that iterated greedy is a very flexible method that can easily be combined with other techniques. By making the relationship among the different proposals clear, we hope to contribute also to a transfer of experience between these different algorithms. In any case, we hope that this review chapter will be useful for other researchers in stochastic local search methods by clearly identifying the potential of iterated greedy and in this way also contribute to its further development.

Acknowledgments This work received support from the COMEX project (P7/36) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a research director. Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds.

References

1. Acan A (2004) An external memory implementation in ant colony optimization. In: Dorigo M et al (eds) Ant colony optimization and swarm intelligence, 4th international workshop (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 73–84
2. Acan A (2005) An external partial permutations memory for ant colony optimization. In: Raidl GR, Gottlieb J (eds) Proceedings of EvoCOP 2005 – 5th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 3448. Springer, Heidelberg, pp 1–11
3. Ahmadi S, Osman IH (2004) Density based problem space search for the capacitated clustering p-median problem. Ann Oper Res 131:21–43
4. Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discret Appl Math 123(1–3):75–102
5. Aickelin U, Burke EK, Li J (2006) Improved squeaky wheel optimisation for driver scheduling. In: Runarsson TP, Beyer HG, Burke EK, Merelo JJ, Whitley LD, Yao X (eds) (2006) Proceedings of PPSN-IX, ninth international conference on parallel problem solving from nature. Lecture notes in computer science, vol 4193. Springer, Heidelberg
6. Arroyo J, Leung JT (2017) An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. Comput Ind Eng 105:84–100
7. Balas E, Ho A (1980) Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. Math Program Study 12:37–60
8. Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) (2010) Experimental methods for the analysis of optimization algorithms. Springer, Berlin

9. Battiti R, Brunato M, Mascia F (2008) Reactive search and intelligent optimization. Operations research/computer science interfaces, vol 45. Springer, New York.
10. Benavides AJ, Ritt M (2015) Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. Comput Oper Res 66:160–169
11. Bertsekas DP, Tsitsiklis JN, Wu C (1997) Rollout algorithms for combinatorial optimization. J Heuristics 3(3):245–262
12. Blum C (2005) Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
13. Bouamama S, Blum C, Boukerram A (2012) A population-based iterated greedy algorithm for the minimum weight vertex cover problem. Appl Soft Comput 12(6):1632–1639
14. Brusco MJ, Jacobs LW, Thompson GM (1999) A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set covering problems. Ann Oper Res 86:611–627
15. Burke EK, Gendreau M, Hyde MR, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724
16. Caseau Y, Laburthe F (1999) Heuristics for large constrained vehicle routing problems. J Heuristics 5(3):281–303
17. Cesta A, Oddi A, Smith SF (2000) Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: Proceedings of AAAI 2000 – seventeenth national conference on artificial intelligence. AAAI Press/MIT Press, Menlo Park, pp 742–747
18. Ciavotta M, Minella G, Ruiz R (2013) Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study. Eur J Oper Res 227(2):301–313
19. Culberson JC (1992) Iterated greedy graph coloring and the difficulty landscape. Tech. Rep. 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta
20. Dees WA Jr, Karger PG (1982) Automated rip-up and reroute techniques. In: Proceedings of the 19th design automation workshop (DAC'82). IEEE Press, pp 432–439
21. Deng G, Gu X (2014) An iterated greedy algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. Int J Syst Sci 45(3):351–362
22. Ding JY, Song S, Gupta JND, Zhang R, Chiong R, Wu C (2015) An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. Appl Soft Comput 30:604–613
23. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
24. Dorigo M, Birattari M, Stützle T (2006) Ant colony optimization: artificial ants as a computational intelligence technique. IEEE Comput Intell Mag 1(4):28–39
25. Dubois-Lacoste J, López-Ibáñez M, Stützle T (2011) A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Comput Oper Res 38(8):1219–1236.
26. Dubois-Lacoste J, Pagnozzi F, Stützle T (2017) An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. Comput Oper Res 81:160–166
27. Duin C, Voß S (1999) The pilot method: a strategy for heuristic repetition with application to the Steiner problem in graphs. Networks 34(3):181–191
28. Dumitrescu I, Stützle T (2009) Usage of exact algorithms to enhance stochastic local search algorithms. In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics—hybridizing metaheuristics and mathematical programming. Annals of information systems, vol 10. Springer, New York, pp 103–134
29. Fanjul-Peyro L, Ruiz R (2010) Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 207(1):55–69
30. Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67–71
31. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–113
32. Fernandez-Viagas V, Framiñán JM (2014) On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Comput Oper Res 45:60–67

33. Fernandez-Viagas V, Framinan JM (2015) A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. Int J Prod Res 53(4):1111–1123
34. Framinan JM, Leisten R (2008) Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm. Int J Prod Res 46(22):6479–6498
35. Framiñán JM, Gupta JN, Leisten R (2004) A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. J Oper Res Soc 55(12): 1243–1255
36. Framiñán JM, Leisten R, Ruiz R (2014) Manufacturing scheduling systems: an integrated view on models, methods, and tools. Springer, New York
37. García-Martínez C, Rodriguez FJ, Lozano M (2014) Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem. Eur J Oper Res 232(3): 454–463
38. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8:156–166
39. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549
40. Glover F (1989) Tabu search – part I. INFORMS J Comput 1(3):190–206. https://doi.org/10.1287/ijoc.1.3.190
41. Glover F, Kochenberger G (eds) (2002) Handbook of metaheuristics. Kluwer Academic Publishers, Norwell
42. Glover F, Kochenberger GA (1996) Critical even tabu search for multidimensional knapsack problems. In: Osman IH, Kelly JP (eds) Metaheuristics: theory & applications. Kluwer Academic Publishers, Norwell, pp 407–427
43. Glover F, Kochenberger GA, Alidaee B (1998) Adaptive memory tabu search for binary quadratic programs. Manag Sci 44(3):336–345
44. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130(3):449–467
45. Hart JP, Shogan AW (1987) Semi-greedy heuristics: an empirical study. Oper Res Lett 6(3):107–114
46. Harvey WD, Ginsberg ML (1995) Limited discrepancy search. In: Mellish CS (ed) Proceedings of the fourteenth international joint conference on artificial intelligence (IJCAI-95). Morgan Kaufmann Publishers, pp 607–615
47. Hejazi SR, Saghafian S (2005) Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res 43(14):2895–2929
48. Hoos HH, Stützle T (2005) Stochastic local search—foundations and applications. Morgan Kaufmann Publishers, San Francisco
49. Huerta-Muñoz DL, Ríos-Mercado RZ, Ruiz R (2017) An iterated greedy heuristic for a market segmentation problem with multiple attributes. Eur J Oper Res 261(1):75–87
50. Jacobs LW, Brusco MJ (1995) A local search heuristic for large set-covering problems. Nav Res Logist 42(7):1129–1140
51. Johnson DS, McGeoch LA (2002) Experimental analysis of heuristics for the STSP. In: Gutin G, Punnen A (eds) The traveling salesman problem and its variations. Kluwer Academic Publishers, Dordrecht, pp 369–443
52. Joslin DE, Clements DP (1999) Squeaky wheel optimization. J Artif Intell Res 10:353–373
53. Kang Q, He H, Wei J (2013) An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. J Parallel Distrib Comput 73(8): 1106–1115
54. Karabulut K, Tasgetiren FM (2014) A variable iterated greedy algorithm for the traveling salesman problem with time windows. Inform Sci 279:383–395
55. Kim JS, Park JH, Lee DH (2017) Iterated greedy algorithms to minimize the total family flow time for job-shop scheduling with job families and sequence-dependent set-ups. Eng Optim 49(10):1719–1732
56. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

57. Lin SW, Ying KC, Huang CY (2013) Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. Int J Prod Res 51(16):5029–5038
58. Lozano M, Molina D, García-Martínez C (2011) Iterated greedy for the maximum diversity problem. Eur J Oper Res 214(1):31–38
59. Lozano M, Glover F, García-Martínez C, Rodríguez FJ, Martí R (2014) Tabu search with strategic oscillation for the quadratic minimum spanning tree. IIE Trans 46(4): 414–428
60. Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369
61. Marchiori E, Steenbeek AG (1998) An iterated heuristic algorithm for the set covering problem. In: Mehlhorn K (ed) Algorithm engineering, 2nd international workshop (WAE'92). Max-Planck-Institut für Informatik, Saarbrücken, pp 155–166
62. Marchiori E, Steenbeek AG (2000) An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In: Cagnoni S et al (eds) Real-world applications of evolutionary computing, EvoWorkshops 2000. Lecture notes in computer science, vol 1803. Springer, Heidelberg, pp 367–381
63. Michel LD, van Hentenryck P (2004) Iterative relaxations for iterative flattening in cumulative scheduling. In: Zilberstein S, Koehler J, Koenig S (eds) Proceedings of the fourteenth international conference on automated planning and scheduling (ICAPS 2004). AAAI Press/ MIT Press, Menlo Park, pp 200–208
64. Minella G, Ruiz R, Ciavotta M (2011) Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. Comput Oper Res 38(11):1521–1533
65. Misevičius A (2003) Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem. Knowl Based Syst 16(5–6):261–268
66. Misevičius A (2003) Ruin and recreate principle based approach for the quadratic assignment problem. In: Cantú-Paz E et al (eds) Genetic and evolutionary computation – GECCO 2003, part I. Lecture notes in computer science, vol 2723. Springer, Heidelberg, pp 598–609
67. Montgomery DC (2012) Design and analysis of experiments, 8th edn. Wiley, New York
68. Nawaz M, Enscore E Jr, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11(1):91–95
69. Oddi A, Cesta A, Policella N, Smith SF (2008) Combining variants of iterative flattening search. Eng Appl Artif Intell 21(5):683–690
70. Oddi A, Cesta A, Policella N, Smith SF (2010) Iterative flattening search for resource constrained scheduling. J Intell Manuf 21(1):17–30
71. Oddi A, Rasconi R, Cesta A, Smith SF (2011) Iterative flattening search for the flexible job shop scheduling problem. In: Walsh T (ed) Proceedings of the twenty-second international joint conference on artificial intelligence (IJCAI-11). IJCAI/AAAI Press, Menlo Park, pp 1991–1996
72. Ow PS, Morton TE (1988) Filtered beam search in scheduling. Int J Prod Res 26:297–307
73. Pan QK, Ruiz R (2012) Local search methods for the flowshop scheduling problem with flowtime minimization. Eur J Oper Res 222(1):31–43
74. Pan QK, Ruiz R (2014) An effective iterated greedy algorithm for the mixed no-idle flowshop scheduling problem. Omega 44(1):41–50
75. Pan QK, Wang L, Zhao BH (2008) An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. Int J Adv Manuf Tech 38(7–8): 778–786
76. Pedraza JA, García-Martínez C, Cano A, Ventura S (2014) Classification rule mining with iterated greedy. In: Polycarpou MM, de Carvalho ACPLF, Pan J, Wozniak M, Quintián H, Corchado E (eds) Hybrid artificial intelligence systems – 9th international conference (HAIS 2014), Salamanca, 11–13 June 2014. Proceedings. Lecture notes in computer science, vol 8480. Springer, Heidelberg, pp 585–596
77. Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Comput Oper Res 34(8):2403–2435

78. Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin JY (eds) (2010) Handbook of metaheuristics. International series in operations research & management science, vol 146, 2nd edn. Springer, New York, pp 399–419
79. Porta J, Parapar J, Doallo R, Barbosa V, Santé I, Crecente R, Díaz C (2013) A population-based iterated greedy algorithm for the delimitation and zoning of rural settlements. Comput Environ Urban Syst 39:12–26
80. Pranzo M, Pacciarelli D (2016) An iterated greedy metaheuristic for the blocking job shop scheduling problem. J Heuristics 22(4):587–611.
81. Rad SF, Ruiz R, Boroojerdian N (2009) New high performing heuristics for minimizing makespan in permutation flowshops. Omega 37(2):331–345
82. Ramalhinho Lourenço H, Martin O, Stützle T (2002) Iterated local search. In: Glover F, Kochenberger G (eds) (2002) Handbook of metaheuristics. Kluwer Academic Publishers, Norwell, pp 321–353
83. Ramalhinho Lourenço H, Martin O, Stützle T (2010) Iterated local search: framework and applications. In: Gendreau M, Potvin JY (eds) (2010) Handbook of metaheuristics. International series in operations research & management science, vol 146, 2nd edn. Springer, New York, chap 9, pp 363–397
84. Resende MGC, Ribeiro CC (2010) Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In: Gendreau M, Potvin JY (eds) (2010) Handbook of metaheuristics. International series in operations research & management science, vol 146, 2nd edn. Springer, New York, pp 283–319
85. Ribas I, Companys R, Tort-Martorell X (2011) An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39(3):293–301
86. Richmond AJ, Beasley JE (2004) An iterative construction heuristic for the ore selection problem. J Heuristics 10(2):153–167
87. Rodríguez FJ, Blum C, Lozano M, García-Martínez C (2012) Iterated greedy algorithms for the maximal covering location problem. In: Hao JK, Middendorf M (eds) Proceedings of EvoCOP 2012 – 12th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 7245. Springer, Heidelberg, pp 172–181
88. Rodriguez FJ, Lozano M, Blum C, García-Martínez C (2013) An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. Comput Oper Res 40(7):1829–1841
89. Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problme with time windows. Transp Sci 40(4):455–472
90. Rubin F (1974) An iterative technique for printed wire routing. In: Proceedings of the 11th design automation workshop (DAC'74). IEEE Press, pp 308–313
91. Ruiz R, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165(2):479–494
92. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177(3):2033–2049
93. Ruiz R, Stützle T (2008) An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 187(3):1143–1159
94. Ruiz R, Vallada E, Fernández-Martínez C (2009) Scheduling in flowshops with no-idle machines. In: Chakraborty UK (ed) Computational intelligence in flow shop and job shop scheduling. Studies in computational intelligence, vol 230. Springer, Berlin, pp 21–51
95. Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G (2000) Record breaking optimization results using the ruin and recreate principle. J Comput Phys 159(2):139–171
96. Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget JF (eds) Principles and practice of constraint programming, CP98. Lecture notes in computer science, vol 1520. Springer, Heidelberg, pp 417–431
97. Taillard ÉD (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47(1):65–74

98. Taillard ÉD (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17(4-5):443–455
99. Taillard ÉD (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2): 278–285
100. Tasgetiren FM, Pan QK, Suganthan PN, Buyukdagli O (2013) A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. Comput Oper Res 40(7):1729–1743
101. Tasgetiren MF, Kizilay D, Pan QK, Suganthan PN (2017) Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. Comput Oper Res 77: 111–126
102. Toyama F, Shoji K, Mori H, Miyamichi J (2012) An iterated greedy algorithm for the binary quadratic programming problem. In: Joint 6th international conference on soft computing and intelligent systems (SCIS) and 13th international symposium on advanced intelligent systems (ISIS), 2012. IEEE Press, pp 2183–2188
103. Tsutsui S (2006) cAS: ant colony optimization with cunning ants. In: Runarsson TP, Beyer HG, Burke EK, Merelo JJ, Whitley LD, Yao X (eds) (2006) Proceedings of PPSN-IX, ninth international conference on parallel problem solving from nature. Lecture notes in computer science, vol 4193. Springer, Heidelberg, pp 162–171
104. Tsutsui S (2007) Ant colony optimization with cunning ants. Trans Jpn Soc Artif Intell 22: 29–36.
105. Urlings T, Ruiz R (2007) Local search in complex scheduling problems. In: Stützle T, Birattari M, Hoos HH (eds) Engineering stochastic local search algorithms. Designing, implementing and analyzing effective heuristics. Lecture notes in computer science, vol 4638. Springer, Brussels, Belgium, pp 202–206
106. Urlings T, Ruiz R, Sivrikaya-Şerifoğlu F (2010) Genetic algorithms for complex hybrid flexible flow line problems. Int J Metaheuristics 1(1):30–54
107. Urlings T, Ruiz R, Stützle T (2010) Shifting representation search for hybrid flexible flowline problems. Eur J Oper Res 207(2):1086–1095.
108. Walsh T (1997) Depth-bounded discrepancy search. In: Pollack ME (ed) Proceedings of the fifteenth international joint conference on artificial intelligence (IJCAI-97). Morgan Kaufmann Publishers, pp 1388–1395
109. Wiesemann W, Stützle T (2006) Iterated ants: an experimental study for the quadratic assignment problem. In: Dorigo M, et al. (eds) Ant colony optimization and swarm intelligence, 5th international workshop, ANTS 2006. Lecture notes in computer science, vol 4150. Springer, Heidelberg, pp 179–190
110. Ying KC (2008) An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. IEEE Trans Evol Comput 60(6):810–817
111. Ying KC (2008) Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. Int J Adv Manuf Tech 38(3–4):348–354
112. Ying KC, Lin SW, Huang CY (2009) Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Syst Appl 36(3):7087–7092
113. Yuan Z, Fügenschuh A, Homfeld H, Balaprakash P, Stützle T, Schoch M (2008) Iterated greedy algorithms for a real-world cyclic train scheduling problem. In: Blesa MJ, Blum C, Cotta C, Fernández AJ, Gallardo JE, Roli A, Sampels M (eds) Hybrid metaheuristics. Lecture notes in computer science, vol 5296. Springer, Heidelberg, pp 102–116
114. Zilberstein S (1996) Using anytime algorithms in intelligent systems. AI Mag 17(3):73–83

Iterated Local Search

19

Thomas Stützle and Rubén Ruiz

Contents

Introduction 580
Iterated Local Search 581
Local Search Heuristics 581
Iterated Local Search Framework 583
Some Examples of Iterated Local Search Algorithms 587
TSP Example 587
QAP Example 588
Permutation Flow-Shop Scheduling Example 589
Historical Development of Iterated Local Search 591
Relationship of ILS to 592
. . . Simple SLS Methods 593
. . . Hybrid SLS Methods 593
. . . Population-Based SLS Methods 595
Applications 595
Iterated Local Search for Routing Problems 596
Iterated Local Search for Scheduling Problems 597
Iterated Local Search for Other Problems 598
Conclusions 599
Cross-References 600
References 600

T. Stützle (✉)
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

R. Ruiz
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B, Universitat Politécnica de València, València, Spain
e-mail: rruiz@eio.upv.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_8

Abstract

Iterated local search is a metaheuristic that embeds an improvement heuristic within an iterative process generating a chain of solutions. Often, the improvement method is some kind of local search algorithm and, hence, the name of the metaheuristic. The iterative process in iterated local search consists in a perturbation of the current solution, leading to some intermediate solution that is used as a new starting solution for the improvement method. An additional acceptance criterion decides which of the solutions to keep for continuing this process. This simple idea has led to some very powerful algorithms that have been successfully used to tackle hard combinatorial optimization problems. In this chapter, we review the main ideas of iterated local search, exemplify its application to combinatorial problems, discuss historical aspects of the development of the method, and give an overview of some successful applications.

Keywords

Stochastic local search · Metaheuristics · Iterated local search · Local search · Perturbation · Acceptance criterion

Introduction

One important rationale of a large number of metaheuristics is to overcome the limitations of local optimality incurred by iterative improvement methods [40]. Several concepts for this task have been considered in the prolific literature on metaheuristics that include the occasional acceptance of worsening moves within a local search process, an idea underlying methods such as simulated annealing [20, 64] or tabu search [40], the usage of populations as a simple means for increasing the exploration of the search space as in evolutionary algorithms [7] or ant colony optimization [29], or the introduction of penalties in the search process as in dynamic local search methods [53].

Certainly, one of the simplest ideas is to call iterative improvement methods several times with new starting solutions to sample possibly new and better local optima. Doing so by generating new starting solutions uniformly at random from the search space is known to be an ineffective approach, especially when instance size increases [100]. Iterated local search (ILS) instead exploits the idea of generating a chain of solutions by creating new starting solutions from a perturbation of some of the previously found solutions; in this way, it creates a biased sampling of starting solutions [74]. An acceptance criterion determines whether the new locally optimal solution is taken as the new incumbent or whether the previous solution is maintained. This basic idea underlying ILS has been implemented in a number of early papers on the method [10, 11, 60, 79, 80]. The method has been extended in several directions in follow-up research, and a number of new variants and generalizations have been considered. In fact, a first generalization is that the underlying improvement heuristic need not be necessarily an iterative improvement

algorithm but can be any method that takes some solution as input and returns a possibly improved solution. Other variants concern the introduction of rather elaborate perturbation mechanisms or the consideration of different acceptance criteria. Nevertheless, what characterizes an ILS algorithm is that a chain of solutions is built through a process that involves as main sub-procedures some form of solution perturbation, an improvement method of whatever type, and an acceptance criterion.

In this chapter, we review the main ideas underlying ILS and illustrate its main principles using some example applications to well-known combinatorial optimization problems in section "Some Examples of Iterated Local Search Algorithms". We then shortly highlight, in section "Historical Development of Iterated Local Search", the main developments on the method from a historical perspective, which also shows that ILS is among the oldest metaheuristic techniques. We discuss the most relevant links ILS has to other metaheuristics in section "Relationship of ILS to . . .". We end with an overview of various noteworthy applications of ILS, which complements some earlier reviews on ILS [74, 75], and some concluding remarks in section "Conclusions".

Iterated Local Search

Here, we first introduce basic notions of local search and then elaborate on the main mechanisms ILS uses to steer the search process.

Local Search Heuristics

Many heuristics for tackling hard optimization problems rely on (perturbative) local search [1, 53]. The idea underlying local search is to replace the current candidate solution s within a search space S of complete candidate solutions by some neighboring candidate solution. The neighborhood $N(s)$ of a candidate solution $s \in S$ is defined by all candidate solutions that can be obtained by applying specific modifications or "movements" to s.

Neighborhoods are problem specific, but for many problems, standard neighborhood concepts can be applied. One of these standard neighborhood concepts is the so-called k-exchange neighborhoods, where two candidate solutions are neighbored if they differ in at most k solution components. For implementing such neighborhoods, one needs to identify, for a given problem, appropriate solution components. Intuitively, a solution component corresponds to an atomic relationship, assignment, or selection that characterizes a candidate solution. For example, in the well-known traveling salesman problem (TSP), solution components refer to the fact that a node j is visited directly after a node i in a tour, that is, to an edge (i, j); in the quadratic assignment problem (QAP), a solution component would correspond to the assignment of a facility to a specific location, that is, an individual assignment. In a k-exchange move for the TSP, one would change a set of k edges by a different

set of k edges, and for the QAP, one would change at most k individual assignments of facilities to locations.

Of critical importance is the size of neighborhoods: large neighborhoods have the advantage that better-quality candidate solutions may be reached in one step of the algorithm, which comes, however, with the disadvantage of increasing the time it takes to actually search for such candidate solutions in large neighborhoods. To balance these two aspects, over the years a large set of techniques have been proposed that try to either define special-purpose neighborhood structures that allow to design algorithms for searching them efficiently or to heuristically explore very large neighborhoods trying to identify good candidate solutions there but without guarantee of finding the best one. For an overview of very large neighborhood search methods, we refer to Ahuja et al. [2].

The widest spread perturbative local search method is probably iterative improvement (also called iterated descent, hill-climbing, etc. in the literature). It replaces a current candidate solution by an improving neighboring one and repeats these steps until a locally optimal candidate solution has been found, that is, a candidate solution without any improving neighbor. In the following, we will indicate by $s^* \in S^*$ a locally optimal candidate solution from the set of locally optimal candidate solutions S^*; more in general, we will refer to the set S^* as the possible output set of search methods that are not necessarily iterative improvement methods. Various variants of this basic algorithm exist differing in the order in which the neighborhood of a candidate solution is searched, which neighboring candidate solution replaces the current one, and so on. The most common mechanisms for these rules are the so-called best- and first-improvement rules, which, respectively, replace the current candidate solution by a most-improving candidate solution in $N(s)$ or by the first one encountered when scanning the neighborhood. The specific choice for these pivoting rules can have a significant impact on the quality of the local optima generated and the computational effort it takes to reach local optima. Similarly, for local search algorithms, speed-up techniques such as incremental move evaluations are of crucial importance for their efficiency [1, 53].

Local optima have an ambivalent importance in optimization. When compared to the candidate solutions in the full search space S, locally optimal solutions are of, on average, much better quality, and there are much less candidate solutions in S^* than in S. Hence, identifying local optima by an iterative improvement algorithm may seem a priori a good idea. However, iterative improvement algorithms are stuck in local optima, and even when several candidate solutions in S^* are sampled independently, the so sampled local optima may still be of rather poor quality [100]. Thus, other mechanisms than independent sampling in S^* are required to allow a highly performant search process. The search for such mechanisms led to a large number of proposals and studies of general mechanisms to pursue the local search beyond the traps of local optimality [40] or on the development and study of rather different search mechanisms, such as populations, which try to focus on a more global search behavior. These mechanisms are commonly referred to as metaheuristics [38, 39] or general-purpose stochastic local search methods [53].

Within the context of ILS, of particular relevance are trajectory-based metaheuristics that at each step modify a single solution. This includes methods such as tabu search, simulated annealing or guided local search. Such metaheuristics can be used in a straightforward way as an improvement method in ILS algorithms.

Iterated Local Search Framework

The main principle of ILS is to generate a sequence of candidate solutions that is obtained by iterating through solution perturbations and subsequent applications of a (local) improvement method. An additional acceptance criterion decides from which candidate solution seen during the run of the algorithm this sequence is continued. Typically, the improvement method is an iterative improvement method or a trajectory-following metaheuristic such as tabu search or simulated annealing. The goal of the perturbation is to introduce a modification into the current candidate solution that is larger than the modifications that are done in the local search phase, thus effectively allowing the search to escape local optima or specific search space regions.

An algorithmic outline of ILS is given in Fig. 1. After generating an initial candidate solution and improving it by a local search, the main loop is invoked. Within the main loop, first a (typically locally optimal) candidate solution $s^* \in S^*$ is perturbed leading to a candidate solution $s' \in S$. After improving this solution and obtaining a new solution $s^{*\prime} \in S^*$, an acceptance criterion decides whether to continue the search from s^* or $s^{*\prime}$. It is also possible to consider in the perturbation and the acceptance criterion aspects of the search history, for example, to adjust the perturbation strength or the choice done in the acceptance criterion; in that case, one would extend the corresponding procedures to Perturbation(s^*, *history*) and AcceptanceCriterion($s^*, s^{*\prime}$, *history*).

An important advantage of ILS when compared to repeatedly starting the LocalSearch method from random initial candidate solutions is that (i) intermediate candidate solutions obtained through the perturbation are often of better quality and can lead to better locally optimal solutions, and (ii) in the case of iterative improvement methods, the subsequent local search may terminate more quickly than when starting from random initial candidate solutions, leading to the exploration of a larger number of local optima within a given computation time. Hence, more local

Fig. 1 Algorithmic outline of Iterated Local Search (ILS)

```
procedure Iterated Local Search
    s_0 = GenerateInitialSolution
    s* = LocalSearch(s_0)        % optional
    repeat
        s' = Perturbation(s*)
        s*' = LocalSearch(s')
        s* = AcceptanceCriterion(s*, s*')
    until termination condition met
end
```

optima and local optima of a better average quality may be generated in an ILS algorithm, increasing the chance to find very high-quality solutions.

A further advantage of ILS is its ease of a first implementation, especially if an improvement method is already available. A first version of an ILS algorithm can be obtained by adding perturbations in some higher-order neighborhoods than the ones used in the local search and a reasonable first choice is to accept only better or equal quality solutions in the acceptance criterion. Such a version can be obtained by adding few lines of code to an existing improvement method and be the basis for an algorithm engineering effort to generate a high-performing ILS algorithm. In fact, ILS is very malleable and can range from very straightforward implementations to rather well-engineered algorithms when problem-specific details are taken into account.

Generally speaking, ILS is a modular approach and the behavior of an ILS algorithm is determined by the four procedures GenerateInitialSolution, Perturbation, LocalSearch, and AcceptanceCriterion and their interaction. In what follows, we discuss the main issues arising for these procedures.

GenerateInitialSolution. This procedure determines the starting point of the search and it may have a significant impact during the initial search phase. Generally, candidate solutions generated by a good constructive heuristic should be preferable as often the better the quality of the initial candidate solution, the better is the quality of the local optima encountered. In addition, an iterative improvement algorithm requires usually less steps to reach a local optimum when starting from a better-quality solution. However, if the other procedures are well designed, the influence of the initial candidate solution is expected to be little especially for longer run-times.

Perturbation. The perturbation transforms one complete candidate solution into another complete candidate solution. Its task is to modify a current candidate solution and to generate a new, promising starting solution for the next local search application. To allow an effective escape from local optima, the perturbation should be larger than or at least different in nature from the modifications that are applied in the local search algorithm.

An important factor is the size of the perturbation, which can be measured by the number of solution components that are changed. If the perturbation is too small, it may quickly be undone by the next local search. If the perturbation is too strong, that is, too many solution components are modified, then much of the quality and the structure of the current candidate solution may be lost; in that case, the algorithm can become akin to a random restart algorithm, which is known to be usually of poor performance. Unfortunately, it depends typically on the problem and often on the specific problem instances how strong perturbations should be [74]. One possibility to adapt the perturbation strength may be either to do a proper parameter tuning, an adaptive scheme following ideas of reactive search [9], or to

adapt the perturbation strength using simple schemes such as those of basic variable neighborhood search [45].

The simplest way of perturbing a candidate solution is by applying random perturbation moves. A disadvantage of choosing the perturbing moves uniformly at random is that one may lose significantly in solution quality. An alternative is to use biased perturbations and to introduce problem knowledge in this way. Biased perturbations may make use of heuristic information on the solution components that are changed. This can be done in various forms, for example, by removing preferentially solution components that have a high contribution to the cost of a candidate solution, or by introducing new solution components that have a low cost contribution. Closely related to ILS is the iterated greedy method that relies on a destruction and reconstruction cycle that is managed through constructive heuristics [98]; in fact, this cycle may be seen as a perturbation in the ILS sense. A different possibility is to introduce more complex perturbations by re-optimizing parts of a candidate solution [73] or applying data perturbations [21].

LocalSearch. The local search procedure is of crucial importance for the performance of an ILS algorithm, which will benefit strongly from good choices and implementations for this procedure. A first crucial issue is the quality of the solutions that the local search algorithm produces. A priori, one may expect that the better the quality of the solutions that are generated by the local search, the better the results of the ILS algorithm. However, this reasoning would not take into account the execution time of the local search. In fact, a crucial aspect for the appropriate choice of a local search method is the trade-off between the quality of the solutions it generates and the computation time it takes to find these solutions. Given a fixed computation time, a fast local search may be applied more often and, thus, generate a larger number of candidate solutions that may be of less average quality than candidate solutions obtained by a more time-consuming local search procedure. Which choice is best, depends on the particular problem or type of problem instances and has to be determined on an experimental basis.

Another important aspect is whether the local search is used as a black-box or whether it is open and can be integrated in possibly more profitable ways into an ILS algorithm. One example is the exploitation of the don't look bit technique [14, 61]; we refer to section "TSP Example" for more details on this technique and how to integrate it with the perturbation. Another example requiring access to the local search is the adoption of techniques of tabu search by declaring solution components changed by the perturbation as tabu for a number of local search steps; this may help to avoid immediately undoing the perturbation.

The local search in an ILS algorithm is not limited to an improvement method, but any (even non-local-search) method that takes as input a complete candidate solution and returns a potentially improved candidate solution upon completion could be used. Common examples include the usage of tabu search, simulated annealing, or dynamic local search algorithms as the local search. One can even

think of using an ILS algorithm as the local search leading to a hierarchy of ILS algorithms [55, 74]. However, to allow a sufficient number of iterations of the ILS algorithm, one needs to define a stopping criterion for the local search method, for example, by limiting the number of local search steps that are applied or the number of steps without improvement.

Finally, an advantage of a strong local search is that the sensitivity of the algorithm to the settings of numerical parameters is often reduced, making in this way the algorithm itself more robust. In this sense, ILS also benefits from the further development of local search techniques and the increased availability of powerful implementations based on very large neighborhoods [2].

AcceptanceCriterion. One interpretation of the search behavior of ILS is that of a biased random walk in S^*, the space of local optima. The acceptance criterion has a crucial impact on the nature of this walk and, thus, the trade-off between exploration and exploitation. As the two extremes, one can have the acceptance criterion that only accepts improved solutions, leading to a search that can be characterized as a randomized iterative improvement algorithm in S^*. On the other extreme is the acceptance criterion that accepts any new solution independent of its quality, leading to a random walk in S^*. Clearly, there are many intermediate choices between these extremes, and various have been explored in the literature. One example is the Metropolis condition [85], which accepts a new, worse solution $s^{*\prime}$ with a probability $\exp\{f((s^*) - f(s^{*\prime}))/T\}$, where T is a parameter; a better or equal quality solution is accepted always. The parameter T may be left fixed throughout a run of ILS or be varied as done in simulated annealing.

Various other choices have been made such as combining short random walks with occasional backtracks to the best candidate solution found so far [22], resulting in a specific usage of the search history within the acceptance criterion. Another simple example for the usage of the search history in the acceptance criterion is the occasional restart of the search from a new initial candidate solution, which can lead to much improved algorithm behavior [103, 105].

If one is to reach high performance with an ILS algorithm, the interactions among the components need to be taken into account. The two main interactions are the following. First, the perturbation should complement the local search and ideally introduce moves that cannot be easily done by the local search. This results in two main advantages: (i) it is easier to avoid returning to the previous locally optimal candidate solution, and (ii) the structure of the candidate solution may be changed in a way that is not possible for the local search to do. Second, the acceptance criterion and the perturbation should be well balanced. In fact, both components can be biased toward a more explorative or more exploitative search behavior: large vs. small perturbations or non-strict (random-walk) vs. strict (improvement) acceptance criteria. Finding the right balance between intensification and diversification of the search is one of the key issues in the design of any effective SLS algorithm, and one advantage of ILS is that the impact specific choices of the algorithm components have on this balance is rather easy to grasp. Thus, ILS is a method for which the SLS algorithm engineering process [107] can follow a well-guided direction.

Some Examples of Iterated Local Search Algorithms

We introduce example applications of ILS algorithms to well-known combinatorial optimization problems, the traveling salesman problem (TSP), the quadratic assignment problem (QAP), and the permutation flow-shop scheduling problem (PFSP). Example implementations for these ILS algorithms are available from http://iridia.ulb.ac.be/~stuetzle/Code.

TSP Example

Traveling salesman problem (TSP). The TSP is given as a graph $G = (N, E)$ where N is the set of $n = |N|$ nodes and E is the set of edges that fully connects the nodes. To each edge (i, j) is associated a distance d_{ij}. Here we assume that the distance matrix is symmetric, that is, we have $d_{ij} = d_{ji}$ for all $(i, j) \in E$; this type of TSP instances are called symmetric and are among the most widely studied types of TSP instances. The objective in the TSP is to determine a Hamiltonian cycle of minimal length. Such a cycle can be represented by a permutation $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$, where $\pi(i)$ is the node index at position i. The objective function to be minimized is

$$\min_{\pi \in S} \; d_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} \tag{1}$$

where S is the search space consisting of the set of all permutations.

The TSP is one of the most famous examples of ILS applications, which is due to the important role it plays in the historical development of the method [10, 11, 60, 79] and due to the very high performance ILS algorithms have reached for the TSP [3, 4, 49, 50, 61].

A basic ILS algorithm for the TSP could be the following. The initial candidate solution is generated by a constructive heuristic such as the nearest neighbor heuristic or the greedy heuristic. For the local search, a first-improvement method in a k-edge-exchange neighborhood (typically, k equal two or three) is used. The perturbation is implemented by the so-called double-bridge move, which is illustrated in Fig. 2; the double-bridge move removes four edges and replaces them with four other edges so that in the schematic view of the move, the characteristic double bridge is formed. The acceptance criterion forces the cost to decrease, that is, only better-quality candidate solutions are accepted.

Such a basic ILS algorithm for the TSP reaches high-quality tours, but its performance can be improved by alternative choices for the components. ILS algorithms for the TSP usually benefit from a local search with better performance. For example, by replacing the local search algorithm in the two- or three-edge-exchange neighborhood with the Lin-Kernighan heuristic [70], better performance is usually achieved. It is interesting to note that the double-bridge move was considered within the Lin-Kernighan heuristic, but many implementations of it did not implement it. It is a type of move that cannot be easily achieved by a concatenation of moves in smaller neighborhoods or by the way complex moves in

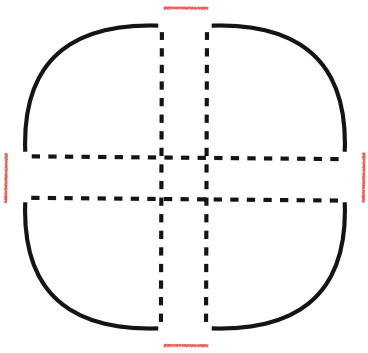


Fig. 2 Schematic view of the double-bridge move. Four edges (continuous, *red lines*) are removed from the current tour, and the four remaining tour segments are reconnected as shown by the *dashed line* in the schematic view of the move

the Lin-Kernighan heuristic are generated, and, thus, it is complementary to standard local search implementations. A main advantage of the double-bridge move is that it only leads to a small increase of the tour length as only four edges are replaced and the subsequent local search typically needs only few improvement steps to reach a new local optimum. In practice, the double-bridge move has found to be rather effective even for very large-sized TSP instances with many thousands or millions of nodes [4, 61].

For speeding-up the search, direct access to the local search through the usage of the well-known don't look bit technique [14] was found to be useful. The don't look bit technique associates one such bit to each node in a TSP instance and starts a local search centered around a node for an improving move only if the don't look bit is set to one (turned on). If this local search is not successful, the don't look bit is turned off by setting it to zero. If a move improves a candidate solution, all end points of the edges involved will have set their don't look bit to one again. This technique leads to a particularly significant speedup, when integrated with the perturbation by re-setting only don't look bits to one for nodes that are directly affected by the perturbation [61] or additionally to some close-by ones [103].

An acceptance criterion that allows only moves to better or equal length tours leads to a strong intensification behavior of the search. While this results in very good performance for short and medium run-times, for longer ILS runs, a stronger diversification through, say, additional restarts improves algorithm performance [103, 105].

QAP Example

Quadratic assignment problem (QAP). In the QAP we are given two matrices F and D corresponding to some kind of flow exchanged between items and a distance matrix between possible locations of the items, respectively. Let f_{ij} be the flow exchanged between items i and j and d_{kl} be the distance between locations k and l. The task in the QAP is to assign items to locations so that the cost given by the sum of flows times the associated distances is minimized. Assuming the number of items is equal to the number of locations, a candidate solution to the QAP can be represented by a permutation π, where $\pi(i)$ gives the location to which item i is assigned. The objective function to be minimized for the QAP can then be written as

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot d_{\pi(i)\pi(j)} \tag{2}$$

Similar to the TSP, the QAP has played a central role in combinatorial optimization. Differently from the TSP, for which optimal solutions can be found for very large instances (the largest being solved optimally having 85,900 nodes [5]), QAP instances can only be solved optimally for relatively small instance sizes of around 30; there are only few exceptions for very particularly structured instances for which an instance of size 128 could be solved [34].

A basic ILS algorithm for the QAP can be implemented as follows. An initial candidate solution for the QAP can be generated uniformly at random, as no constructive algorithms generate generally very high-quality candidate solutions. As local search algorithm, an iterative improvement algorithm in the two-exchange neighborhood, where at each step the locations of two items are exchanged, is used. More formally, two candidate solutions π and π' are neighbored if for exactly one index pair (i, j) we have $\pi'(i) = \pi(j)$ and $\pi'(j) = \pi(i)$ and for all other indices $l \neq i, j$ we have $\pi'(l) = \pi(l)$. As a perturbation, a random move in a k-exchange neighborhood, $k > 2$, can be chosen and, as acceptance criterion, one accepts only better-quality candidate solutions.

Such a straightforward ILS version was tested by Stützle [103] and Lourenço et al. [74], and it was shown that its relative performance with respect to other choices of the main ILS components depended on the specific type of QAP instances. In fact, the specific structure of QAP instances as identified by different distributions of the entries of the flow or the distance matrices can induce widely different behavior [110]. For example, the benchmark instances that have been generated randomly assuming uniform distributions of the matrix entries result in landscapes with many local optima that are spread across the whole search space. Such a landscape requires strong search diversification, which can be achieved in ILS by using an acceptance criterion that accepts every new local optimum independently of its quality and helps to improve the performance on such instances even when using only very small perturbations [74, 104]. Differently, for randomly generated instances that resemble real-life instances, which exchange most flow among relatively few items, accepting only better-quality candidate solutions combined with some occasional restarts was found to result in high performance [74, 104]. Further studies on the QAP have shown that plain ILS algorithms reach very high performance for a variety of QAP instances and that population-based extensions of ILS algorithms are state of the art for tackling various classes of QAP instances [104].

Permutation Flow-Shop Scheduling Example

Permutation flow-shop scheduling problem (PFSP). The PFSP arises in many practical situations as it is common to have production lines where machines are disposed in series. In the PFSP are given n jobs $J_1, J_2, \ldots J_n$, each consisting of m operations that are to be executed on m distinct machines $M_1, M_2, \ldots, M_m$. Each job needs to be processed on

every machine and all jobs pass through the machines in the same machine order, that is, first on machine M_1, next on machine M_2, and so on until machine M_m. The non-negative, known and deterministic processing time of job J_i on machine M_j is denoted by p_{ij}. The PFSP enforces that all jobs are processed on every machine in the same order; therefore, a permutation of the job indices is enough to define the processing sequence. In the basic PFSP, one assumes that jobs are available for processing at time zero, pre-emption is not allowed, each machine can process only one job at a time and each job can be processed by at most one machine at a time, and that the capacity of buffers between machines is unlimited. A common objective is to minimize the completion time of the last job in the permutation, also known as makespan ($C_{\max}$).

The PFSP with the makespan objective is one of the most widely studied scheduling problems with hundreds of papers published; for reviews on it, see [33, 35, 48, 98]. Furthermore, it is the basis of many other variants and extensions.

A first ILS algorithm for this problem has been proposed by Stützle [102]. It used the NEH heuristic [92] to generate an initial candidate solution. The NEH heuristic is a state-of-the-art constructive method for the PFSP with some recent variants being proposed only in 2014 [32]. It is a heuristic that builds a schedule by inserting at each step a next job in a position of the partial schedule where it least increases the objective function value. Using the speedup techniques proposed by Taillard [109], the NEH heuristic has a computational complexity of $O(n^2m)$. As an iterative improvement method in the ILS algorithm, the insert neighborhood was used, which exploits the same speedups as implemented for NEH. The insert neighborhood consists of all sequences that can be obtained by removing a job at position i and inserting it in all other possible positions; this results in a neighborhood of size $n \cdot (n-1)$. If one removes the job with index $\pi(i)$ from position i and inserts it at a position j, for $i < j$, one obtains $\pi' = (\pi(1), \ldots \pi(i-1), \pi(i+1), \ldots \pi(j), \pi(i), \pi(j+1), \ldots, \pi(n))$ and for $i > j$ one obtains $\pi' = (\pi(1), \ldots \pi(j-1), \pi(i), \pi(j), \ldots \pi(i-1), \pi(i+1), \ldots, \pi(n))$. The local search implemented in Stützle [102] used a kind of first-improvement neighborhood scan. It removes a job J_i from the sequence and checks all $n-1$ possible insertion positions; if several result in improvements, the position is taken that leads to the largest reduction in the makespan. One scan of the neighborhood consists in repetitions of this process for all possible positions i. The neighborhood scans are then repeated until a local optimum with respect to the insert neighborhood is found. As the perturbation, a mix of two contiguous swap moves, and one interchange move was proposed. In a contiguous swap move, a permutation $\pi = (\pi(1), \ldots, \pi(i), \pi(i+1), \ldots, \pi(n))$ is modified to $\pi' = (\pi(1), \ldots, \pi(i+1), \pi(i), \ldots, \pi(n))$; in an interchange-move $\pi = (\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n))$ is modified to $\pi' = (\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots, \pi(n))$. A further restriction was imposed on the positions involved in the interchange move by enforcing $|i-j| \leq \max\{n/5, 30\}$ to avoid too strong disruptions. Finally, as the acceptance criterion, the Metropolis condition with a fixed temperature was suggested.

The performance of this ILS algorithm was found to be superior to the best methods for the PFSP published until then. The excellent performance was confirmed in the extensive study of heuristics for the PFSP under makespan minimization by Ruiz and Maroto [98], where it was found to be the top performing heuristic.

Historical Development of Iterated Local Search

ILS has a long history, and the ideas underlying the method can be traced back to several approaches. The most influential have been the early developments for tackling the TSP. Baum [10, 11] has proposed an algorithm he called *iterated descent*, which used an iterative improvement method based on two-exchanges as the local search and random three-exchanges as the perturbation and forced the tour length to decrease. His results were not very impressive, but inspired Martin et al. [79] to propose their *large-stop Markov chains* (LSMC) algorithm. The name of this algorithm stems from the acceptance criterion, which corresponds to the Metropolis condition. In first instantiations, LSMC used a three-exchange neighborhood in the local search but was later extended to use the Lin-Kernighan heuristic. One contribution of the LSMC algorithm was the introduction of the double-bridge move as the perturbation, which is used in many other ILS algorithms for the TSP. The results with the LSMC algorithm gave a major leap in the performance of TSP heuristics. Further refinements of this approach, in particular, by Johnson [60] and Johnson and McGeoch [61] on one side and Applegate et al. [3] on the other side have defined for a long time the state-of-the-art TSP heuristics. Differences in these implementations concern, on the high-level ILS side, mainly the choices for the acceptance criteria and different choices of the perturbation operators, but decisive are also implementation details of the local search and the data structures used for tackling large instances. Noteworthy is the experimental study by Applegate et al. [4] who performed tests on very large TSP instances with up to 25 million cities and the more recent work of Merz and Huhse [84], which is particularly well suited for very large TSP instances under severe time constraints.

In a number of papers, specific choices for ILS algorithms for the TSP have been considered and analyzed. Codenotti et al. [21] study complex perturbation schemes based on data perturbation. Hong et al. [51] examined the impact the perturbation strength has on ILS performance, indicating that for some instances perturbations larger than those introduced by the double-bridge move are beneficial; they also studied population-based extensions of ILS algorithms for the TSP. Katayama and Narihisa [63] proposed to use candidate solutions different from the incumbent one to generate directed perturbations and showed that this idea can lead to very competitive results when compared to an iterated Lin-Kernighan algorithm. Stützle [103] and Stützle and Hoos [105] have analyzed the run-time behavior of ILS algorithms for the TSP, showing that the most common implementations show stagnation behavior for long runs. As a simple remedy to this behavior, they have proposed occasional restarts of the ILS algorithms [103,105]. Finally, there are some approaches for TSP solving that in their iterated version do not follow necessarily the main ILS steps but are, say, inspired from it. The most notable example is probably Helsgaun's iterated version of his Lin-Kernighan implementation [49]. It generates new starting candidate solution through a constructive mechanism, which is strongly biased toward generating a candidate solution that is close to the incumbent candidate solution.

The ILS principle has been discovered before its first applications to the TSP. Baxter [12] proposed an algorithm for a location problem that made repeated use of a data perturbation technique to create new starting solutions for a local search algorithm. However, it appears that these initial approaches have not been the main inspiring source for other ILS approaches and that this work has been overlooked for quite some time. In fact, most early ILS adaptations to problems other than the TSP seem to have been inspired by the proposals of ILS algorithms for the TSP.

Among the first problems other than TSP tackled by ILS is graph bi-partitioning. Martin and Otto [77, 78] reported for specific types of graphs better performance with their ILS algorithm than for simulated annealing heuristics. Many other early applications of ILS algorithms have been for scheduling problems. Lourenço [73] presented an innovative idea that combined ILS with exact algorithms for job-shop scheduling. In her approach, the exact algorithm is used to solve exactly subproblems on one or two machines, keeping the sequence on other machines fixed; the so modified candidate solution is then used as the perturbed one.

It is important to stress that ILS algorithms have been published and made popular under a variety of different names, ranging from problem or algorithm specific names such as iterated Lin-Kernighan [60,61] or chained Lin-Kernighan [4] to generic names such as large-step Markov chains [79], chained local optimization [78], iterated descent [10, 11], iterated hill-climbing [90], or others. In the local search template proposed by Vaessens et al. [114], ILS was referred to as a multilevel point-based local search. Even today, apparent variations of the ideas underlying ILS are published under new names such as breakout local search [13], creating possibly confusion with the earlier proposed breakout method [88], which relies on a different mechanism for escaping from local optima. A review and unification of the various papers published in the literature has been proposed by Lourenço et al. [74] in their 2002 book chapter on ILS. This chapter has led to a more coherent view of the method, has discussed the main ingredients of ILS algorithms, and has shown trade-offs in the design of effective ILS algorithms.

Relationship of ILS to . . .

ILS, being a relatively old method, has a number of links to other well-known methods, which we will shortly discuss in this section. We focus our discussion on relationships to simple, hybrid, and population-based metaheuristics (or stochastic local search methods), following the classification of Hoos and Stützle [53]. Within this classification, ILS belongs to hybrid SLS methods, which can be characterized as methods that interleave search steps in different neighborhoods or that combine different types of procedures in an interleaved fashion. As such, most links of ILS are within that group.

. . . Simple SLS Methods

Simple SLS methods typically move within one type of neighborhood. They can be seen as direct extensions of perturbative local searches that avoid local optima either by occasionally accepting moves to worse candidate solutions or modifications of the evaluation function during the search. Well-known examples for the former are simulated annealing [20, 64] and tabu search [40], while a well-known example for the latter is guided local search [118].

A first relation is that any of these methods may play the local search part within an ILS algorithm, replacing simpler iterative improvement methods. While iterative improvement methods do have a natural stopping condition, namely, hitting a local optimum, for simple local search methods, in principle, arbitrary computation times can be invested. This leads to the task of balancing the intensification obtained through the simple SLS method and the diversification given through perturbation steps and the acceptance criterion. A main task therefore for the algorithm designer is to determine an appropriate computation time for the SLS method to obtain a good trade-off between the computation time and solution quality.

Another relation is the usage of diversification measures in methods such as tabu search. For example, the random shake-up ideas [38] are directly linked to perturbations in ILS. The well-known reactive tabu search algorithms invoke occasional sequences of random moves to provide a means for search diversification [8], which can directly be seen as a random perturbation whose size is adjusted in dependence of features of the search process. Several ideas from the area of tabu search and, in particular, the memory usage explored there [40] may provide a source of inspiration of how to improve ILS algorithms or to define schemes for adapting parameters at run-time [9].

. . . Hybrid SLS Methods

Hybrid SLS methods combine more than one basic search strategy into an overall method. For example, ILS combines typically one search strategy in the local search with a different type of search strategy in the perturbation. From a high-level perspective, one point ILS has in common with many other hybrid SLS methods is that it combines a local optimization with search steps oriented toward diversification. These diversifying steps may be based on constructive or perturbative search steps.

Considering hybrid SLS methods that obtain diversification through constructive procedures, the closest to ILS is the iterated greedy algorithm, if the latter uses a local search phase. In iterated greedy methods, a new candidate solution is obtained by removing from a complete candidate solution $s \in S$ some solution components, obtaining a partial solution s_p, from which in turn a solution construction process is performed. While such iterated greedy algorithms may result in reasonable performance without improving the newly constructed candidate solution by a local search

phase, in many cases, an additional perturbative local search improves performance. In that latter case, the destruction–construction cycle may be seen as a (directed) perturbation in the ILS sense. Another link between the two methods is to see both as creating an iterative process that in the ILS case iterates across local search applications and in the iterated greedy case iterates across applications of constructive algorithms. A different perspective on the mechanism of iterated greedy algorithms is taken by large neighborhood search techniques [101]. They consider also the removal of solution components, but then the subsequent completion of the resulting partial solution may be done with exact techniques, originally taken from constraint programming, or also construcitve-type approaches. In any case, large neighborhood search interprets the completion of a partial solution as the exploration of a large neighborhood, hence the name of the method. In a sense, the application of one iteration of LNS would correspond in ILS terms to one (complex) perturbation that moves a solution $s \in S$ (typically a local optimum) to another complete candidate solution $s' \in S$. More details on iterated greedy and further related methods such as large neighborhood search can be found in the chapter on this topic by Stützle and Ruiz in this handbook. The use of constructive mechanisms to provide search diversification also underlies greedy randomized adaptive search procedures (GRASP) [31,96], which in their basic form create many independent starting points for a local search through randomized greedy constructive searches. GRASP is clearly distinct from the basic principle underlying ILS, as it does not create a biased walk in the space of local optima. Later extensions of GRASP add various mechanisms with the goal of making it more performing. In fact, some extensions reuse parts of candidate solutions and, hence, move GRASP more toward mechanisms that underlie ILS; see Resende and Ribeiro [96] for an overview of GRASP.

The hybrid SLS method that shares most similarities with ILS is (basic) variable neighborhood search (VNS) [45,46,87]. VNS is based on the principle of changing the neighborhood during the search to avoid getting trapped in local minima with respect to one specific neighborhood. While the rationale of the main search mechanism underlying VNS and ILS is rather different, many instantiations of VNS, in particular those related to basic, skewed, or general VNS, can be seen directly as specific instantiations of ILS algorithms. For example, in basic VNS, an iterative improvement process is performed in the smallest neighborhood. Larger neighborhoods are explored randomly, and this random exploration is interleaved with the iterative improvement search in the smallest neighborhood. This loop corresponds to the perturbation and local search steps in ILS algorithms, where the main specificity of VNS is to modify the strength of the perturbation following some fixed scheme in which the neighborhoods are explored: the scheme could be in an increasing or decreasing order and additionally taking into account step sizes. In early instantiations of a basic VNS, only candidate solutions that improve on the incumbent candidate solution after the local search phase are accepted, while in skewed VNS worse candidate solutions are accepted depending on how far the new candidate solution is from the incumbent one. Finally, general VNS is a variation of the basic VNS schemes, where the underlying local search can be a variable neighborhood descent algorithm.

. . . Population-Based SLS Methods

Population-based SLS methods are characterized by the use of a population of candidate solutions to drive the search. As such, they are clearly distinct from ILS, which is based on modifying a single search point. In addition, several population-based search methods such as evolutionary algorithms [7] and ant colony optimization [29] do not necessarily make use of local search algorithms. However, several links exist. In fact, many population-based algorithms can profit from the introduction of a local search phase as it was clearly shown for ant colony optimization [29], for scatter search [41], and for many evolutionary algorithms resulting in methods such as genetic (or evolutionary) local search [90, 112] or memetic algorithms [89]. ILS as well as the population-based methods can in that case be seen as sampling candidate solutions in the space of local optima.

ILS and memetic algorithms are linked by taking an extreme parameterization of the latter, in particular, using a memetic algorithms with a population size of one. In that case, the mutation operator takes over the role of the perturbation and the selection scheme used for population replacement the role of the acceptance criterion. The usage of only mutation to generate candidate solutions has been called parthenogenetic algorithm [60]. In fact, even when using a population size of more than one individual and excluding recombination operations, in the context of memetic algorithms good performance results are reported for some problems [83]; mutation-only candidate solution modifications are also often used in evolution strategies.

Considering the usage of populations as a convenient means of providing search space exploration, several authors have explored population-based variants of ILS algorithms [51, 103, 104, 111]. The approaches proposed by Hong et al. [51] and Stützle [103] maintain the usual perturbation scheme, applying perturbation at each iteration to a single candidate solution. The usage of time-varying distance bounds among the candidate solutions in the population as a specific diversity mechanism was explored by Stützle [104], resulting in high solution quality for the quadratic assignment problem. Thierens [111] uses the population to (i) perturb only solution components in which pairs of candidate solutions differ, similar to what was done in the genetic transformations proposed by Katayama and Narihisa [63], and (ii) to reduce the neighborhood size during the local search.

Applications

This section summarizes some noteworthy applications of ILS in addition to those mentioned in sections "Some Examples of Iterated Local Search Algorithms" and "Historical Development of Iterated Local Search". As the number of applications of ILS has strongly increased over the recent years, we do not give here an exhaustive list but mainly refer to recent publications for illustrating the progress of ILS. Here, we mention only papers where the authors have identified their algorithms explicitly as ILS algorithms, even though a larger number of published

algorithms would fit the framework of ILS without this being made explicit in the respective papers.

Iterated Local Search for Routing Problems

Among the first explicit ILS algorithms for vehicle routing problems (VRPs) are those proposed by Ibaraki et al. [58] and Hashimoto et al. [47]. In the first paper, the authors tackle the VRP with time windows, where a dynamic programming approach minimizes the penalties for time window violations. The ILS algorithm, which used various neighborhoods in the local search, was shown to reach high performance on instances with up to 1000 customers. Hashimoto et al. [47] tackle a VRP with time windows and time-dependent travel times and costs. They introduce various speedups and neighborhood restrictions and show the effectiveness of their algorithm compared to previous proposals in the literature. Vaz Penna et al. [117] consider a variant where the fleet of vehicles is heterogeneous, that is, it consists of vehicles with different characteristics such as capacities. Melo Silva et al. [82] consider a VRP where split deliveries are allowed, that is, a customer's demand may be satisfied by splitting deliveries across various vehicles or tours; they report very high performance for their algorithm improving a large number of benchmark instances. Palhazi Cuervo et al. [94] tackle the vehicle routing problems with backhauls, where in addition to the consumers who get delivered from the depot, there are also suppliers that send goods to the depot. The authors propose a structurally simple ILS algorithm, where the main ingredient is a local search algorithm that uses multiple neighborhoods and allows oscillations between feasible and infeasible candidate solutions. Michallet et al. [86] consider a periodic VRP with time windows under the consideration of malevolence acts, and the goal becomes spreading the repeated visits to customers across their time windows. A VRP with multiple, incompatible commodities and multiple trips per work day is considered by Cattaruzza et al. [19]. They propose an effective ILS algorithm for this problems, which is shown to perform better than previous approaches. Nguyen et al. [93] consider a two-echelon location-routing problem, where two types of trips arise. One type of trips serves from a main depot a number of subordinate depots, which have to be located appropriately, and a second type of trips delivers goods to customers from the subordinate depots. Vansteenwegen and Mateo [115] solve a cyclic inventory routing problem for a single vehicle. The goal is to minimize the costs of the distribution and the inventory costs at the customers. An efficient ILS algorithm for this problem was shown to outperform previous approaches for this problem. Laurent and Hao [68] considered a multiple depot vehicle scheduling problem, which arises in public transport. Related to routing problems is the team orienteering problem, which arises commonly as a problem faced by tourists when planning their trips. Vansteenwegen et al. [116] have tackled a variant of this problem, which considers time windows, with an effective ILS algorithm that could reach high solution quality in relatively short computation times and improve, for 31 benchmark instances, the best candidate solutions known at that time.

Iterated Local Search for Scheduling Problems

Scheduling problems have been among the first problems tackled by ILS algorithms, and by now a large number of high-performing ILS algorithms have been proposed for many variants of scheduling problems including single and parallel machine scheduling problems as well as scheduling problems of flow- and job-shop type.

The first applications of ILS to single- and parallel-machine scheduling problems are due to Brucker et al. [16, 17]; they have used two neighborhood structures that are nested, where the outer neighborhood essentially is used for perturbation. ILS algorithms have reached particularly excellent results for the well-known single-machine total weighted tardiness problem (SMTWTP), for which the iterated dynasearch algorithms by Congram et al. [22] and the extension thereof by Grosso et al. [43] are state-of-the-art algorithms. den Besten et al. [26] have applied an ILS with a variable neighborhood descent local search also to the SMTWTP. Later ILS algorithms have been applied to the SMTWTP with additional sequence-dependent setup times in two independent works, reaching high performance when compared to other heuristics [108, 120]. A recent application of ILS to the parallel-machine total weighted tardiness problem has been presented by Della Croce et al. [25]. They use generalized interchange moves, ideas from dynasearch, and new large-scale machine-based neighborhoods for the local search to improve over the current state of the art.

ILS algorithms have been proposed to tackle a variety of flow-shop scheduling problems, starting with the best-known variant that considers the minimization of the makespan. The first ILS algorithm for this problem [102] has already been described in section "Permutation Flow-Shop Scheduling Example". Some issues in the design of ILS algorithms for the permutation flow-shop problem have been considered by Juan et al. [62]. ILS algorithms have been adapted by other researchers to variants of the flow-shop scheduling problem and shown high performance for the flow-shop problem with flowtime objective [27]. Later, the authors of [27] further improve their ILS algorithm by including a multi-restart perturbation strategy, where the ILS is continued from the best of a set of perturbations obtained from a local optimum [28]. State-of-the-art results for the same flowtime objective are obtained by [95] with ILS variants, including versions that entail populations. ILS has been used to solve more complex variants of the flow-shop problem. Yang et al. [121] presented an ILS algorithm for a flow-shop variant with several stages in series, where at each stage a number of machines is available for processing the jobs. Ribas et al. [97] deal with the blocking flow-shop problem and propose an ILS procedure where the local search combines moves in different neighborhoods as does the perturbation step. M'Hallah [91] tackles the flow-shop scheduling problem with the objective of minimizing the earliness and tardiness, where due dates are distinct. An ILS algorithm is presented, which uses a variable neighborhood descent in the local search phase. Geiger [36] proposes the usage of an ILS algorithm for tackling the multi-objective flow-shop scheduling problem, obtaining promising solution quality despite the simplicity of the proposed approach. Finally, complex hybrid flexible flowline problems are studied by [113],

where ILS approaches are intermingled with other schemes to obtain high-quality candidate solutions for scheduling problems that are very close to real settings found at production floors.

As mentioned before, the first scheduling problem tackled by ILS was the well-known job-shop scheduling problem [73]. The ILS algorithm by Kreipl [66] for the total weighted tardiness job-shop scheduling problem reached high performance, being surpassed only quite some time later by an evolutionary algorithm that integrated an ILS algorithm as the local search operator [30]. An ILS algorithm is underlying the generic approach proposed by Mati et al. [81], which tackles job-shop problems with regular objectives, that is, objectives whose values increase along with an increase of the completion times of the jobs. The proposed ILS algorithm, which uses a perturbation with a size chosen uniformly at random and an acceptance criterion that accepts every new candidate solution, was shown to reach very high performance for a number of criteria, including weighted tardiness and weighted completion time.

Iterated Local Search for Other Problems

ILS algorithms have been applied to tackle many other problems, illustrating the wide range of possible applications of the ILS principle. Corte and Sörensen [24] use an ILS algorithm for designing a water distribution network, obtaining excellent computational results despite the fact that the proposed algorithm has a much simpler structure than many of the other proposed methods for the same task. Grosso et al. [44] propose an ILS algorithms for the maximin Latin hypercube design problem, which consists in assigning positions to n points in a k-dimensional grid such that no two points have a same coordinate and the distance between the closest pair of points is maximized. A possibility to improve the state estimation for the monitoring of power systems is to insert power measure units into the network at appropriate positions. Hurtgen and Maun [54] propose an ILS algorithm for minimizing the size of the configuration to reach full observability of the network. While virtually all ILS applications have been to deterministic problems, Grasas et al. [42] consider adaptations for ILS to integrate simulation in order to be able to tackle stochastic combinatorial optimization problems. The approach is illustrated with some example application of the proposed SimILS framework. The integration of information from lower bounding techniques into an ILS algorithm is considered by Buson et al. [18] for solving the fixed-charge transportation problem, which extends the transportation problem by the consideration of fixed costs for sending a flow from some origin to a destination. The reduced costs from the lower bounding are used to guide a restart phase of the algorithm. Lai and Hao [67] apply ILS to obtain high-quality candidate solutions to the maximally diverse grouping problem, where the goal is to partition the vertices of an edge-weighted and undirected complete graph under some constraints on the group size. Wolf and Merz [119] develop an ILS algorithm to find a symmetric connectivity topology with minimum power consumption in wireless ad hoc networks. Benchmark results on

large instances with up to 1000 nodes show that their algorithm outperforms other heuristics that were previously used to solve this problem. ILS has been applied to image analysis tasks and Cordón and Damas [23] applied it to image registration, obtaining promising initial results. Imamichi et al. [59] apply ILS as a method to improve the placement of irregular polygons in a rectangular surface with the goal of minimizing, for a fixed width, the length of the container so that no polygon overlaps with any other or protrudes the container. The proposed approach allows overlaps but tries to minimize them using nonlinear programming methods and swaps between polygons. Few papers have considered the adoption of the ideas underlying ILS for tackling continuous optimization problems. Kramer [65] has explored a simple continuous optimization algorithm, which embedded Powell's methods into an ILS for continuous optimization. Liao and Stützle [69] use ILS as one of the component algorithms in their hybrid, competition-based approach for continuous optimization. This approach decides in a preliminary competition phase which of two continuous optimizers, an ILS algorithm or IPOP-CMAES [6] to execute and then, in a second phase, continues with the execution of the winning algorithm. This approach was a winner of the CEC 2013 benchmark competition for real-parameter optimization.

Conclusions

In the initial phases of metaheuristic research, many efforts were dedicated toward developing and refining new metaheuristic methods and on studying their performance and behavior. As such, the focus was on rather pure applications of the respective methods to tackle computationally hard problems. Iterated local search contributed by relying on a clear principle that is easy to identify and that leads usually to high-performing algorithms. In addition, ILS algorithms are relatively intuitive to design and at the same time malleable. Therefore, iterated local search served often as a basis for a larger algorithm engineering effort if high-performing heuristics are desired. This approach was very successful, as shown by the various problems for which ILS algorithms have been or still are state of the art. In later research efforts in the metaheuristics area, often hybrid methods were proposed that mix in one algorithm concepts from different principles to derive effective algorithms [15, 76]. Even if this trend somehow blurs the differences behind the methods, it is nevertheless important to have methods that rely on clear principles and to have knowledge of their particular strengths, which can be exploited in the design of hybrid methods.

The research in heuristic search methodologies and metaheuristics, in particular, has now reached a mature state, and fundamentally new ideas appear more and more rarely. Within this context, ILS has a clear role as one of the main methods that offers an excellent trade-off between simplicity and flexibility. As future trends we expect on one side that the number of applications of ILS increases further, and that more complex problems and also non-combinatorial problems may be tackled by it. On the other side, as the development of effective heuristic algorithms is becoming

increasingly more streamlined, we expect that the clear principles underlying ILS support well a sound algorithm engineering effort. While traditionally such an algorithm engineering effort relied in part on the manual configuration and tuning of the algorithms, over the recent years, the advent of automatic algorithm configuration tools [52, 106] such as ParamILS [56], SMAC [57], or irace [71, 72] has helped to alleviate the algorithm designer from the manual algorithm design and parameter tuning tasks. In fact, we foresee that in the future, the development of effective ILS algorithms and, more in general, of any other metaheuristic algorithms will be strongly based on the exploitation of automatic algorithm configuration techniques.

Cross-References

- ▸ GRASP
- ▸ Iterated Greedy
- ▸ Memetic Algorithms
- ▸ Tabu Search
- ▸ Variable Neighborhood Search

Acknowledgments This work received support from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a senior research associate. Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds.

References

1. Aarts EHL, Lenstra JK (eds) (1997) Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester, UK
2. Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discret Appl Math 123(1–3):75–102
3. Applegate D, Bixby RE, Chvátal V, Cook WJ (1999) Finding tours in the TSP. Technical report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn
4. Applegate D, Cook WJ, Rohe A (2003) Chained Lin-Kernighan for large traveling salesman problems. INFORMS J Comput 15(1):82–92
5. Applegate D, Bixby RE, Chvátal V, Cook WJ, Espinoza D, Goycoolea M, Helsgaun K (2009) Certification of an optimal TSP tour through 85,900 cities. Oper Res Lett 37(1):11–15
6. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of CEC 2005. IEEE Press, Piscataway, pp 1769–1776
7. Bäck T, Fogel DB, Michalewicz Z (1997) Handbook of evolutionary computation. IOP Publishing Ltd., Bristol
8. Battiti R, Tecchiolli G (1994) The reactive tabu search. ORSA J Comput 6(2):126–140
9. Battiti R, Brunato M, Mascia F (2008) Reactive search and intelligent optimization. Operations research/computer science interfaces, vol 45. Springer, New York
10. Baum EB (1986) Iterated descent: a better algorithm for local search in combinatorial optimization problems, manuscript

11. Baum EB (1986) Towards practical "neural" computation for combinatorial optimization problems. In: Neural networks for computing. AIP conference proceedings, pp 53–64
12. Baxter J (1981) Local optima avoidance in depot location. J Oper Res Soc 32(9): 815–819
13. Benlic U, Hao JK (2013) Breakout local search for the quadratic assignment problem. Appl Math Comput 219(9):4800–4815
14. Bentley JL (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Comput 4(4):387–411
15. Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11(6):4135–4151
16. Brucker P, Hurink J, Werner F (1996) Improving local search heuristics for some scheduling problems—part I. Discret Appl Math 65(1–3):97–122
17. Brucker P, Hurink J, Werner F (1997) Improving local search heuristics for some scheduling problems—part II. Discret Appl Math 72(1–2):47–69
18. Buson E, Roberti R, Toth P (2014) A reduced-cost iterated local search heuristic for the fixed-charge transportation problem. Oper Res 62(5):1095–1106
19. Cattaruzza D, Absi N, Feillet D, Vigo D (2014) An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. Comput Oper Res 51:257–267
20. Cerný V (1985) A thermodynamical approach to the traveling salesman problem. J Optim Theory Appl 45(1):41–51
21. Codenotti B, Manzini G, Margara L, Resta G (1996) Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. INFORMS J Comput 8(2): 125–133
22. Congram RK, Potts CN, van de Velde S (2002) An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. INFORMS J Comput 14(1): 52–67
23. Cordón O, Damas S (2006) Image registration with iterated local search. J Heuristics 12(1–2): 73–94
24. Corte AD, Sörensen K (2016) An iterated local search algorithm for water distribution network design optimization. Networks 67(3):187–198
25. Della Croce F, Garaix T, Grosso A (2012) Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. Comput Oper Res 39(6):1213–1217
26. den Besten ML, Stützle T, Dorigo M (2001) Design of iterated local search algorithms: an example application to the single machine total weighted tardiness problem. In: Boers EJW et al (eds) Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2001. LNCS, vol 2037. Springer, Heidelberg, pp 441–452
27. Dong X, Huang H, Chen P (2009) An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Comput Oper Res 36(5):1664–1669
28. Dong X, Ping, Huang H, Nowak M (2013) A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. Comput Oper Res 40(2): 627–632
29. Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, MA
30. Essafi I, Mati Y, Dauzère-Pérés S (2008) A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Comput Oper Res 35(8):2599–2616
31. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–113
32. Fernandez-Viagas V, Framinan JM (2014) On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Comput Oper Res 45:60–67
33. Fernandez-Viagas V, Ruiz R, Framiñan JM (2017) A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. Eur J Oper Res 257(3):707–721. https://doi.org/10.1016/j.ejor.2016.09.055
34. Fischetti M, Monaci M, Salvagnin D (2012) Three ideas for the quadratic assignment problem. Oper Res 60(4):954–964

35. Framinan JM, Gupta JN, Leisten R (2004) A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. J Oper Res Soc 55(12): 1243–1255
36. Geiger MJ (2011) Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology. Comput Ind Eng 61(3):805–812
37. Gendreau M, Potvin JY (eds) (2010) Handbook of metaheuristics. International series in operations research & management science, vol 146, 2nd edn. Springer, New York
38. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549
39. Glover F, Kochenberger G (eds) (2002) Handbook of metaheuristics. Kluwer Academic Publishers, Norwell
40. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Boston
41. Glover F, Laguna M, Martí R (2002) Scatter search and path relinking: advances and applications. In: [38], pp 1–35
42. Grasas A, Juan AA, Lourenço HR (2016) SimILS: a simulation-based extension of the iterated local search metaheuristic for stochastic combinatorial optimization. J Simul 10(1):69–77
43. Grosso A, Della Croce F, Tadei R (2004) An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. Oper Res Lett 32(1):68–72
44. Grosso A, Jamali ARMJU, Locatelli M (2009) Finding maximin Latin hypercube designs by iterated local search heuristics. Eur J Oper Res 197(2):541–547
45. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130(3):449–467
46. Hansen P, Mladenović N, Brimberg J, Pérez JAM (2010) Variable neighborhood search. In: [36], pp 61–86
47. Hashimoto H, Yagiura M, Ibaraki T (2008) An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. Discret Optim 5(2):434–456
48. Hejazi SR, Saghafian S (2005) Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res 43(14):2895–2929
49. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126:106–130
50. Helsgaun K (2009) General *k*-opt submoves for the Lin-Kernighan TSP heuristic. Math Program Comput 1(2–3):119–163
51. Hong I, Kahng AB, Moon BR (1997) Improved large-step Markov chain variants for the symmetric TSP. J Heuristics 3(1):63–81
52. Hoos HH (2012) Automated algorithm configuration and parameter tuning. In: Hamadi Y, Monfroy E, Saubion F (eds) Autonomous search. Springer, Berlin, pp 37–71
53. Hoos HH, Stützle T (2005) Stochastic local search—foundations and applications. Morgan Kaufmann Publishers, San Francisco
54. Hurtgen M, Maun JC (2010) Optimal PMU placement using iterated local search. Int J Electr Power Energy Syst 32(8):857–860
55. Hussin MS, Stützle T (2009) Hierarchical iterated local search for the quadratic assignment problem. In: Blesa MJ et al (eds) Hybrid metaheuristics. LNCS, vol 5818. Springer, Heidelberg, pp 115–129
56. Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. J Artif Intell Res 36:267–306
57. Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello Coello CA (ed) LION 5. LNCS, vol 6683. Springer, Heidelberg, pp 507–523
58. Ibaraki T, Imahori S, Nonobe K, Sobue K, Uno T, Yagiura M (2008) An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. Discret Appl Math 156(11):2050–2069
59. Imamichi T, Yagiura M, Nagamochi H (2009) An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Discret Optim 6(4): 345–361

60. Johnson DS (1990) Local optimization and the traveling salesman problem. In: Paterson M (ed) 17th international colloquium on Automata, languages and programming. LNCS, vol 443. Springer, Heidelberg, pp 446–461
61. Johnson DS, McGeoch LA (1997) The traveling salesman problem: a case study in local optimization. In: Aarts EHL, Lenstra JK (eds) Local search in combinatorial optimization. John Wiley & Sons, Chichester, pp 215–310
62. Juan AA, Lourenço HR, Mateo M, Luo R, Castellà Q (2014) Using iterated local search for solving the flow-shop problem: parallelization, parametrization, and randomization issues. Int Trans Oper Res 21(1):103–126
63. Katayama K, Narihisa H (1999) Iterated local search approach using genetic transformation to the traveling salesman problem. In: Banzhaf W et al (eds) Proceedings of GECCO 1999, vol 1. Morgan Kaufmann Publishers, San Francisco, pp 321–328
64. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
65. Kramer O (2010) Iterated local search with Powell's method: a memetic algorithm for continuous global optimization. Memetic Comput 2(1):69–83
66. Kreipl S (2000) A large step random walk for minimizing total weighted tardiness in a job shop. J Sched 3(3):125–138
67. Lai X, Hao JK (2016) Iterated maxima search for the maximally diverse grouping problem. Eur J Oper Res 254(3):780–800
68. Laurent B, Hao JK (2009) Iterated local search for the multiple depot vehicle scheduling problem. Comput Ind Eng 57(1):277–286
69. Liao T, Stützle T (2013) Benchmark results for a simple hybrid algorithm on the CEC 2013 benchmark set for real-parameter optimization. In: Proceedings of CEC 2013. IEEE Press, Piscataway, pp 1938–1944
70. Lin S, Kernighan B (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21(2):498–516
71. López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package, iterated race for automatic algorithm configuration. Technical report, TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles. http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
72. López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspectives 3: 43–58
73. Lourenço HR (1995) Job-shop scheduling: computational study of local search and large-step optimization methods. Eur J Oper Res 83(2):347–364
74. Lourenço HR, Martin O, Stützle T (2002) Iterated local search. In: [38], pp 321–353
75. Lourenço HR, Martin O, Stützle T (2010) Iterated local search: framework and applications. In: [36], chap 9, pp 363–397
76. Maniezzo V, Stützle T, Voß S (eds) (2009) Matheuristics—hybridizing metaheuristics and mathematical programming. Annals of information systems, vol 10. Springer, New York
77. Martin O, Otto SW (1995) Partitioning of unstructured meshes for load balancing. Concurr Pract Exp 7(4):303–314
78. Martin O, Otto SW (1996) Combining simulated annealing with local search heuristics. Ann Oper Res 63:57–75
79. Martin O, Otto SW, Felten EW (1991) Large-step Markov chains for the traveling salesman problem. Complex Syst 5(3):299–326
80. Martin O, Otto SW, Felten EW (1992) Large-step Markov chains for the TSP incorporating local search heuristics. Oper Res Lett 11(4):219–224
81. Mati Y, Dauzère-Pèrés S, Lahlou C (2011) A general approach for optimizing regular criteria in the job-shop scheduling problem. Eur J Oper Res 212(1):33–42
82. Melo Silva M, Subramanian A, Ochi LS (2015) An iterated local search heuristic for the split delivery vehicle routing problem. Comput Oper Res 53:234–249
83. Merz P, Freisleben B (2000) Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Trans Evol Comput 4(4):337–352

84. Merz P, Huhse J (2008) An iterated local search approach for finding provably good solutions for very large TSP instances. In: Rudolph G et al (eds) PPSN X. LNCS, vol 5199. Springer, Heidelberg, pp 929–939
85. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller A, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21:1087–1092
86. Michallet J, Prins C, Yalaoui F, Vitry G (2014) Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. Comput Oper Res 41:196–207
87. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
88. Morris P (1993) The breakout method for escaping from local minima. In: Proceedings of the 11th National Conference on Artificial Intelligence. AAAI Press/MIT Press, Menlo Park, pp 40–45
89. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 219–234
90. Mühlenbein H (1991) Evolution in time and space—the parallel genetic algorithm. In: Rawlins G (ed) Foundations of genetic algorithms (FOGA). Morgan Kaufmann Publishers, San Mateo, pp 316–337
91. M'Hallah R (2014) An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. Int J Prod Res 52(13):3802–3819
92. Nawaz M, Enscore E Jr, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11(1):91–95
93. Nguyen VP, Prins C, Prodhon C (2012) A multi-start iterated local search with tabu list and path relinking for the two-echelon location-routing problem. Eng Appl Artif Intell 25(1): 56–71
94. Palhazi Cuervo D, Goos P, Sörensen K, Arráiz E (2014) An iterated local search algorithm for the vehicle routing problem with Backhauls. Eur J Oper Res 237(2):454–464
95. Pan QK, Ruiz R (2012) Local search methods for the flowshop scheduling problem with flowtime minimization. Eur J Oper Res 222(1):31–43
96. Resende MGC, Ribeiro CC (2010) Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In: [36], pp 283–319
97. Ribas I, Companys R, Tort-Martorell X (2013) An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. Int J Prod Res 51(17):5238–5252
98. Ruiz R, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165(2):479–494
99. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177(3):2033–2049
100. Schreiber GR, Martin O (1999) Cut size statistics of graph bisection heuristics. SIAM J Optim 10(1):231–251
101. Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget JF (eds) CP'98. LNCS, vol 1520. Springer, Heidelberg, pp 417–431
102. Stützle T (1998) Applying iterated local search to the permutation flow shop problem. Technical report AIDA–98–04, FG Intellektik, FB Informatik, TU Darmstadt
103. Stützle T (1998) Local search algorithms for combinatorial problems—analysis, improvements, and new applications. PhD thesis, FB Informatik, TU Darmstadt
104. Stützle T (2006) Iterated local search for the quadratic assignment problem. Eur J Oper Res 174(3):1519–1539
105. Stützle T, Hoos HH (2001) Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In: Hansen P, Ribeiro C (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Boston, pp 589–611
106. Stützle T, López-Ibáñez M (2015) Automatic (offline) configuration of algorithms. In: Laredo JLJ, Silva S, Esparcia-Alcázar AI (eds) GECCO (Companion). ACM Press, New York, pp 681–702

107. Stützle T, Birattari M, Hoos HH (eds) (2007). SLS 2007. LNCS, vol 4638. Springer, Heidelberg
108. Subramanian A, Battarra M, Potts CN (2014) An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. Int J Prod Res 52(9):2729–2742
109. Taillard ÉD (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47(1):65–74
110. Taillard ÉD (1995) Comparison of iterative searches for the quadratic assignment problem. Locat Sci 3(2):87–105
111. Thierens D (2004) Population-based iterated local search: restricting the neighborhood search by crossover. In: Deb K et al (eds) GECCO 2004, part II. LNCS, vol 3103. Springer, Heidelberg, pp 234–245
112. Ulder NLJ, Aarts EHL, Bandelt HJ, van Laarhoven PJM, Pesch E (1991) Genetic local search algorithms for the travelling salesman problem. In: Schwefel HP, Männer R (eds) Proceedings of PPSN-I. LNCS, vol 496. Springer, Heidelberg, pp 109–116
113. Urlings T, Ruiz R, Stützle T (2010) Shifting representation search for hybrid flexible flowline problems. Eur J Oper Res 207(2):1086–1095
114. Vaessens RJM, Aarts EHL, Lenstra JK (1998) A local search template. Comput Oper Res 25(11):969–979
115. Vansteenwegen P, Mateo M (2014) An iterated local search algorithm for the single-vehicle cyclic inventory routing problem. Eur J Oper Res 237(3):802–813
116. Vansteenwegen P, Souffriau W, Berghe GV, Oudheusden DV (2009) Iterated local search for the team orienteering problem with time windows. Comput Oper Res 36(12):3281–3290
117. Vaz Penna PH, Subramanian A, Ochi LS (2013) An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. J Heuristics 19(2):201–232
118. Voudouris C, Tsang EPK (2002) Guided local search. In: [38], pp 185–218
119. Wolf S, Merz P (2009) Iterated local search for minimum power symmetric connectivity in wireless networks. In: Cotta C, Cowling P (eds) Proceedings of EvoCOP 2009. LNCS, vol 5482. Springer, Heidelberg, pp 192–203
120. Xu H, Lü Z, Cheng TCE (2014) Iterated local search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. J Sched 17(3): 271–287
121. Yang Y, Kreipl S, Pinedo M (2000) Heuristics for minimizing total weighted tardiness in flexible flow shops. J Sched 3(2):89–108

Memetic Algorithms

20

Carlos Cotta, Luke Mathieson, and Pablo Moscato

Contents

Introduction 608
Structure of a Memetic Algorithm 610
Skeleton of a Classical Memetic Algorithm 611
A Note on More Complicated Memetic Algorithms 614
Memetic Algorithms in Practice 616
An Example Memetic Algorithm for NETWORK ALIGNMENT 616
An Example Memetic Algorithm for WEIGHTED CONSTRAINT SATISFACTION PROBLEMS 617
A Brief Survey of Recent Memetic Algorithm Applications 620
Future-Generation Memetic Algorithms 622
Conclusion 623
Cross-References 624
References 624

Abstract

Memetic algorithms provide one of the most effective and flexible metaheuristic approaches for tackling hard optimization problems. Memetic algorithms address

C. Cotta (✉)
Departamento Lenguajes y Ciencias de la Computación, Universidad de Málaga, Málaga, Spain
e-mail: ccottap@lcc.uma.es

L. Mathieson
Centre for Bioinformatics, Biomarker Discovery and Information-Based Medicine, University of Newcastle, Callaghan, NSW, Australia
e-mail: luke.mathieson@newcastle.edu.au

P. Moscato
School of Electrical Engineering and Computing, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, Australia
e-mail: pablo.moscato@newcastle.edu.au

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_29

the difficulty of developing high-performance universal heuristics by encouraging the exploitation of multiple heuristics acting in concert, making use of all available sources of information for a problem. This approach has resulted in a rich arsenal of heuristic algorithms and metaheuristic frameworks for many problems. This chapter discusses the philosophy of the memetic paradigm, lays out the structure of a memetic algorithm, develops several example algorithms, surveys recent work in the field, and discusses the possible future directions of memetic algorithms.

Keywords
Evolutionary algorithms · Hybridization · Local search · Memetic computing · Metaheuristics

Introduction

The effectiveness and efficiency of (meta)heuristics – and memetic algorithms which may be viewed as particularly good heuristics in this sense – rests upon their ability to explore the solution space thoroughly while avoiding exhaustive or near-exhaustive searching. If polynomial-time computability is taken as an approximation of tractability, then a polynomial-time algorithm can be viewed as a very clever search procedure; in these cases there is a small search space, or the search space can be reduced drastically. In dealing with intractable problems however, reducing the search space to a reasonable size is a much more difficult task. Most central is the problem of local versus global improvement; an improvement to a solution does not give any guarantee of movement toward the optimum. Actually, depending upon the problem under consideration, a local improvement may be a move *away* from the global optimum (hence the notion of deception in search algorithms [265]), or at the very least the solution may be getting closer to being trapped in a nonoptimal configuration for which no simple modification can lead to an improvement (i.e., a local optimum).

Thus, many (meta)heuristic methods include techniques for allowing non-improving alterations to a solution or for nonlocal moves across the search space in order to be able to escape from local optima [28]. Perhaps the most archetypical example of such a metaheuristic is the genetic algorithm (GA) [99, 110]: inspired by the principles of natural evolution, GAs maintain a population (i.e., a multiset) of solutions that are subject to successive phases of selection, reproduction (via recombination and mutation), and replacement. The use of a population of solutions provides a better chance of avoiding local optima than maintaining a single solution: on one hand, the search is driven by operators that (1) allow the search to take non-improving steps, most notably in the case of mutations, and (2) allow the search to move to significantly different portions of the search space, particularly by virtue of recombination. On the other hand, selection and replacement typically work on a global (population-wise) scale, meaning that non-improving solutions have a chance of persisting for a nontrivial amount of time, hence allowing escape from local optima. However, although this heuristic structure has proven quite effective, it

relies almost entirely upon recombination mechanisms to improve solution quality, and evolutionary processes are slow. In particular, they are also less capable of fine-tuning solutions, that is, the progress toward a fully optimized solution once the algorithm has located its basin of attraction (i.e., the region of the search space from which a series of small – local – improvements can lead to a certain local optimum; see [131,224]) is often sluggish. This is precisely in contrast to local search (single-solution or trajectory-based) techniques which can readily locate local optima (and hence are more sensitive to them).

To address this weakness, researchers began developing *hybridized* metaheuristics [29, 31], that is, metaheuristics which combine ideas from different search paradigms and/or different algorithms altogether. The underlying idea in such approaches is obviously trying to achieve some synergetic behavior whereby the deficiencies of a certain search technique are compensated by the combination with other techniques and their advantages are boosted due to this very same combination. This strict interpretation of the term *hybrid* has been broadened with time to encompass all forms of non-blind (i.e., not domain-independent) metaheuristics. Under this broad interpretation, hybridization is the process of augmenting a generic (problem-independent) metaheuristic with problem (or problem class, i.e., domain) knowledge. Since this augmentation is often achieved via the blend of different metaheuristic components, both interpretations are equivalent in most situations. The broad interpretation has, in any case, the advantage of fitting better into theoretical results such as those of Hart and Belew [107] and – most conspicuously – those of Wolpert and Macready [267] in the so-called no-free-lunch theorem, which states that search algorithms perform strictly in accordance with the amount and quality of the problem knowledge they incorporate. While these results spurred controversy in their time and have been refined [69,70], the bottom line still holds.

Memetic algorithms (MAs) championed this philosophy. The denomination *memetic algorithm* was coined in [175] to characterize and codify these hybrid approaches. The term "memetic" was developed from Dawkin's [62] notion of a "meme" (from the Ancient Greek μίμημα, meaning "imitated thing") as a unit of cultural inheritance (and hence cultural evolution) – the cultural analogue of a gene. The use of the term meme was intended to capture the fact that information traits in human culture are subject to periods of lifetime learning and therefore they are different when transmitted to what they were when first acquired. This bears a strong resemblance with the Lamarckian model of evolution, whereby traits acquired during the lifetime of an individual are transmitted to its offspring. It is therefore not surprising that MAs are sometimes disguised under other denominations featuring the terms "Lamarckian" or "hybrid."

While the initial conception of memetic search did not include the idea of GAs or evolutionary algorithms (EAs) whatsoever [178], it turned out that these techniques were ideal recipients for exploiting the metaphor of MAs, namely, having a collection of "agents" alternating periods of self-improvement with phases of cooperation and competition, cf. [177]. Indeed, early MAs mixed GAs and EAs with simulated annealing and tabu search [180, 198], eventually developing the idea that MAs are EAs endowed with some kind of local search (LS) technique, leading to the restrictive definition MA = EA + LS [218]. Note however that the

Algorithm 1: A generic memetic algorithm

```
1:  parfor j := 1 to μ do                    ▷ Initialise population Pop of search agents
2:      pop_i := new SearchAgent
3:  end parfor
4:  repeat
5:      parfor j := 1 to μ do                ▷ individual learning phase
6:          pop_i.learn()
7:      end parfor
8:      Pop.cooperate()                      ▷ cooperation phase
9:      Pop.compete()                        ▷ competition phase
10: until Termination condition is true.
```

central concept of MAs is not to tie ourselves to a particular heuristic approach or metaphor, but to provide a coherent structure for employing *several* heuristics (including exact methods [30]) that deploy *complementary* heuristics exploiting *all available knowledge*. Thus EA + LS $\subset$ MA is a consequence of this broader definition of MAs, cf. [50]. The next section will explore in more detail the structure of an MA with particular emphasis on the classical characterization of the paradigm.

Structure of a Memetic Algorithm

As mentioned above, early definitions of MAs envisioned the paradigm as a pragmatic integration of ideas from different metaheuristics. These were orchestrated in terms of a collection of search agents carrying out individual explorations (i.e., lifetime learning) of the space of solutions and engaging in periodic phases of cooperation and competition [198]. An abstract formulation of such an approach is provided in Algorithm 1. This pseudocode matches the initial conception of MAs as an inherently parallel approach whereby a collection of local searchers (simulated annealing in early developments [178]) run either concurrently or physically in parallel and establish synchronization points in which information was exchanged among them. This said, this depiction of MAs is still generic enough to encompass most actual incarnations of the paradigm as shown later, as it captures the essential feature of MAs, namely, the carefully crafted interplay between global (population-based) and local (individual-based) search. It must be noted that the terms *global* and *local* are used in connection to the mechanics of the search rather than to the ability of eventually (or asymptotically) finding the global optimum. It is certainly the case that many local search approaches (simulated annealing, tabu search, etc.) are capable of escaping from local optima and navigate the search space in order to find the global optimum. The distinctive feature of these techniques (as opposed to, e.g., genetic algorithms) is that they do this following a trajectory-based approach.

Skeleton of a Classical Memetic Algorithm

Following early works in which the population-based aspects of MAs, namely, the collection of agents and the synchronized stages of cooperation and competition, were captured by a genetic algorithm [180], the classical memetic model coalesced. The basic skeleton of such an MA is relatively straightforward, adding little additional complexity beyond that of a GA. Algorithm 2 gives a pseudocode sketch of the salient structure, using *local search* as a placeholder for any particular individual improvement heuristic including, for instance, a complete exact algorithm like branch and bound, and others that guarantee optimality of the final solution obtained when they stop. Although a small structural change to a typical GA, the inclusion of the individual improvement phase can dramatically alter its performance. This mix allows the metaheuristic to benefit from the solution diversity engendered by the evolutionary approach, but to avoid the lethargic pace of improvement via more directed optimization: instead of relying random processes subjected to fitness-based selection alone, each individual solution is optimized before the evolutionary mechanism is applied, significantly increasing the rate at which individual solutions converge to an optima.

Algorithm 2: A local-search based memetic algorithm

```
1:  Pop := new Solution [popsize]                          ▷ Create population Pop
2:  for i ∈ Pop do
3:      i.initialise()                                     ▷ Generate initial solution
4:      i.local-search()  ▷ Individual improvement. Comprises solution evaluation
5:  end for
6:  repeat
7:      for j := 1 to #recombinations do                   ▷ Recombination phase
8:          Parents := Pop.select(numparents)              ▷ Select parent set
    Parents ⊆ Pop
9:          c := Parents.recombine()      ▷ Recombine parents to create child c
10:         c.local-search()
11:         Pop.update(c)                 ▷ Inserts new solution in the population
12:     end for
13:     for j := 1 to #mutations do                        ▷ Mutation phase
14:         i := Pop.select(1);
15:         i_m := i.mutate()                              ▷ Mutate solution i
16:         i_m.local-search()
17:         Pop.update(i_m)
18:     end for
19:     if Pop.converged() then          ▷ Refresh population upon convergence
20:         Pop.restart()
21:     end if
22: until Termination condition is true.
```

The structure of an MA is quite flexible, and the performance of the implementation, both in terms of solution quality and speed, can be affected by a number of factors. As an evolutionary, population-based metaheuristic, the typical issues regarding choice and implementation of mutation and recombination operators are inherited from the GA paradigm. For those familiar with GAs however, it should be readily apparent that the individual improvement phase is most likely to be the computational bottleneck – the improvement of every individual and the subsequence evaluation of every individual are inherently expensive simply because they are done for every individual (as shown in [167], it can easily take up to 95% of the computational cost of the algorithm). The tradeoff is that with a good choice of individual improvement heuristic, far fewer generations of mutation and recombination are required. The careful reader will also notice that the local search procedure (and, typically, any individual improvement heuristic) is highly amenable to parallelization. This helps to ameliorate the cost of the individual improvement, but more importantly lends the MA approach a high degree of scalability.

From the point of view of the different components into which a classical MA can be dissected, all of which encapsulate some portion of problem knowledge. Consider, for instance, recombination. This is the component that captures most appropriately the idea of agent cooperation. Such a cooperation is typically established between a pair of agents but can in general involve an arbitrary number of *parents* [76] (notice nevertheless that in this case some forms of heuristic recombination can be very complex [53]). The generic idea of a knowledge-augmented recombination operator is to combine, in an intelligent way, pieces of information contained in the parents. How these pieces are defined is a problem-dependent issue that arises from the issue of representation in EAs. The underlying objective of an appropriate representation would be to have solutions described by some structured collection of objects whose values truly capture solution features of relevance (i.e., ultimately responsible for determining whether a solution is good or not). Even from the beginnings of MAs, the importance of developing a suitable representation – in which evolution of the representation reflects the correlation of elements in the fitness landscape – was identified [175]. This is a substantial topic for which the interested reader is referred to, e.g., [226]. Focusing on the *smart* manipulation of these information units (however they are defined), that is, processing them in a problem-specific way instead of using domain-independent templates (such as those in [217]), the goal is picking the right combination of such units from either parent. Of course, this is easier said than done (and in fact, doing it is in general *provably* hard for arbitrary problems and/or definitions of *right* combination – see the discussion on the polynomial merger complexity class in [177, 179]), but there are numerous heuristic ideas in the literature that can be used to this end. In many cases – and following design advice already present in classical texts of hybridization pioneers, e.g., [61] – these ideas are based on the use of problem-specific heuristics such as greedy algorithms [132,185], backtracking [48], dynamic programming [123], or branch and bound [55], just to mention a few.

Mutation is another classical operator, well known for its role of maintaining a continuous supply of genetic diversity that can be subsequently exploited by the

remaining operators. In certain EA models, such as evolutionary programming [86], it actually bears sole responsibility for driving the search. Note that while this latter philosophy can be also used in an MA context – see section "An Example Memetic Algorithm for WEIGHTED CONSTRAINT SATISFACTION PROBLEMS" – it is typically the case that MAs use sophisticated recombination operators such as those described before. Thus, the criticism of recombination being just a disguised form of macromutation would not apply to them. Moreover, due to the presence of a local search stage in the main evolutionary cycle, one has to be careful to pick a mutation operator whose effects on solutions cannot be trivially undone by the local search, since that would defeat the very purpose of mutation. Following this line, in some cases there are MAs that even refrain from using mutation, e.g., [163, 262]. While such a decision could be further vindicated by the fact that MAs usually feature a population-restart procedure (see line 20 in Algorithm 2) and hence premature convergence is not so troublesome, this is not the most common course of action. An appropriate mutation operator (i.e., one using a sufficiently different neighborhood to that used by the local searcher) is often utilized. In fact, it is not unusual to have more than one such mutation operator, e.g., [152, 231], much like in metaheuristic approaches such as variable neighborhood search [105] (VNS). In some cases, these multiple mutation operators are used with the purpose of exerting different degrees of perturbation (i.e., *light* and *heavy* mutations) depending on the convergence of the population [87].

As to the local search component, it can take the form of any stand-alone method such as hill climbing, simulated annealing, tabu search, variable neighborhood search, etc. [199]. The choice of a particular technique must take into account two major issues, namely, its parameterization and its interaction with the remaining components of the algorithm. Regarding the latter, and in addition to the issues discussed above in connection to the mutation operator, one has to consider the interplay between the local searcher and the recombination operator. For example, a highly intensive local search procedure may be better suited to interact with a more diversification-oriented recombination operator – see, e.g., [88]. This heuristic recipe does not necessarily conflict with the use of a powerful recombination operator (see, e.g., [91]) but underlines that the knowledge embedded in either component, recombination operator and local search heuristic, must be complementary in terms of the effect they produce in the search dynamics (much as was discussed for the mutation operator). An interesting analysis of these issues from the point of view of fitness landscapes is provided in [170]. Whatever the definition of the neighborhood is (and notice that it can be complex, even combining several simpler neighborhood schemes), it is often crucial to be able to evaluate solutions incrementally for performance reasons [106]. It is desirable to avoid having to resort to a full evaluation and only recompute the fitness contribution of the solution components that were modified. This may require the use of appropriate data structures and is normally associated to discrete optimization (the high nonlinearity – and sometimes even the lack of a closed fitness function – often makes this complicated in continuous optimization).

The parameterization of the local search heuristic is another complex issue. This includes both high-level algorithmic aspects as well as low-level parameters. The high-level aspects include factors such as when to apply local search, to which solutions it should be applied and which local search operator to apply. The low-level aspects include parameters such as the breadth (number of neighbors explored in each iteration of the local search heuristic) and depth (how many iterations of local search will be performed). Further discussion is given in [245]. Determining an adequate setting for these parameters is crucial for the performance of the algorithm since it has been shown theoretically that small parameter changes can turn a problem from being polynomial-time solvable with high probability to requiring super-polynomial (even exponential) time [144, 244]. Unfortunately, a priori design guidelines to provably avoid this kind of behavior are ruled out by intractability results [245]. Thus, design by analogy and empirical testing seem to be the handiest tools to approach this endeavor (although self-parameterization is an appealing alternative that is increasingly gaining relevance – see section "Future-Generation Memetic Algorithms"). In this regard, it has been, for example, shown in several contexts that partial Lamarckism [112], that is, not applying local search to every individual but just applying it some probability p_{LS}, can produce notably better results than a fully Lamarckian approach [49, 126] although the best value of this parameter is problem dependent. On a related note with regard to the depth of the local search, it has been also proposed in the literature to save the store of the local search together with the solution it was applied to, so as to resume the process from that point if required [173, 174].

The restarting procedure is another important element in an MA. The goal of this procedure is to perform a warm reinitialization of the population when the search is deemed stagnated (i.e., the population has converged to a suboptimal state). Of course, that stagnation can be hindered by taking preventive measures such as the light/heavy mutation scheme mentioned before, the use of spatial structure in populations [250] (see also next subsection), or some other diversity-preservation policy [236] – see also [188]. A more drastic measure may be eventually required though. For that purpose, a common approach is to keep a certain percentage of the current population and use the solution creation mechanism (the one used to create the initial population – line 3 in Algorithm 2) to complete the new population. Regarding the former, they constitute a seed that allows keeping a part of the search momentum without having to start from scratch. As to the latter, notice that they need not be purely random solutions but any available constructive heuristic can be used for this purpose.

A Note on More Complicated Memetic Algorithms

Although Algorithms 1 and 2 lay out a basic MA framework, the structure can be made significantly more complex. As the central motivation of MAs is to exploit

all available information, the restriction of any particular component would be antithetical. Apart from employing different heuristics, *multiple* heuristics can be employed in concert. It is easy to combine different individual improvement heuristics, applying them to different individuals and different populations, in parallel, in sequence, or even in competition. Similarly the population-based heuristic can employ multiple improvement techniques – such approaches are well known in the GA community.

Moscato and Tinetti [181] demonstrate a more complicated MA that uses a number of heuristics in concert and to achieve different goals within the algorithm. The algorithm employs a *tree-structured population* where the population is divided into subpopulations of size 4, composed in a ternary tree structure:

1. Each subpopulation is divided into a *leader* node and three *supporters*. The supporters are stored one level below their leader.
2. The intermediate nodes in the tree hold an individual that is part of two populations; it is the leader of the three supporters lower in the tree and a support of its leader higher in the tree.
3. The number of subpopulations can be manipulated by adding levels to the tree.

Each individual can be optimized using a local search procedure that selects from a variety of local optimization moves: approximate 2-OPT, One-City Insertion, and Two-City Insertion [134,156]. Genetic recombination occurs "normally" within each subpopulation. The leader individual represents the best tour in the subpopulation. Note that the overlap of subpopulations ensures improvement propagates up the tree. The small subpopulation size, however, can quickly lead to a lack of diversity, in which case the recombination mechanism switches to an external recombination procedure for the lowest level of the tree.

In this example a number of variations on the basic structure of an MA are evident: multiple local search variants, a multipopulation variation of a GA which itself employs multiple recombination procedures. Another example of a more complex improvement strategy is given by Moscato [176], where a small population of individuals is maintained (only 16 individuals, each a binary vector), with a tabu search procedure for individual improvement. Again, in a small population, a loss of diversity is a potential drawback. To combat this, each individual notes the 16 best single-bit-flip moves available. When diversity falls below a given threshold, instead of following a simple tabu search approach, individual i makes move i from its list of best moves. This deliberate (potentially) nonoptimal has the effect of spreading the individuals further across the configuration space, increasing diversity. Once diversity is restored, the normal tabu search optimization is restored.

The continuation of these ideas has led to the development of what are now called *self-adaptive* memetic algorithms, which allow the context-specific, dynamic application of different heuristics or tuning of search parameters by the algorithm itself. See section "Future-Generation Memetic Algorithms".

Memetic Algorithms in Practice

This section presents two extended examples of memetic algorithms for specific problems – NETWORK ALIGNMENT and WEIGHTED CONSTRAINT SATISFACTION PROBLEMS – and surveys recent interesting applications of memetic algorithms in different fields.

An Example Memetic Algorithm for NETWORK ALIGNMENT

The optimization version of the basic NETWORK ALIGNMENT problem takes as input two networks G_1 and G_2 and asks for an injective partial mapping $f : V(G_1) \rightarrow V(G_2)$ between the vertices of the two networks that maximize $\sum_{u,v \in V(G_1)} \tau_f(u, v)$ where

$$\tau_f = \begin{cases} 1 \text{ if } uv \in E(G_1) \text{ and } f(u)f(v) \in E(G_2) \\ 0 \text{ otherwise} \end{cases}$$

It may be assumed that the mapping is total and bijective by adding "dummy" vertices to the smaller network. Of course τ_f is open to variation as are the precise details of f, leading to many variants of NETWORK ALIGNMENT. The decision variant of NETWORK ALIGNMENT is NP-complete [138] and W[1]-complete (Mathieson et al., 2015, Using network alignment to uncover topological structure and identify consumer behaviour modelling constructs, unpublished Manuscript), suggesting, subject to standard complexity assumptions, that no suitably efficient exact algorithm for NETWORK ALIGNMENT exists, making it a prime candidate for heuristic methods.

In developing a memetic algorithm for this (and any) problem, it is necessary (at a minimum) to select an individual solution representation, mutation, and recombination operators and an individual improvement heuristic and its attending concerns.

For NETWORK ALIGNMENT, the most direct individual representation is the mapping itself. Assuming any necessary dummy vertices have already been added, the mapping can be represented by, for example, an array of size $|V(G_1)|$ storing a permutation of $V(G_2)$. For ease of representation, it is sufficient to assign each vertex a unique integer in the range $[0, |V(G_1)|]$. An alternative representation suitable for NETWORK ALIGNMENT would be to store the alignment of the edges, making computing the basic fitness function simple, but the individual could be polynomially larger, impacting the efficiency of mutation, recombination, individual improvement, and even evaluation. Moreover care would need to be taken as to how to determine which vertices were aligned.

With this individual representation, a simple, reasonable mutation operator is that of a random shuffle, where each element of an individual is randomly swapped with another, randomly chosen element, with a given probability. A naïve recombination

operator is, given two parent individuals, to select a linear segment of the individual (i.e., a set of contiguous indices) and swap the mappings for those indices between the parents (with adjustment to take care of duplication of elements), producing two children. This recombination is commonly known as a partially matched crossover [100]. However, considering the problem at hand, it is easy to see that this choice may be somewhat inefficient. The NETWORK ALIGNMENT problem, in essence, seeks to preserve as much topological structure (i.e., edge matchings) as possible – in this sense it is a relaxed GRAPH ISOMORPHISM problem. Swapping a set of arbitrarily chosen indices is unlikely to preserve interesting structure, contrary to the goal of a recombination operator, which is to produce children of higher quality than their parents by mixing the better parts of the parents, aiming to place the child solution closer to the global optima. For NETWORK ALIGNMENT, it is much more interesting to preserve neighborhoods of vertices in this regard. So a better choice of recombination operator is to select a vertex and its 2-neighborhood (all vertices at distance at most 2) as the set of indices which will be swapped.

To complete the GA component, a tournament selection process is employed to choose the individuals included in the new generation and a restart mechanism whereby the best solution is recorded and the population is restarted if no improvement has been observed after a given number of generations.

For individual improvement, a local search heuristic is used, where the neighborhood of each individual is the 2-swap neighborhood – the set of individuals obtained by swapping any two elements. The search is implemented by selecting an element in the individual and taking the optimal swap in the local neighborhood. If this is not the identity mapping, the neighbors of the preimage of the swapped vertex are placed into a list of vertices to swap. If no initial swap is found, the process is repeated with a new starting point until a swap is found or all vertices have been tested.

In combination with the skeletons given by Algorithms 1 and 2, these components constitute an MA for NETWORK ALIGNMENT. The reader will notice that, even without considering more complicated approaches, there are a number of tunable parameters present. These include the probabilities and frequencies which control mutation and recombination (as in GAs) and, more specifically for MAs, the frequency and application régime of the individual improvement step. The individual improvement may be applied, at essentially one extreme, regularly, to all individuals, or at the other extreme, only when the evolutionary progress slows and to a select few individuals, or of course in some intermediate régime. As discussed in section "A Note on More Complicated Memetic Algorithms", an adaptive approach could also be taken, allowing the algorithm to adjust these parameters itself.

An Example Memetic Algorithm for WEIGHTED CONSTRAINT SATISFACTION PROBLEMS

WEIGHTED CONSTRAINT SATISFACTION PROBLEMS (WCSPs) are a general class of combinatorial problems in which (i) solutions are assignment of values to a

collection of variables, each of them taken from a possibly different domain, (ii) there are hard constraints making some particular combinations of variable values infeasible, and (iii) there are some soft constraints establishing preferences among solutions. For example, consider a school timetabling problem in which courses have to be fit into different time slots: no two courses can use the same time slot if they are taught by the same lecturer (a hard constraint), and lecturer preferences (e.g., teaching in the morning or in the afternoon) have to be respected if possible. In essence, both types of constraints can be represented by defining a collection of integer functions f_i, one for each constraint; these functions are used to weight the fulfillment/violation of the corresponding constraint, and therefore an objective function F (to be minimized, without loss of generality) can be built by summing them. Thus, it will be typically the case that hard constraints have a much larger weight (even infinite if violated) than soft constraints.

Formally, a WCSP can be characterized as a triplet $\langle \mathscr{X}, \mathscr{D}, \mathscr{F} \rangle$, where each $x_i \in \mathscr{X}$, $1 \leqslant i \leqslant n$ is a problem variable whose domain is $D_i \in \mathscr{D}$. Each function $f_j \in \mathscr{F}$, $1 \leqslant j \leqslant m$ has signature $f_j : V_j \to \mathbb{N}$, where $V_j \in 2^{\mathscr{X}}$ is the subset of variables involved in the j-th constraint. With this formulation, a naïve evolutionary approach can be defined by using the Cartesian product $\mathscr{S} = D_1 \times \cdots \times D_n$ as search space, taking the fitness function to be $F(x) = \sum_j \hat{f}_j(x)$ (where $\hat{f}_i$ is a function that picks from its argument the variables in V_j and feeds them to f_j), and utilizing standard operations for recombination and mutation. Such an approach is however going to perform poorly in general due to the lack of problem-specific knowledge. A much more sensible approach can be built on the basis of (i) a smart recombination operator and (ii) a powerful local search technique.

Regarding recombination, it is very easy to define a greedy recombination mechanism for WCSPs: (1) start from a solution s with all variables unassigned, (2) sort constraints in some particular order (arbitrary or heuristically selected) $j_1, \cdots, j_m$, and (3) traverse this ordered list of constraints, checking for each j_k the variables in V_{j_k} that are still unassigned in s, constructing two (or as many as parents) candidate sets using the assigned values in s plus the values that the remaining variables in V_{j_k} take in either parent, and keeping the candidate set v minimizing $f_{j_k}(v)$ (which is subsequently used to expand the solution s). This procedure has been used, for example, in [57] for the construction of Golomb rulers and in [225] for the construction of balanced incomplete blocks, to cite just two examples.

It is possible to define a more intensive recombination approach by taking ideas from complete techniques [59]. More precisely, a complete technique can be used to explore the set of potential solutions that can be created using a given collection of parents, returning the best solution attainable. Different possibilities can be used for this purpose such as branch and bound [55] or integer linear programming techniques [164]. A more WCSP-specific approach can be found in the use of bucket elimination (BE) [63]. BE can be regarded as a dynamic programming approach based on the successive elimination of variables and the use of an auxiliary table to store the best value of the fitness function for specific variable combinations. More precisely, BE considers some ordering of the variables (again, arbitrarily or

heuristically selected – it must be noted that while the particular choice ordering is irrelevant for correction purposes, it can have a huge impact in the computational complexity of the algorithm though) $i_1, \cdots, i_n$. Then, it traverses this sequence and for each variable x_{i_k} (1) determines the constraints $\mathscr{C} \subseteq \mathscr{F}$ in which x_{i_k} is involved; (2) computes the bucket

$$B_{i_k} = \left(\cup_{f_j \in \mathscr{C}} V_j\right) \setminus \{x_{i_k}\},$$

namely, the collection of variables related to x_{i_k} in any constraint; (3) determines for each combination t of values for variables in B_{i_k} the value v_t^* for x_{i_k} such that $w = \sum_{f_j \in \mathscr{C}} \hat{f}_j(t \cdot (x_{i_k} = v))$ is minimal; and (4) removes $\mathscr{C}$ from $\mathscr{F}$ and adds a new constraint f' with domain $V' = B_{i_k}$ defined as $f'(t) = \sum_{f_j \in \mathscr{C}} \hat{f}_j(t \cdot (x_{i_k} = v_t^*))$. When all variables have been eliminated, the optimal cost w is found, and one only has to trace back the process (using the auxiliary table) to determine the best variable assignment [93]. This procedure has been used with great success in [91] for solving the MAXIMUM DENSITY STILL LIFE PROBLEM in conduction with a local search procedure based on tabu search.

A potential drawback of recombination schemes such as those defined above is scalability: the use of an exact technique for recombination is less costly than using it to solve the problem completely from scratch, but its cost will nevertheless grow with the problem size until becoming impractical at some point. To alleviate this problem, the granularity of the representation can be adjusted [54], that is, grouping variables in larger chunks which are subsequently used as basic units for the purposes of constructing solutions (hence reducing the number of potential solutions attainable and therefore the computational cost of the exact technique). In the context of the BE method described before, this approach is termed mini-buckets [64] and can be readily applied to the recombination mechanism described above [93]. Another source of difficulties is the existence of symmetries or partial isomorphisms between solutions. This scenario is typical in many WCSPs in which variables or groups thereof can be relabeled without altering the solution. In such a situation, recombination can reduce to macromutation unless it is effectively capable of identifying correspondences between variables in different parents. This is, for instance, done with success in [171] in the context of clustering genomic data. Of course, it may be very complex in general to find a perfect matching between variables in an arbitrary WCSP with symmetries. In problems for which this is deemed too complicated or time-consuming, it must be noted that a recombination-less MA – essentially a population of local searchers subject to interleaved phases of self-improvement and competition via selection/replacement, much in the line of go-with-the-winner approaches [8] – can also provide acceptable results. This is, for example, the case of the SOCIAL GOLFER PROBLEM, a WCSP with a large degree of symmetry that was successfully attacked using a memetic evolutionary programming approach [56] (an MA propelled by selection, mutation, and local improvement).

A Brief Survey of Recent Memetic Algorithm Applications

In recent years, MAs have become a significant part of the optimization toolkit and have become particularly well used in recent years. As a rough gauge, the number of academic papers (found via searching DBLP and ISI Web of Science for relevant papers with the word "memetic" in their title, abstract, or keywords) published has risen to over 300 per year since 2011, with thousands of academic publications in total since 1998. Possibly the most interesting aspect of this expanding interest in memetic algorithms is the diversity of techniques and application areas.

Memetic Algorithms in the Wild

While many algorithms developed in the areas of Computer Science and Optimization are demonstrated via application to practical problems drawn from a variety of areas, a more reliable indicator of the effectiveness of a technique is the adoption of the technique as a tool within the communities from which the problems are drawn. The following briefly surveys some of the areas in which memetic algorithms have been successfully applied. Table 1 gives an overview of the breadth of application areas for memetic algorithms, with recent references. Of course this table is far from exhaustive, even within the application areas mentioned. As a matter of fact, in some areas the number of memetic applications has deserved individualized treatment in specialized surveys, e.g., scheduling and timetabling [50], engineering and design [37], bioinformatics [24], etc. – see also [189] for a recent general application

Table 1 Some recent publications reporting on memetic algorithm applications by field of application

Application area	References
Biology	[89, 90, 186, 187, 197, 201, 238–241, 269, 270]
Chemistry	[77–80, 108, 194]
Chemical engineering	[47, 75, 140, 141, 150, 158, 242, 253–257]
Data compression	[146, 169, 246, 271, 275, 280]
Drug design	[104, 109, 130, 161, 191, 192, 251, 252]
Electronic engineering	[41, 46, 97, 98, 101, 113–118, 135, 200, 205–207, 210, 214, 272]
Finance	[15, 45, 71, 243]
Geoscience	[36, 258]
Image analysis	[66, 127, 162, 228]
Materials science & engineering	[14, 25, 124, 125, 222, 260]
Microarray analysis	[12, 13, 19, 73, 94, 148, 167, 182, 208, 237, 278]
Computer networking	[16, 215, 216, 248, 261, 276]
Oncology	[1, 39, 74, 133, 166, 230, 251, 252, 279, 281]
Operations research	[2, 6, 10, 22, 40, 68, 95, 111, 129, 159, 160, 165, 209, 212, 213, 219, 227, 259, 266, 268, 274, 277]
Physics	[5, 103, 139, 264, 273]
Power engineering	[17, 18, 26, 67, 120, 121, 136, 147, 149, 151, 157, 168, 183, 195, 220, 223, 232]

survey. The breadth of the application areas suggests a significant generality and flexibility in the memetic paradigm.

Memetic Speciation

Along with a wide set of application areas, memetic algorithms have also embraced many forms, employing a wide variety of combinations of population-based heuristics and individual improvement heuristics. Table 2 lists some of the more prominent combinations, with example references for each. Not only are different combinations of population-based heuristic and individual improvement heuristic extant, more exotic memetic algorithms that use heuristics of only one type, or multiple heuristics of each type, exist. The adaptability of memetic algorithms to parallel implementation also encourages the use of multiple different types of heuristics simultaneously – the exploitation of *all* available knowledge is, after all, the central idea of the memetic paradigm.

Table 2 Some varietal combinations of heuristics forming memetic algorithms

Population-based heuristic	Individual improvement heuristic	References
Ant colony optimization	Local search	[44, 72, 84, 154]
Bee colony optimization	–	[32, 34]
	Random optimization	[21]
	Nelder-Mead simplex	[85]
–	Nelder-Mead simplex with bidirectional random optimization	[4]
Binary differential evolution	Tabu search	[102]
Continuous differential evolution	Pool of strategies	[122, 249]
	Hooke-Jeeves-like	[211]
	Stochastic local search	[190]
Cross entropy	Hill climbing, tabu search	[11]
Genetic algorithm	Local search	[15, 22]
	Tabu search	[27, 33, 92, 153, 155, 263]
	Mathematical programming	[254]
Genetic algorithm with particle swarm optimization	–	[26, 119]
Particle swarm optimization	Local search	[20, 35, 65, 121, 274]
	Variable neighborhood search	[9]
	Sequential quadratic programming	[221]
Particle swarm optimization with differential evolution	Nelder-Mead simplex with Rosenbrock algorithm	[38]

Future-Generation Memetic Algorithms

Back in the days when MAs were just a nascent approach for optimization, different visions of what MAs would be in the future were foreseen. Among these, maybe the one which has come closest to reality refers to the self-⋆ capabilities [23] of the paradigm and more precisely to self-generation properties. Early works envisioned that the algorithm could work on two timescales, one in which solutions would be optimized and another one in which the problem-solving strategies used to optimize solutions would be themselves optimized [177]. In essence, this has been a long-standing goal in metaheuristics. It is widely acknowledged that the design of an effective problem-solving technique is in itself a hard task. Attempting to transfer a part of this design effort to the actual metaheuristic is just the logical course of action [58] – see, for example, the corpus of research in hyperheuristics [42, 60]. This latter approach is actually related to what has been termed "meta-Lamarckian" learning [202], a memetic approach in which a collection of local searchers is available and there is a decision-maker that decides which of them should be applied to specific solutions based on different criteria (e.g., the past performance of each local searcher, the adequacy of the current solution for being improved by a certain local searcher according to past experience, etc.). A much more general approach was provided by multi-memetic or multimeme algorithms [142, 143, 145]. In this approach an encoding of a local searcher (ranging from the definition of the neighborhood or pivot rule used up to a full algorithmic description of the procedure) is attached to each solution and evolves alongside it. Thus, the algorithm not only looks for improved solutions but also for algorithmic structures capable of improving the latter. The next natural step is detaching these memes from the genes and have them evolve in separate populations [233–235], paving the way for the emergence of complex structures of interacting memes [43]. An overview of adaptation in MAs is provided in [203]. This view of memes as explicit representations of problem-solving strategies that interact in a complex and dynamic way within an evolutionary context for optimization purposes leads to the notion of memetic computing [193, 204] – see [189] for a literature review on memetic computing. A further iteration of this concept is to apply metaheuristic approaches to develop worst-case instances of a problem, which can then be fed back into the process of optimizing the algorithm. This technique has been explored in regard to sorting [51] and the TRAVELLING SALESMAN PROBLEM [3].

Another dimension along which some early ideas (farfetched at their time) about MAs may become a reality is parallel computing. The deployment of metaheuristics in parallel and/or distributed environments is by no means new [7] and has been extensively used since the late 1980s; see, for example, [184, 247]. However, the continuous evolution of computational platforms is dragging these parallel modes along, forcing them to adapt to new scenarios. Thus, whereas early works often assumed dedicated local area networks, it is nowadays more common to have emerging computational environments such as peer-to-peer networks [172] and volunteer computing networks [229], which are much more pervasive, of a larger scale and inherently dynamic. Coping with the complex, dynamic structure of the

computational substrate is undoubtedly a challenge. Fortunately, population-based metaheuristics have been shown to be intrinsically robust at a fine-grain scale [128] and can be endowed with appropriate churn-aware strategies if required [196]. They are therefore ripe for being deployed on these platforms to exploit the possibilities they offer. In this line – and connected to the previous discussion on meme evolution and interaction – some initial concepts revolving around "meme pools," that is, repositories of problem-solving methods to be used synergistically, acquire a new scope more akin to service-oriented architectures [96]. Furthermore, to build on the idea of automated self-design of the MA requires the ability to keep or gather some sort of distributed knowledge about the state of the search and make design decisions on its basis. Some ideas from multi-agent systems and epistemic logic were proposed as potential tools for this purpose [52], but the concept still remains largely unexplored.

There are also opportunities for the development of MAs (and GAs) at the small scale. Any use of recombination operators is naturally limited by the expectation that the recombination step will be performed many, many times during a run of the algorithm. This leads to the requirement that a recombination operator must be able to be implemented very efficiently. Traditionally this would mean at most linear or close to linear time in the size of the individual (of course, ideally constant time). This immediately rules out the possibility of optimal recombination strategies for many problems, as typically such strategies would be NP-hard. Parameterized complexity, for example, offers some opportunity to exploit the naturally arising parameters in many recombination strategies. If such parameters are small, or can be made small, then the complexity of optimal recombination may be effectively reduced to polynomial time [53,81–83]. For further reading on the challenges raised by evolutionary approaches to optimization, many of the problems posed in [52] remain open.

Conclusion

Since their primordial conception in the late 1980s, memetic algorithms have developed to become one of the most adaptable and flexible metaheuristic approaches available. While many heuristic techniques perform well for some problems, the No-Free-Lunch theorem [267] guarantees that their performance falters on the majority of problems. Memetic algorithms, with their insistence on adaptability and utilitarianism (both on the part of the algorithm and the implementer), are free to exploit the performance of multiple approaches and choose the best suited for the problem at hand.

The adaptability, efficiency, and amenability to the current availability of large-scale parallelism, including traditional parallel architectures along with GPU computing and cloud- and peer-based approaches, along with a tendency toward modularity in implementation, have led to their adoption across a broad range of fields with excellent results. The field of memetic algorithms research has grown dramatically since 1998. With over 2000 academic papers published at a current rate

of over 300 per year, the field is vibrant and dynamic. The importance and influence of memetic algorithms has grown such that Thomson Reuters selected it as one of the top ten research fronts in Mathematics, Computer Science, and Engineering in 2013 [137]. To put it simply, memetic algorithms are one of the most flexible and effective tools in the heuristic toolbox and a key technique for anyone involved in combinatorial optimization to learn.

Cross-References

- Adaptive and Multilevel Metaheuristics
- Evolution Strategies
- Hyper-heuristics
- Scatter Search
- Tabu Search
- Variable Neighborhood Search

Acknowledgments Carlos Cotta acknowledges support from the MINECO under the project EphemeCH (TIN2014-56494-C4-1-P), from the Junta de Andalucía under the project P10-TIC-6083 (DNEMESIS) and from the Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

Pablo Moscato acknowledges support from the Australian Research Council Future Fellowship FT120100060 and Australian Research Council Discovery Projects DP120102576 and DP140104183.

References

1. Abbass HA (2002) An evolutionary artificial neural networks approach for breast cancer diagnosis. Artif Intell Med 25(3):265–281
2. Afsar HM, Prins C, Santos AC (2014) Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size. Int Trans Oper Res 21(1): 153–175
3. Ahammed F, Moscato P (2011) Evolving L-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances. In: Di Chio C et al (eds) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol 6624. Springer, Berlin, pp 1–11
4. Ahandani MA, Vakil-Baghmisheh MT, Talebi M (2014) Hybridizing local search algorithms for global optimization. Comput Optim Appl 59(3):725–748
5. Ahn Y, Park J, Lee CG, Kim JW, Jung SY (2010) Novel memetic algorithm implemented with GA (genetic algorithm) and MADS (mesh adaptive direct search) for optimal design of electromagnetic system. IEEE Trans Magn 46(6):1982–1985
6. Al-Betar MA, Khader AT, Abu Doush I (2014) Memetic techniques for examination timetabling. Ann Oper Res 218(1):23–50
7. Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley-Interscience, Hoboken
8. Aldous D, Vazirani U (1994) “go with the winners” algorithms. In: Proceedings of 35th IEEE Symposium on Foundations of Computer Science. IEEE Press, Los Alamitos, pp 492–501

9. Ali AF, Hassanien AE, Snasel V, Tolba MF (2014) A new hybrid particle swarm optimization with variable neighborhood search for solving unconstrained global optimization problems. In: Kromer P, Abraham A, Snasel V (eds) Fifth International Conference on Innovations in Bio-inspired Computing and Applications. Advances in Intelligent Systems and Computing, vol 303. Springer, Berlin/Ostrava, pp 151–160
10. Amaya JE, Cotta C, Fernández-Leiva AJ (2012) Solving the tool switching problem with memetic algorithms. Artif Intell Eng Des Anal Manuf 26:221–235
11. Amaya JE, Cotta C, Fernández-Leiva AJ (2013) Cross entropy-based memetic algorithms: an application study over the tool switching problem. Int J Comput Intell Syst 6(3):559–584
12. Andres Gallo C, Andrea Carballido J, Ponzoni I (2009) BiHEA: a hybrid evolutionary approach for microarray biclustering. In: Guimaraes K, Panchenko A, Przytycka T (eds) 4th Brazilian Symposium on Bioinformatics (BSB 2009). Lecture Notes in Bioinformatics, vol 5676. Springer, Berlin/Porto Alegre, pp 36–47
13. Andres Gallo C, Andrea Carballido J, Ponzoni I (2009) Microarray biclustering: a novel memetic approach based on the PISA platform. In: Pizzuti C, Ritchie M, Giacobini M (eds) 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Lecture Notes in Computer Science, vol 5483. Springer, Berlin/Tubingen, pp 44–55
14. António CC (2014) A memetic algorithm based on multiple learning procedures for global optimal design of composite structures. Memetic Comput 6(2):113–131
15. Arab A, Alfi A (2015) An adaptive gradient descent-based local search in memetic algorithm applied to optimal controller design. Inf Sci 299:117–142
16. Arivudainambi D, Balaji S, Rekha D (2014) Improved memetic algorithm for energy efficient target coverage in wireless sensor networks. In: 11th IEEE International Conference on Networking, Sensing and Control (ICNSC), Miami, pp 261–266
17. Arshi SS, Zolfaghari A, Mirvakili SM (2014) A multi-objective shuffled frog leaping algorithm for in-core fuel management optimization. Comput Phys Commun 185(10):2622–2628
18. de Assis LS, Vizcaino Gonzalez JF, Usberti FL, Lyra C, Cavellucci C, Von Zuben FJ (2015) Switch allocation problems in power distribution systems. IEEE Trans Power Syst 30(1):246–253
19. Ayadi W, Hao JK (2014) A memetic algorithm for discovering negative correlation biclusters of DNA microarray data. Neurocomputing 145:14–22
20. Aziz M, Tayarani-N MH (2014) An adaptive memetic particle swarm optimization algorithm for finding large-scale latin hypercube designs. Eng Appl Artif Intell 36:222–237
21. Baghmisheh MTV, Ahandani MA, Talebi M (2008) Frequency modulation sound parameter identification using novel hybrid evolutionary algorithms. In: International Symposium on Telecommunications. IEEE, Tehran, pp 67–72
22. Benlic U, Hao JK (2015) Memetic search for the quadratic assignment problem. Expert Syst Appl 42(1):584–595
23. Berns A, Ghosh S (2009) Dissecting self-$\star$ properties. In: Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems. IEEE Press, San Francisco, pp 10–19
24. Berretta R, Cotta C, Moscato P (2012) Memetic algorithms in bioinformatics. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 261–271
25. Bertagnoli G, Giordano L, Mancini S (2014) Optimization of concrete shells using genetic algorithms. ZAMM: Zeitschrift für Angewandte Mathematik und Mechanik 94(1–2, SI): 43–54
26. Biao S, Hua HC, Hua YX, Chuan H (2014) Mutation particle swarm optimization algorithm for solving the optimal operation model of thermal power plants. J Renew Sustain Energy 6(4):043118
27. Bilal N, Galinier P, Guibault F (2014) An iterated-tabu-search heuristic for a variant of the partial set covering problem. J Heuristics 20(2):143–164

28. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308
29. Blum C, Blesa Aguilera MJ, Roli A, Sampels M (2008) Hybrid metaheuristics: an emerging approach to optimization. Studies in computational intelligence, vol 144. Springer, Berlin/Heidelberg
30. Blum C, Cotta C, Fernández AJ, Gallardo JE, Mastrolilli M (2008) Hybridizations of metaheuristics with branch & bound derivates. In: Blum C, Blesa Aguilera MJ, Roli A, Sampels M (eds) Hybrid Metaheuristics, an Emerging Approach to Optimization. Studies in Computational Intelligence, vol 144. Springer, Berlin/Heidelberg, pp 85–116
31. Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11(6):4135–4151
32. Bose D, Biswas S, Vasilakos AV, Laha S (2014) Optimal filter design using an improved artificial bee colony algorithm. Inform Sci 281:443–461
33. Boskovic B, Brglez F, Brest J (2014) Low-autocorrelation binary sequences: on the performance of memetic-tabu and self-avoiding walk solvers. CoRR abs/1406.5301
34. Bullinaria JA, AlYahya K (2014) Artificial bee colony training of neural networks: comparison with back-propagation. Memetic Comput 6(3):171–182
35. Cai K, Zhang J, Zhou C, Cao X, Tang K (2012) Using computational intelligence for large scale air route networks design. Appl Soft Comput 12(9):2790–2800
36. Caorsi S, Massa A, Pastorino M, Randazzo A (2003) Electromagnetic detection of dielectric scatterers using phaseless synthetic and real data and the memetic algorithm. IEEE Trans Geosci Remote Sens 41(12):2745–2753
37. Caponio A, Neri F (2012) Memetic algorithms in engineering and design. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 241–260
38. Caponio A, Neri F, Tirronen V (2009) Super-fit control adaptation in memetic differential evolution frameworks. Soft Comput 13(8–9, SI):811–831
39. Capp A, Inostroza-Ponta M, Bill D, Moscato P, Lai C, Christie D, Lamb D, Turner S, Joseph D, Matthews J, Atkinson C, North J, Poulsen M, Spry NA, Tai KH, Wynne C, Duchesne G, Steigler A, Denham JW (2009) Is there more than one proctitis syndrome? A revisitation using data from the TROG 96.01 trial. Radiother Oncol 90(3):400–407
40. Cattaruzza D, Absi N, Feillet D, Vidal T (2014) A memetic algorithm for the multi trip vehicle routing problem. Eur J Oper Res 236(3):833–848
41. Chabuk T, Reggia J, Lohn J, Linden D (2012) Causally-guided evolutionary optimization and its application to antenna array design. Integr Comput Aided Eng 19(2):111–124
42. Chakhlevitch K, Cowling P (2008) Hyperheuristics: recent developments. In: Cotta C, Sevaux M, Sörensen K (eds) Adaptive and Multilevel Metaheuristics. Studies in Computational Intelligence, vol 136. Springer, Berlin/Heidelberg, pp 3–29
43. Chen X, Ong YS (2012) A conceptual modeling of meme complexes in stochastic search. IEEE Trans Syst Man Cybern C 42(5):612–625
44. Chen X, Ong YS, Feng L, Lim MH, Chen C, Ho CS (2013) Towards believable resource gathering behaviours in real-time strategy games with a memetic ant colony system. In: Cho S, Sato A, Kim K (eds) 17th Asia Pacific Symposium on Intelligent and Evolutionary Systems. Procedia Computer Science, vol 24. Elsevier, Seoul, pp 143–151
45. Chiam SC, Tan KC, Mamun AM (2009) A memetic model of evolutionary pso for computational finance applications. Expert Syst Appl 36(2):3695–3711
46. Chowdhury A, Giri R, Ghosh A, Das S, Abraham A, Snasel V (2010) Linear antenna array synthesis using fitness-adaptive differential evolution algorithm. In: IEEE Congress on Evolutionary Computation (CEC 2010). IEEE, Barcelona
47. Conradie AVE, Aldrich C (2010) Neurocontrol of a multi-effect batch distillation pilot plant based on evolutionary reinforcement learning. Chem Eng Sci 65(5):1627–1643
48. Cotta C (2003) Protein structure prediction using evolutionary algorithms hybridized with backtracking. In: Mira J, Álvarez J (eds) Artificial Neural Nets Problem Solving Methods. Lecture Notes in Computer Science, vol 2687. Springer, Berlin/Heidelberg, pp 321–328

49. Cotta C (2005) Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In: Mira J, Álvarez J (eds) Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach. Lecture Notes in Computer Science, vol 3562. Springer, Berlin/Heidelberg, pp 84–91
50. Cotta C, Fernández A (2007) Memetic algorithms in planning, scheduling, and timetabling. In: Dahal K, Tan K, Cowling P (eds) Evolutionary Scheduling. Studies in Computational Intelligence, vol 49. Springer, Berlin, pp 1–30
51. Cotta C, Moscato P (2003) A mixed evolutionary-statistical analysis of an algorithm's complexity. Appl Math Lett 16:41–47
52. Cotta C, Moscato P (2004) Evolutionary computation: challenges and duties. In: Menon A (ed) Frontiers of Evolutionary Computation. Kluwer Academic, Boston, pp 53–72
53. Cotta C, Moscato P (2005) The parameterized complexity of multiparent recombination. In: Proceedings of the 6th Metaheuristic International Conference – MIC 2005, Universität Wien, Vienna, pp 237–242
54. Cotta C, Troya JM (2000) On the influence of the representation granularity in heuristic forma recombination. In: Carroll J, Damiani E, Haddad H, Oppenheim D (eds) Applied Computing 2000. ACM, New York, pp 433–439
55. Cotta C, Troya JM (2003) Embedding branch and bound within evolutionary algorithms. Appl Intell 18(2):137–153
56. Cotta C, Dotú I, Fernández AJ, Van Hentenryck P (2006) Scheduling social golfers with memetic evolutionary programming. In: Almeida F et al (eds) Hybrid Metaheuristics – HM 2006. Lecture Notes in Computer Science, vol 4030. Springer, Berlin/Heidelberg, pp 150–161
57. Cotta C, Dotú I, Fernández AJ, Van Hentenryck P (2007) Local search-based hybrid algorithms for finding golomb rulers. Constraints 12(3):263–291
58. Cotta C, Sevaux M, Sörensen K (2008) Adaptive and multilevel metaheuristics. Studies in computational intelligence, vol 136. Springer, Berlin/Heidelberg
59. Cotta C, Fernández Leiva AJ, Gallardo JE (2012) Memetic algorithms and complete techniques. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 189–200
60. Cowling P, Kendall G, Soubeiga E (2008) A hyperheuristic approach to schedule a sales submit. In: Burke E, Erben W (eds) PATAT 2000. Lecture Notes in Computer Science, vol 2079. Springer, Berlin/Heidelberg, pp 176–190
61. Davis L (1991) Handbook of genetic algorithms. Van Nostrand Reinhold Computer Library, New York
62. Dawkins R (1976) The selfish gene. Clarendon Press, Oxford
63. Dechter R (1999) Bucket elimination: a unifying framework for reasoning. Artif Intell 113(1–2):41–85
64. Detcher R, Rish I (2003) Mini-buckets: a general scheme for bounded inference. J ACM 50(2):107–153
65. Devi S, Jadhav DG, Pattnaik SS (2011) PSO based memetic algorithm for unimodal and multimodal function optimization. In: Panigrahi B, Suganthan P, Das S, Satapathy S (eds) 2nd Swarm, Evolutionary and Memetic Computing Conference. Lecture Notes in Computer Science, vol 7076. Springer, Berlin, pp 127–134
66. Di Gesù V, Lo Bosco G, Millonzi F, Valenti C (2008) A memetic algorithm for binary image reconstruction. In: Proceedings of the 2008 International Workshop on Combinatorial Image Analysis, Buffalo. Lecture Notes in Computer Science, vol 4958, pp 384–395
67. Dimitroulas DK, Georgilakis PS (2011) A new memetic algorithm approach for the price based unit commitment problem. Appl Energy 88(12):4687–4699
68. Divsalar A, Vansteenwegen P, Sorensen K, Cattrysse D (2014) A memetic algorithm for the orienteering problem with hotel selection. Eur J Oper Res 237(1):29–49
69. Droste S, Jansen T, Wegener I (1999) Perhaps not a free lunch but at least a free appetizer. In: Banzhaf W, Daida JM, Eiben AE, Garzon MH, Honavar V, Jakiela MJ, Smith RE (eds) Proceedings of the First Genetic and Evolutionary Computation Conference – GECCO 1999. Morgan Kaufmann, Orlando, pp 833–839

70. Droste S, Jansen T, Wegener I (2002) Optimization with randomized search heuristics – the (A)NFL theorem, realistic scenarios, and difficult functions. Theor Comput Sci 287(1): 131–144
71. Du J, Rada R (2012) Memetic algorithms, domain knowledge, and financial investing. Memetic Comput 4(2):109–125
72. Duan H, Yu X (2007) Hybrid ant colony optimization using memetic algorithm for traveling salesman problem. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. IEEE, Honolulu, pp 92–95
73. Duval B, Hao JK (2010) Advances in metaheuristics for gene selection and classification of microarray data. Brief Bioinform 11(1):127–141
74. Duval B, Hao JK, Hernández JCH (2009) A memetic algorithm for gene selection and molecular classification of cancer. In: 11th Annual Conference on Genetic and Evolutionary Computation – GECCO '09. ACM, New York, pp 201–208
75. Egea JA, Balsa-Canto E, Garcia MSG, Banga JR (2009) Dynamic optimization of nonlinear processes with an enhanced scatter search method. Ind Eng Chem Res 48(9): 4388–4401
76. Eiben AE, Raue PE, Ruttkay Z (1994) Genetic algorithms with multi-parent recombination. In: Davidor Y, Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature III. Lecture Notes in Computer Science, vol 866. Springer, Berlin/Heidelberg, pp 78–87
77. Ellabaan M, Ong YS, Handoko SD, Kwoh CK, Man HY (2013) Discovering unique, low-energy transition states using evolutionary molecular memetic computing. IEEE Comput Intell Mag 8(3):54–63
78. Ellabaan MM, Handoko SD, Ong YS, Kwoh CK, Bahnassy SA, Elassawy FM, Man HY (2012) A tree-structured covalent-bond-driven molecular memetic algorithm for optimization of ring-deficient molecules. Comput Math Appl 64(12, SI):3792–3804
79. Ellabaan MMH, Chen X, Nguyen QH (2012) Multi-modal valley-adaptive memetic algorithm for efficient discovery of first-order saddle points. In: Bui L et al (eds) Simulated Evolution and Learning. Lecture Notes in Computer Science, vol 7673. Berlin/Heidelberg, pp 83–92
80. Ellabaan MMH, Ong YS, Nguyen QC, Kuo JL (2012) Evolutionary discovery of transition states in water clusters. J Theor Comput Chem 11(5):965–995
81. Eremeev AV (2008) On complexity of optimal recombination for binary representations of solutions. Evol Comput 16(1):127–147
82. Eremeev AV (2011) On complexity of the optimal recombination for the travelling salesman problem. In: Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, vol 6622. Springer, Berlin, pp 215–225
83. Eremeev AV, Kovalenko JV (2014) Optimal recombination in genetic algorithms, Part II. Yugoslav J Oper Res 24(2):165–186
84. Fathi M, Rodriguez V, Jesus Alvarez M (2014) A novel memetic ant colony optimization-based heuristic algorithm for solving the assembly line part feeding problem. Inter J Adv Manuf Technol 75(1–4):629–643
85. Fister I, Fister I Jr, Brest J, Zumer V (2012) Memetic artificial bee colony algorithm for large-scale global optimization. In: IEEE Congress on Evolutionary Computation (CEC 2012). IEEE, Brisbane
86. Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, New York
87. França PM, Gupta JND, Mendes AS, Moscato P, Veltnik KJ (2005) Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Comput Ind Eng 48:491–506
88. Freisleben B, Merz P (1996) A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: 1996 IEEE International Conference on Evolutionary Computation, Nagoya. IEEE Press, pp 616–621

89. Gallardo JE (2012) A multilevel probabilistic beam search algorithm for the shortest common supersequence problem. PLoS ONE 7(12):1–14
90. Gallardo JE, Cotta C (2015) A GRASP-based memetic algorithm with path relinking for the far from most string problem. Eng Appl Artif Intell. https://doi.org/10.1016/j.engappai.2015.01.020
91. Gallardo JE, Cotta C, Fernández AJ (2006) A memetic algorithm with bucket elimination for the still life problem. In: Gottlieb J, Raidl G (eds) Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, vol 3906. Springer, Berlin/Heidelberg, pp 73–85
92. Gallardo JE, Cotta C, Fernández AJ (2009) Finding low autocorrelation binary sequences with memetic algorithms. Appl Soft Comput 9(4):1252–1262
93. Gallardo JE, Cotta C, Fernández AJ (2009) Solving weighted constraint satisfaction problems with memetic/exact hybrid algorithms. J Artif Intell Res 35:533–555
94. Gallo CA, Carballido JA, Ponzoni I (2009) Microarray biclustering: a novel memetic approach based on the PISA platform. In: Pizzuti C, Ritchie MD, Giacobini M (eds) Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Tübingen. Lecture Notes in Computer Science, vol 5483. Berlin/Heidelberg, pp 44–55
95. Gao L, Zhang C, Li X, Wang L (2014) Discrete electromagnetism-like mechanism algorithm for assembly sequences planning. Int J Prod Res 52(12):3485–3503
96. García-Sánchez P, González J, Castillo P, Arenas M, Merelo-Guervós J (2013) Service oriented evolutionary algorithms. Soft Comput 17(6):1059–1075
97. Garcia-Valverde T, Garcia-Sola A, Botia JA, Gomez-Skarmeta A (2012) Automatic design of an indoor user location infrastructure using a memetic multiobjective approach. IEEE Trans Syst Man Cybern C Appl Rev 42(5, SI):704–709
98. Ghosh P, Zafar H (2010) Linear array geometry synthesis with minimum side lobe level and null control using dynamic multi-swarm particle swarm optimizer with local search. In: Panigrahi B, Das S, Suganthan P, Dash S (eds) 1st International Conference on Swarm, Evolutionary, and Memetic Computing. Lecture Notes in Computer Science, vol 6466. Springer, Berlin/Chennai, pp 701–708
99. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman Publishing Co., Inc., Boston
100. Goldberg DE, Lingle R Jr (1985) Alleles, loci, and the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum Associates, Hillsdale
101. Goudos SK, Gotsis KA, Siakavara K, Vafiadis EE, Sahalos JN (2013) A multi-objective approach to subarrayed linear antenna arrays design based on memetic differential evolution. IEEE Trans Antennas and Propag 61(6):3042–3052
102. Gu X, Li Y, Jia J (2015) Feature selection for transient stability assessment based on kernelized fuzzy rough sets and memetic algorithm. Int J Electr Power Energy Syst 64:664–670
103. Guimaraes FG, Lowther DA, Ramirez JA (2008) Analysis of the computational cost of approximation-based hybrid evolutionary algorithms in electromagnetic design. IEEE Trans Magn 44(6):1130–1133
104. Handoko SD, Ouyang X, Su CTT, Kwoh CK, Ong YS (2012) QuickVina: accelerating AutoDock Vina using gradient-based heuristics for global optimization. IEEE-ACM Trans Comput Biol Bioinf 9(5):1266–1272
105. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130(3):449–467
106. Hao J (2012) Memetic algorithms in discrete optimization. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 73–94

107. Hart W, Belew R (1991) Optimizing an arbitrary function is hard for the genetic algorithm. In: Belew R, Booker L (eds) Fourth International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp 190–195
108. Hervas C, Silva M (2007) Memetic algorithms-based artificial multiplicative neural models selection for resolving multi-component mixtures based on dynamic responses. Chemom Intel Lab Syst 85(2):232–242
109. Hirsch R, Mullergoymann C (1995) Fitting of diffusion-coefficients in a 3-compartment sustained-release drug formulation using a genetic algorithm. Int J Pharm 120(2): 229–234
110. Holland JH (1992) Adaptation in natural and artificial systems. MIT, Cambridge
111. Hosseini S, Farahani RZ, Dullaert W, Raa B, Rajabi M, Bolhari A (2014) A robust optimization model for a supply chain under uncertainty. IMA J Manag Math 25(4):387–402
112. Houck C, Joines J, Kay M, Wilson J (1997) Empirical investigation of the benefits of partial lamarckianism. Evol Comput 5(1):31–60
113. Hsu CH (2007) Uplink MIMO-SDMA optimisation of smart antennas by phase-amplitude perturbations based on memetic algorithms for wireless and mobile communication systems. IET Commun 1(3):520–525
114. Hsu CH, Shyr WJ (2008) Adaptive pattern nulling design of linear array antenna by phase-only perturbations using memetic algorithms. Commun Numer Methods Eng 24(11):1121–1133
115. Hsu CH, Shyr WJ, Chen CH (2006) Adaptive pattern nulling design of linear array antenna by phase-only perturbations using memetic algorithms. In: Pan J, Shi P, Zhao Y (eds) First International Conference on Innovative Computing, Information and Control, vol 3 (ICICIC 2006). IEEE, Beijing, pp 308–311
116. Hsu CH, Chou PH, Shyr WJ, Chung YN (2007) Optimal radiation pattern design of adaptive linear array antenna by phase and amplitude perturbations using memetic algorithms. Int J Innov Comput Inf Control 3(5):1273–1287
117. Hsu CH, Shyr WJ, Ku KH, Chou PH (2008) Optimal radiation pattern design of adaptive linear phased array antenna using memetic algorithms. Int J Innov Comput Inf Control 4(9):2391–2403
118. Hsu CH, Shyr WJ, Kuo KH, Chou PH, Wu MJ (2010) Memetic algorithms for multiple interference cancellations of linear array based on phase-amplitude perturbations. J Optim Theory Appl 144(3):629–642
119. Hu M, Weir JD, Wu T (2014) An augmented multi-objective particle swarm optimizer for building cluster operation decisions. Appl Soft Comput 25:347–359
120. Hu Z, Bao Y, Xiong T (2013) Electricity load forecasting using support vector regression with memetic algorithms. Sci World J 2013:article ID 292575
121. Hu Z, Bao Y, Xiong T (2014) Comprehensive learning particle swarm optimization based memetic algorithm for model selection in short-term load forecasting using support vector regression. Appl Soft Comput 25:15–25
122. Iacca G, Neri F, Caraffini F, Suganthan PN (2014) A differential evolution framework with ensemble of parameters and strategies and pool of local search algorithms. In: Esparcia-Alcázar AI, Mora AM (eds) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol 8602. Springer, Berlin, pp 615–626
123. Ibaraki T (1997) Combination with dynamic programming. In: Bäck T, Fogel D, Michalewicz Z (eds) Handbook of Evolutionary Computation. Oxford University Press, New York NY, pp D3.4:1–2
124. Ihesiulor OK, Shankar K, Zhang Z, Ray T (2012) Delamination detection using methods of computational intelligence. In: Barsoum N, Faiman D, Vasant P (eds) Sixth Global Conference on Power Control and Optimization. AIP Conference Proceedings, vol 1499. American Institute of Physics, Las Vegas, pp 303–310
125. Ihesiulor OK, Shankar K, Zhang Z, Ray T (2014) Delamination detection with error and noise polluted natural frequencies using computational intelligence concepts. Compos B Eng 56:906–925

126. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans Evol Comput 7(2):204–223
127. Jiao L, Gong M, Wang S, Hou B, Zheng Z, Wu Q (2010) Natural and remote sensing image segmentation using memetic computing. IEEE Comput Intell Mag 5(2):78–91
128. Jiménez Laredo JL, Bouvry P, Lombraña González D, Fernández de Vega F, García Arenas M, Merelo Guervós JJ, Fernandes CM (2014) Designing robust volunteer-based evolutionary algorithms. Genet Program Evolvable Mach 15(3):221–244
129. Jolai F, Tavakkoli-Moghaddam R, Rabiee M, Gheisariha E (2014) An enhanced invasive weed optimization for makespan minimization in a flexible flowshop scheduling problem. Scientia Iranica 21(3):1007–1020
130. Jones G, Willett P, Glen R, Leach A, Taylor R (1997) Development and validation of a genetic algorithm for flexible docking. J Mol Biol 267(3):727–748
131. Jones T (1995) Evolutionary algorithms, fitness landscapes and search. PhD thesis, University of New Mexico
132. Julstrom BA (1995) Very greedy crossover in a genetic algorithm for the traveling salesman problem. In: Proceedings of the 1995 ACM Symposium on Applied Computing. ACM, New York, pp 324–328
133. Kalantzis G, Apte A, Radke R, Jackson A (2013) A reduced order memetic algorithm for constraint optimization in radiation therapy treatment planning. In: Takahashi S, Leo R (eds) 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking And Parallel/Distributed Computing (SNPD 2013). IEEE, Honolulu, pp 225–230
134. Kernighan B, Lin S (1972) An efficient heuristic procedure for partitioning graphs. Bell Syst J 49:291–307
135. Khan SU, Qureshi IM, Zaman F, Shoaib B, Naveed A, Basit A (2014) Correction of faulty sensors in phased array radars using symmetrical sensor failure technique and cultural algorithm with differential evolution. Sci World J 2014:article ID 852539
136. Kim J, Kim CS, Geem ZW (2014) A memetic approach for improving minimum cost of economic load dispatch problems. Math Probl Eng 2014:article ID 906028
137. King C, Pendlebury DA (2013) Web of knowledge research frontiers 2013: 100 top ranked specialties in the sciences and social sciences. http://sciencewatch.com/sites/sw/files/sw-article/media/research-fronts-2013.pdf
138. Klau GW (2009) A new graph-based method for pairwise global network alignment. BMC Bioinf 10(Suppl 1):S59
139. Kleeman MP, Lamont GB, Cooney A, Nelson TR (2007) A multi-tiered memetic multiobjective evolutionary algorithm for the design of quantum cascade lasers. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Proceeedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization (EMO 2007). Lecture Notes in Computer Science, vol 4403. Springer, Berlin/Matsuhima, pp 186–200
140. Kononova AV, Hughes KJ, Pourkashanian M, Ingham DB (2007) Fitness diversity based adaptive memetic algorithm for solving inverse problems of chemical kinetics. In: IEEE Congress on Evolutionary Computation. IEEE, Singapore, pp 2366–2373
141. Kononova AV, Ingham DB, Pourkashanian M (2008) Simple scheduled memetic algorithm for inverse problems in higher dimensions: application to chemical kinetics. In: IEEE Congress on Evolutionary Computation. IEEE, Hong Kong, pp 3905–3912
142. Krasnogor N (2004) Self generating metaheuristics in bioinformatics: the proteins structure comparison case. Genet Program Evolvable Mach 5(2):181–201
143. Krasnogor N, Gustafson S (2004) A study on the use of "self-generation" in memetic algorithms. Nat Commun 3(1):53–76
144. Krasnogor N, Smith J (2008) Memetic algorithms: the polynomial local search complexity theory perspective. J Math Modell Algorithms 7(1):3–24
145. Krasnogor N, Blackburne B, Burke E, Hirst J (2002) Multimeme Algorithms for Protein Structure Prediction. Lecture Notes in Computer Science, vol 2439. Springer, Berlin, pp 769–778

146. Krishna K, Ramakrishnan K, Thathachar M (1997) Vector quantization using genetic k-means algorithm for image compression. In: 1997 International Conference on Information, Communications and Signal Processing, vol 3. IEEE Press, New York, pp 1585–1587
147. Kumar JV, Kumar DMV (2014) Generation bidding strategy in a pool based electricity market using shuffled frog leaping algorithm. Appl Soft Comput 21:407–414
148. Kumar PK, Sharath S, D'Souza RG, Chandra K (2007) Memetic nsga – a multi-objective genetic algorithm for classification of microarray data. In: 15th International Conference on Advanced Computing And Communications (ADCOM 2007). IEEE, Guwahati, pp 75–80
149. Kumle AN, Fathi SH, Broujeni ST (2014) Harmonic optimization in multi-level inverters by considering adjustable DC sources using memetic algorithm. In: 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2014). IEEE, Nakhon Ratchasima
150. Lam AYS, Li VOK (2012) Chemical reaction optimization: a tutorial. Memetic Comput 4(1):3–17
151. Li YF, Pedroni N, Zio E (2013) A memetic evolutionary multi-objective optimization method for environmental power unit commitment. IEEE Trans Power Syst 28(3):2660–2669
152. Liaw CF (2000) A hybrid genetic algorithm for the open shop scheduling problem. Eur J Oper Res 124:28–42
153. Liefooghe A, Verel S, Hao JK (2014) A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming. Appl Soft Comput 16:10–19
154. Lim KK, Ong YS, Lim MH, Chen X, Agarwal A (2008) Hybrid ant colony algorithms for path planning in sparse graphs. Soft Comput 12(10):981–994
155. Lin G, Zhu W, Ali MM (2014) A tabu search-based memetic algorithm for hardware/software partitioning. Math Probl Eng 2014:article ID 103059
156. Lin S, Kernighan B (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
157. Linda O, Wijayasekara D, Manic M, McQueen M (2014) Optimal placement of phasor measurement units in power grids using memetic algorithms. In: 23rd IEEE International Symposium on Industrial Electronics (ISIE 2014). IEEE, Istanbul, pp 2035–2041
158. Liu B, Wang L, Liu Y, Qian B, Jin YH (2010) An effective hybrid particle swarm optimization for batch scheduling of polypropylene processes. Comput Chem Eng 34(4):518–528
159. Liu S, Chen D, Wang Y (2014) Memetic algorithm for multi-mode resource-constrained project scheduling problems. J Syst Eng Electron 25(4):609–617
160. Liu T, Jiang Z, Geng N (2014) A genetic local search algorithm for the multi-depot heterogeneous fleet capacitated arc routing problem. Flex Serv Manuf J 26(4, SI):540–564
161. Lorber D, Shoichet B (1998) Flexible ligand docking using conformational ensembles. Protein Sci 7(4):938–950
162. Ma W, Huang Y, Li C, Liu J (2012) Image segmentation based on a hybrid immune memetic algorithm. In: Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, pp 1–8
163. Maheswaran R, Ponnambalam SG, Aranvidan C (2005) A meta-heuristic approach to single machine scheduling problems. Int J Adv Manuf Technol 25:772–776
164. Marino A, Prügel-Bennett A, Glass CA (1999) Improving graph colouring with linear programming and genetic algorithms. In: Proceedings of EUROGEN 99, Jyväskylä, pp 113–118
165. Matei O, Pop PC, Sas JL, Chira C (2015) An improved immigration memetic algorithm for solving the heterogeneous fixed fleet vehicle routing problem. Neurocomputing 150(A, SI):58–66
166. Mendes A (2011) Identification of breast cancer subtypes using multiple gene expression microarray datasets. In: Wang D, Reynolds M (eds) 24th Australasian Joint Conference on Artificial Intelligence (AI 2011). Lecture Notes in Artificial Intelligence, vol 7106. Springer Berlin/Perth, pp 92–101

167. Mendes A, Cotta C, Garcia V, França P, Moscato P (2005) Gene ordering in microarray data using parallel memetic algorithms. In: Skie T, Yang CS (eds) Proceedings of the 2005 International Conference on Parallel Processing Workshops. IEEE Press, Oslo, pp 604–611
168. Mendes A, Boland N, Guiney P, Riveros C (2013) Switch and tap-changer reconfiguration of distribution networks using evolutionary algorithms. IEEE Trans Power Syst 28(1):85–92
169. Mendoza M, Bonilla S, Noguera C, Cobos C, Leon E (2014) Extractive single-document summarization based on genetic operators and guided local search. Expert Syst Appl 41(9):4158–4169
170. Merz P (2012) Memetic algorithms and fitness landscapes in combinatorial optimization. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 95–119
171. Merz P, Zell A (2002) Clustering Gene Expression Profiles with Memetic Algorithms. Lecture Notes in Computer Science, vol 2439. Springer, Berlin, pp 811–820
172. Milojičić DS, Kalogeraki V, Lukose R, Nagaraja K, Pruyne J, Richard B, Rollins S, Xu Z (2002) Peer-to-peer computing. Technical report HPL-2002-57, Hewlett-Packard Labs
173. Molina D, Lozano M, García-Martínez C, Herrera F (2010) Memetic algorithms for continuous optimisation based on local search chains. Evol Comput 18(1):27–63
174. Molina D, Lozano M, Sánchez AM, Herrera F (2011) Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-chains. Soft Comput 15(11):2201–2220
175. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical report 826, California Institute of Technology, Pasadena
176. Moscato P (1993) An introduction to population approaches for optimization and hierarchical objective functions: the role of tabu search. Ann Oper Res 41(1–4):85–121
177. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F (eds) New Ideas in Optimization. McGraw-Hill, London, pp 219–234
178. Moscato P (2012) Memetic algorithms: the untold story. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 275–309
179. Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics. Kluwer Academic Publishers, Boston, pp 105–144
180. Moscato P, Norman MG (1992) A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Valero M, Onate E, Jane M, Larriba JL, Suarez B (eds) Parallel Computing and Transputer Applications. IOS Press, Amsterdam, pp 177–186
181. Moscato P, Tinetti F (1992) Blending heuristics with a population-based approach: a memetic algorithm for the traveling salesman problem. Report 92–12, Universidad Nacional de La Plata
182. Moscato P, Mendes A, Berretta R (2007) Benchmarking a memetic algorithm for ordering microarray data. Biosystems 88(1–2):56–75
183. Mozaffari A, Chehresaz M, Azad NL (2013) Component sizing of a plug-in hybrid electric vehicle powertrain, part a: coupling bio-inspired techniques to meshless variable-fidelity surrogate models. Int J Bio-Inspired Comput 5(6):350–383
184. Mühlenbein H (1989) Parallel genetic algorithms, population genetics and combinatorial optimization. In: Schaffer (ed) 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp 416–421
185. Nagata Y, Kobayashi S (1997) Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. In: Bäck T (ed) Seventh International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp 450–457

186. Nair SSK, Reddy NVS, Hareesha KS (2011) Exploiting heterogeneous features to improve in silico prediction of peptide status – amyloidogenic or non-amyloidogenic. BMC Bioinf 12(13):S21
187. Nair SSK, Reddy NVS, Hareesha KS (2012) Machine learning study of classifiers trained with biophysiochemical properties of amino acids to predict fibril forming peptide motifs. Protein Pept Lett 19(9):917–923
188. Neri F (2012) Diversity management in memetic algorithms. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 153–165
189. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol Comput 2:1–14
190. Neri F, Mininno E (2010) Memetic compact differential evolution for Cartesian robot control. IEEE Comput Intell Mag 5(2):54–65
191. Neri F, Toivanen J, Cascella GL, Ong YS (2007) An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE-ACM Trans Comput Biol Bioinform 4(2):264–278
192. Neri F, Toivanen J, Makinen RAE (2007) An adaptive evolutionary algorithm with intelligent mutation local searchers for designing multidrug therapies for HIV. Appl Intell 27(3):219–235
193. Neri F, Cotta C, Moscato P (eds) (2012) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg
194. Nguyen QC, Ong YS, Kuo JL (2009) A hierarchical approach to study the thermal behavior of protonated water clusters H+(H2O)(n). J Chem Theory Comput 5(10):2629–2639
195. Nikzad M, Farahani SSS, Tabar MB, Tourang H, Yousefpour B (2012) A new optimization method for pss design in New-England power system. Life Sci J Acta Zhengzhou Univ Overseas Ed 9(4):5478–5483
196. Nogueras R, Cotta C (2015) Studying fault-tolerance in Island-based evolutionary and multimemetic algorithms. J Grid Comput. https://doi.org/10.1007/s10723-014-9315-6
197. Noman N, Iba H (2007) Inferring gene regulatory networks using differential evolution with local search heuristics. IEEE-ACM Trans Comput Biol Bioinform 4(4):634–647
198. Norman M, Moscato P (1989) A competitive and cooperative approach to complex combinatorial search. Technical report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena. Expanded version published at the 20th Informatics and Operations Research Meeting, Buenos Aires (20th JAIIO), Aug 1991, pp 3.15–3.29
199. Montes de Oca MA, Cotta C, Neri F (2012) Local search. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 29–41
200. Oliveri G, Lizzi L, Pastorino M, Massa A (2012) A nested multi-scaling inexact-Newton iterative approach for microwave imaging. IEEE Trans Antennas Propag 60(2, 2):971–983
201. Olson BS, Shehu A (2012) Evolutionary-inspired probabilistic search for enhancing sampling of local minima in the protein energy surface. Proteome Sci 10(1):S5
202. Ong YS, Keane A (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Trans Evol Comput 8(2):99–110
203. Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst Man Cybern B: Cybern 36(1):141–152
204. Ong YS, Lim MH, Chen X (2010) Memetic computation-past, present and future. IEEE Comput Intell Mag 5(2):24–31
205. Ortega JC, Gimenez D, Alvarez-Melcon A, Quesada FD (2013) Hybrid metaheuristics for the design of coupled resonator filters. Appl Artif Intell 27(5):323–350
206. Pal S, Basak A, Das S, Abraham A (2009) Linear antenna array synthesis with invasive weed optimization algorithm. In: Abraham A, Muda A, Herman N, Shamsuddin S, Huoy C (eds) International Conference of Soft Computing and Pattern Recognition. IEEE, Malacca, pp 161–166
207. Pal S, Basak A, Das S (2011) Linear antenna array synthesis with modified invasive weed optimisation algorithm. Int J Bio-Inspired Comput 3(4):238–251

208. Palacios P, Pelta D, Blanco A (2006) Obtaining biclusters in microarrays with population-based heuristics. In: Rothlauf F (ed) Proceedings of Applications of Evolutionary Computing. Lecture Notes in Computer Science, vol 3907. Springer, Berlin/Budapest, pp 115–126
209. Pan QK, Dong Y (2014) An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. Inform Sci 277:643–655
210. Perales-Gravan C, Lahoz-Beltra R (2008) An AM radio receiver designed with a genetic algorithm based on a bacterial conjugation genetic operator. IEEE Trans Evol Comput 12(2):129–142
211. Poikolainen I, Neri F (2013) Differential evolution with concurrent fitness based local search. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation. IEEE Press, Cancun, pp 384–391
212. Prodhon C, Prins C (2014) A survey of recent research on location-routing problems. Eur J Oper Res 238(1):1–17
213. Qin H, Zhang Z, Qi Z, Lim A (2014) The freight consolidation and containerization problem. Eur J Oper Res 234(1):37–48
214. Quevedo-Teruel O, Rajo-Iglesias E, Oropesa-Garcia A (2007) Hybrid algorithms for electromagnetic problems and the no-free-lunch framework. IEEE Trans Antennas Propag 55(3, 1):742–749
215. Quintero A, Pierre S (2003) Evolutionary approach to optimize the assignment of cells to switches in personal communication networks. Comput Commun 26(9):927–938
216. Quintero A, Pierre S (2003) Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks. Comput Netw 43(3):247–261
217. Radcliffe N (1994) The algebra of genetic algorithms. Ann Math Artif Intell 10:339–384
218. Radcliffe NJ, Surry PD (1994) Formal memetic algorithms. In: Fogarty TC (ed) AISB Workshop on Evolutionary Computing. Lecture Notes in Computer Science, vol 865. Springer, Berlin/Heidelberg, pp 1–16
219. Rager M, Gahm C, Denz F (2015) Energy-oriented scheduling based on evolutionary algorithms. Comput Oper Res 54:218–231
220. Rahiminejad A, Alimardani A, Vahidi B, Hosseinian SH (2014) Shuffled frog leaping algorithm optimization for AC-DC optimal power flow dispatch. Turk J Electr Eng Comput Sci 22(4):874–892
221. Raja MAZ, Ahmad SuI, Samar R (2014) Solution of the 2-dimensional bratu problem using neural network, swarm intelligence and sequential quadratic programming. Neural Comput Appl 25(7–8):1723–1739
222. Rao ARM, Lakshmi K (2012) Optimal design of stiffened laminate composite cylinder using a hybrid sfl algorithm. J Compos Mater 46(24):3031–3055
223. Rao BS, Vaisakh K (2013) New variants/hybrid methods of memetic algorithm for solving optimal power flow problem with load uncertainty. Int J Hybrid Intell Syst (IJHIS) 10(3):117–128
224. Richter H, Engelbrecht A (2014) Recent Advances in the Theory and Application of Fitness Landscapes, Emergence, Complexity and Computation, vol 6. Springer, Berlin/Heidelberg
225. Rodríguez Rueda D, Cotta C, Fernández Leiva AJ (2011) A memetic algorithm for designing balanced incomplete blocks. Int J Comb Optim Probl Inform 2(1):14–22
226. Rothlauf F, Goldberg DE (2002) Representations for Genetic and Evolutionary Algorithms. Physica-Verlag, Heidelberg
227. Salhi A, Rodriguez JAV (2014) Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. Memetic Comput 6(2):77–84
228. Santamaría J, Cordón O, Damas S, García-Torres JM, Quirin A (2009) Performance evaluation of memetic approaches in 3D reconstruction of forensic object. Soft Comput 13(8–9):883–904
229. Sarmenta LF (1998) Bayanihan: web-based volunteer computing using java. In: Masunaga Y, Katayama T, Tsukamoto M (eds) Worldwide Computing and Its Applications – WWCA'98. Lecture Notes in Computer Science, vol 1368. Springer, Berlin/Heidelberg, pp 444–461

230. Schaefer G (2014) Aco classification of thermogram symmetry features for breast cancer diagnosis. Memetic Comput (3):207–212
231. Sevaux M, Dauzère-Pérès S (2003) Genetic algorithms to minimize the weighted number of late jobs on a single machine. Eur J Oper Res 151:296–306
232. Silva R, Berenguel M, Perez M, Fernandez-Garcia A (2014) Thermo-economic design optimization of parabolic trough solar plants for industrial process heat applications with memetic algorithms. Appl Energy 113(SI):603–614
233. Smith JE (2007) Coevolving memetic algorithms: a review and progress report. IEEE Trans Syst Man Cybern B Cybern 37(1):6–17
234. Smith JE (2010) Meme fitness and memepool sizes in coevolutionary memetic algorithms. In: 2010 IEEE Congress on Evolutionary Computation. IEEE Press, Barcelona, pp 1–8
235. Smith JE (2012) Self-Adaptative and Coevolving Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 167–188
236. Sörensen K, Sevaux M (2006) MA | PM: memetic algorithms with population management. Comput OR 33:1214–1225
237. Speer N, Merz P, Spieth C, Zell A (2003) Clustering gene expression data with memetic algorithms based on minimum spanning trees. In: IEEE Congress on Evolutionary Computation (CEC 2003), Canberra, pp 1848–1855
238. Speer N, Spieth C, Zell A (2004) A memetic clustering algorithm for the functional partition of genes based on the gene ontology. In: IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, La Jolla, pp 252–259
239. Speer N, Spieth C, Zell A (2004) A memetic co-clustering algorithm for gene expression profiles and biological annotation. In: Congress on Evolutionary Computation (CEC 2004). IEEE, Portland, pp 1631–1638
240. Spieth C, Streichert F, Speer N, Zell A (2004) A memetic inference method for gene regulatory networks based on s-systems. In: 2004 IEEE Congress on Evolutionary Computation (CEC 2004), Portland, pp 152–157
241. Spieth C, Streichert F, Supper J, Speer N, Zell A (2005) Feedback memetic algorithms for modeling gene regulatory networks. In: IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, La Jolla, pp 61–67
242. Steimel J, Engell S (2014) Conceptual design and optimisation of chemical processes under uncertainty by two-stage programming. In: Eden M, Siirola J, Towler G (eds) 8th International Conference on Foundations of Computer-Aided Process Design, vol 34. Elsevier Science BV, Cle Elum, pp 435–440
243. Streichert F, Tanaka-Yamawaki M (2006) The effect of local search on the constrained portfolio selection problem. In: IEEE Congress on Evolutionary Computation, Vancouver, pp 2353–2359
244. Sudholt D (2009) The impact of parametrization in memetic evolutionary algorithms. Theor Comput Sci 410(26):2511–2528
245. Sudholt D (2012) Parametrization and balancing local and global search. In: Neri F, Cotta C, Moscato P (eds) Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol 379. Springer, Berlin/Heidelberg, pp 55–72
246. Sun X, Wang Z, Zhang D (2008) A watermarking algorithm based on MA and DWT. In: IEEE International Symposium on Electronic Commerce and Security, Guangzhou, pp 916–919
247. Tanese R (1989) Distributed genetic algorithms. In: 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Francisco, pp 434–439
248. Ting CK, Liao CC (2010) A memetic algorithm for extending wireless sensor network lifetime. Inform Sci 180(24):4818–4833
249. Tirronen V, Neri F, Kärkkäinen T, Majava K, Rossi T (2008) An enhanced memetic differential evolution in filter design for defect detection in paper production. Evol Comput 16(4):529–555
250. Tomassini M (2005) Spatially Structured Evolutionary Algorithms. Natural Computing Series. Springer, Berlin/New York

251. Tse S, Liang Y, Leung K, Lee K, Mok S (2005) Multi-drug cancer chemotherapy scheduling by a new memetic optimization algorithm. In: IEEE Congress on Evolutionary Computation (CEC 2005), Edinburgh, pp 699–706
252. Tse SM, Liang Y, Leung KS, Lee KH, Mok TSK (2007) A memetic algorithm for multiple-drug cancer chemotherapy schedule optimization. IEEE Trans Syst Man Cybern B Cybern 37(1):84–91
253. Urselmann M, Engell S (2010) Optimization-based design of reactive distillation columns using a memetic algorithm. In: Pierucci S, Ferraris B (eds) 20th European Symposium on Computer Aided Process Engineering. Elsevier Science BV, Ischia, vol 28, pp 1243–1248
254. Urselmann M, Engell S (2015) Design of memetic algorithms for the efficient optimization of chemical process synthesis problems with structural restrictions. Comput Chem Eng 72:87–108
255. Urselmann M, Sand G, Engell S (2009) A memetic algorithm for global optimization in chemical process synthesis. In: IEEE Congress on Evolutionary Computation (CEC 2009), Trondheim, pp 1721–1728
256. Urselmann M, Barkmann S, Sand G, Engell S (2011) A memetic algorithm for global optimization in chemical process synthesis problems. IEEE Trans Evol Comput 15(5, SI):659–683
257. Urselmann M, Barkmann S, Sand G, Engell S (2011) Optimization-based design of reactive distillation columns using a memetic algorithm. Comput Chem Eng 35(5):787–805
258. Vesselinov VV, Harp DR (2012) Adaptive hybrid optimization strategy for calibration and parameter estimation of physical process models. Comput Geosci 49:10–20
259. Vidal T, Crainic TG, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. Eur J Oper Res 234(3):658–673
260. Wang G, Wang J, Chen H (2014) Application of operator libraries on laminate strength optimization of composites. In: Yang G (ed) International Conference on Materials Science, Machinery and Energy Engineering (MSMEE 2013). Advanced Materials Research, vol 853. Trans Tech Publications Ltd, Hong Kong, pp 686–692
261. Wang L, Liu J (2013) A scale-free based memetic algorithm for resource-constrained project scheduling problems. In: Yin H, Tang K, Gao Y, Klawonn F, Lee M, Li B, Weise T, Yao X (eds) 14th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2013). Lecture Notes in Computer Science, vol 8206. Springer, Hefei, pp 202–209
262. Wang L, Zheng DZ (2002) A modified genetic algorithm for job-shop scheduling. Int J Adv Manuf Technol 20:72–76
263. Wang Y, Hao JK, Glover F, Lu Z (2014) A tabu search based memetic algorithm for the maximum diversity problem. Eng Appl Artif Intell 27:103–114
264. Wanner EF, Guimaraes FG, Takahashi RHC, Lowther DA, Ramirez JA (2008) Multiobjective memetic algorithms with quadratic approximation-based local search for expensive optimization in electromagnetics. IEEE Trans Mag 44(6):1126–1129
265. Whitley LD (1991) Fundamental principles of deception in genetic search. In: Rawlins G (ed) Foundations of Genetic Algorithms I. Morgan Kaufmann, San Mateo, pp 221–241
266. Widl M, Musliu N (2014) The break scheduling problem: complexity results and practical algorithms. Memetic Comput 6(2):97–112
267. Wolpert D, Macready W (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
268. Xu J, Yin Y, Cheng TCE, Wu CC, Gu S (2014) An improved memetic algorithm based on a dynamic neighbourhood for the permutation flowshop scheduling problem. Int J Prod Res 52(4):1188–1199
269. Yang CH, Cheng YH, Chang HW, Chuang LY (2009) Primer design with specific PCr product size using memetic algorithm. In: IEEE Conference on Soft Computing in Industrial Applications (SMCIA 2008), Muroran, pp 332–337
270. Yang CH, Cheng YH, Chuang LY, Chang HW (2009) Specific PCr product primer design using memetic algorithm. Biotechnol Prog 25(3):745–753

271. Yang S, Wang S, Liu Z, Wang M, Jiao L (2014) Improved bandelet with heuristic evolutionary optimization for image compression. Eng Appl Artif Intell 31(SI):27–34
272. Yang SH, Kiang JF (2014) Optimization of asymmetrical difference pattern with memetic algorithm. IEEE Trans Antennas Propag 62(4, 2):2297–2302
273. Zaman F, Qureshi IM, Munir F, Khan ZU (2014) Four-dimensional parameter estimation of plane waves using swarming intelligence. Chin Phys B 23(7):078402
274. Zhao F, Tang J, Wang J, Jonrinaldi J (2014) An improved particle swarm optimization with decline disturbance index (DDPSO) for multi-objective job-shop scheduling problem. Comput Oper Res 45:38–50
275. Zhou J, Ji Z, Zhu Z, He S (2014) Compression of next-generation sequencing quality scores using memetic algorithm. BMC Bioinform 15(15):S10
276. Zhou M, Liu J (2014) A memetic algorithm for enhancing the robustness of scale-free networks against malicious attacks. Physica A-Stat Mech Appl 410:131–143
277. Zhu GY, Zhang WB (2014) An improved shuffled frog-leaping algorithm to optimize component pick-and-place sequencing optimization problem. Expert Syst Appl 41(15):6818–6829
278. Zhu Z, Ong YS (2007) Memetic algorithms for feature selection on microarray data. In: Liu D, Fei S, Hou Z, Zhang H, Sun C (eds) 4th International Conference on Advances in Neural Networks (ISNN 2007). Lecture Notes in Computer Science, vol 4491. Springer, Berlin/Nanjing, pp 1327–1335
279. Zhu Z, Ong YS, Zurada JM (2010) Identification of full and partial class relevant genes. IEEE-ACM Trans Comput Biol Bioinform 7(2):263–277
280. Zhu Z, Zhou J, Ji Z, Shi YH (2011) DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm. IEEE Trans Evol Comput 15(5, SI):643–658
281. Zibakhsh A, Abadeh MS (2013) Gene selection for cancer tumor detection using a novel memetic algorithm with a multi-view fitness function. Eng Appl Artif Intell 26(4):1274–1281

Particle Swarm Methods

21

Konstantinos E. Parsopoulos

Contents

Introduction . . . 640
Basic Model . . . 642
Convergence and Parameter Setting . . . 645
  Early Precursors . . . 646
  Velocity Clamping and Inertia Weight . . . 647
  Stability Analysis . . . 649
  Concept of Neighborhood . . . 651
  Initialization, Stopping Conditions, and Boundaries Violation . . . 653
  Performance-Enhancing Techniques . . . 655
Enhanced and Specialized PSO Variants . . . 662
  Binary PSO . . . 662
  Guaranteed Convergence PSO . . . 663
  Bare Bones PSO . . . 663
  Fully Informed PSO . . . 664
  Quantum PSO . . . 664
  Unified PSO . . . 665
  Cooperative PSO . . . 667
  Comprehensive Learning PSO . . . 668
  TRIBES . . . 669
  Niching PSO . . . 670
  Standard PSO . . . 671
  Memetic PSO . . . 672
  Opposition-Based PSO . . . 673
  PSO in Noisy Environments . . . 674
  Multiobjective PSO . . . 674
Applications . . . 676
Conclusions . . . 676
Cross-References . . . 677
Appendix . . . 677
References . . . 678

K. E. Parsopoulos (✉)
Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
e-mail: kostasp@cs.uoi.gr

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_22

Abstract

Particle swarm optimization has gained increasing popularity in the past 15 years. Its effectiveness and efficiency has rendered it a valuable metaheuristic approach in various scientific fields where complex optimization problems appear. Its simplicity has made it accessible to the non-expert researchers, while the potential for easy adaptation of operators and integration of new procedures allows its application on a wide variety of problems with diverse characteristics. Additionally, its inherent decentralized nature allows easy parallelization, taking advantage of modern high-performance computer systems. The present work exposes the basic concepts of particle swarm optimization and presents a number of popular variants that opened new research directions by introducing novel ideas in the original model of the algorithm. The focus is placed on presenting the essential information of the algorithms rather than covering all the details. Also, a large number of references and sources is provided for further inquiry. Thus, the present text can serve as a starting point for researchers interested in the development and application of particle swarm optimization and its variants.

Keywords

Particle Swarm Optimization · Swarm Intelligence · Metaheuristics · Nature-Inspired Algorithms · Stochastic Search · Optimization · Computational Intelligence

Introduction

Particle swarm optimization (PSO) was introduced in the pioneering works of Russell C. Eberhart and James Kennedy in [33, 60]. At that time, the wide success of Evolutionary Algorithms (EAs) motivated researchers worldwide to develop and experiment with novel nature-inspired methods. Thus, besides the interest in evolutionary procedures that governed EAs, new paradigms from nature were subjected to investigation. In this context, the hierarchically organized societies of simple organisms such as ants, bees, and fishes, which have limited range of individual responses but fascinating collective behaviors, immediately attracted scientific interest.

Simulations conducted with modeled populations of such organisms exhibited traits of intelligent behavior and remarkable problem solving capabilities [12]. The underlying mathematical models shared concepts with particle physics, enriched with elements from probability theory and stochastic processes. The theoretical background along with the potential for developing powerful optimization metaheuristics resulted in the emergence of a new category of algorithms under the name of *Swarm Intelligence* (SI) [12, 34, 62].

PSO is placed in a salient position among SI algorithms. Its inspiration stemmed from simulators of social behavior that implement rules such as neighbor velocity matching and acceleration by distance. These properties were shown to produce

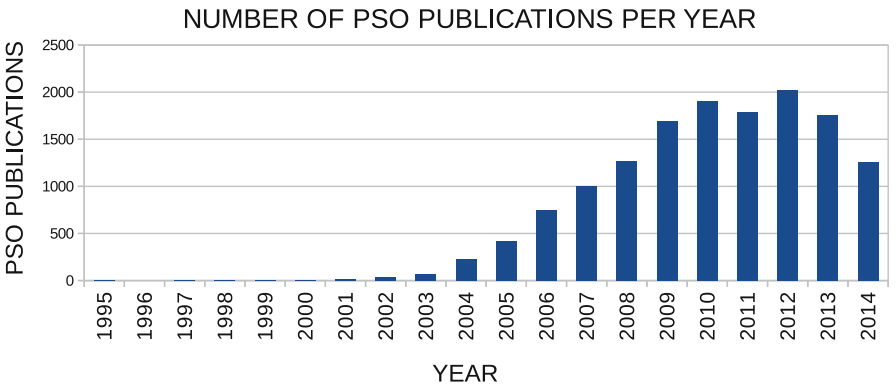


Fig. 1 Number of publications with the term "particle swarm" in the title (Source: Scopus search engine, November 2014)

swarming motives in groups of simple artificial agents. The early models were properly refined to fit the framework of optimization algorithms. The first PSO models introduced the novelty of using difference vectors among population members to sample new points in the search space. This novelty diverged from the established procedures of EAs, which were mostly based on sampling new points from explicit probability distributions [126]. Nevertheless, it proved to be appealing and, almost concurrently with PSO, another algorithm with similar structure, namely, Differential Evolution [131], appeared in the literature. Additional advantages of PSO were its potential for easy adaptation of operators and procedures to match the specific requirements of a given problem, as well as its inherent decentralized structure that promoted parallelization.

PSO has gained wide recognition due to its effectiveness, efficiency, and easy implementation. This is illustrated in Fig. 1, which reports the number of scientific works that include the term "Particle Swarm" in the titles for the years 1995–2014, as returned by the Scopus search engine on a search conducted in November 2014. The illustrated numbers of papers include all sources provided by the Scopus search engine (journals, conferences, books, etc.).

The present work aims at introducing the basic concepts of PSO and presenting a number of its most influential variants, based on the criteria of novelty at the time of development, popularity, and potential for opening new research directions. Naturally, neither the selection of variants can be complete nor the presentation can be thorough within the limited space of a book chapter. For this reason, only basic information is provided, while further details can be acquired in the reported references and sources. Thus, the present text can serve as the "Ariadne's thread" for researchers with interest in PSO.

The rest of the present chapter is organized as follows: section "Basic Model" introduces a basic PSO model that is used as reference point for the presented variants. Section "Convergence and Parameter Setting" outlines the theoretical properties of PSO and probes interesting aspects such as convergence, parame-

ter setting, special features of the algorithm, as well as performance-enhancing techniques. Section "Enhanced and Specialized PSO Variants" presents a number of enhanced and specialized PSO variants that constitute the state-of-the-art, while section "Applications" briefly comments on applications. Finally, the paper concludes in section "Conclusions". The Appendix at the end of the text offers additional sources for further inquiry and experimentation with PSO.

Basic Model

PSO was primarily developed for optimization in real-valued search spaces. For this reason, the general d-dimensional bound-constrained optimization problem,

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}), \tag{1}$$

where,

$$\mathcal{X} = \left[x_1^{\min}, x_1^{\max}\right] \times \cdots \times \left[x_d^{\min}, x_d^{\max}\right] \subset \mathbb{R}^d, \tag{2}$$

is henceforth considered as the reference problem for the presentation of the algorithm. In case of different search spaces, explicit definitions will be given. The only necessary assumption regarding the objective function f is the availability of its value $f(\mathbf{x})$ for every point $\mathbf{x} \in \mathcal{X}$. Smoothness properties such as continuity and differentiability are not required.

The main search mechanism of PSO is based on a group of *search agents*, called *particles*, which iteratively change their position in the search space $\mathcal{X}$. The particles retain in an external memory the best positions they have ever visited in $\mathcal{X}$. The move of each particle is stochastically biased toward its own findings as well as promising regions of the search space discovered by the rest of the particles. This implicit information-exchange scheme is responsible for the emergent convergence properties of the whole group, which is accordingly called a *swarm*.

Putting it formally, let A_i denote the i-th particle (search agent) and,

$$\mathcal{S} = \{A_1, A_2, \ldots, A_N\}, \tag{3}$$

be a swarm of size N (the ordering of the particles is irrelevant). Also, let,

$$I = \{1, 2, \ldots, N\}, \qquad D = \{1, 2, \ldots, d\},$$

be the sets of indices of the particles and the coordinate directions, respectively, and t denote the algorithm's iteration counter (this notation will be henceforth used in the present work). Then, each particle can be defined by four essential elements,

$$A_i^{(t)} = \left\langle \mathbf{x}_i^{(t)}, \mathbf{v}_i^{(t)}, \mathbf{p}_i^{(t)}, NB_i^{(t)} \right\rangle, \qquad i \in I.$$

The first vector,

$$\mathbf{x}_i^{(t)} = \left(x_{i1}^{(t)}, x_{i2}^{(t)}, \ldots, x_{id}^{(t)}\right)^\top \in \mathcal{X},$$

defines the *current position* of the particle in $\mathcal{X}$ at iteration t. In many works in literature, the term "particle" is used to define solely the current position $\mathbf{x}_i$. However, in the author's opinion, the use of this term to define the search agent with all its components promotes a more compact presentation, and it is aligned with notation in similar SI algorithms (e.g., the term "ant" is similarly used to define search agents in ant colony optimization [12]). The second vector,

$$\mathbf{v}_i^{(t)} = \left(v_{i1}^{(t)}, v_{i2}^{(t)}, \ldots, v_{id}^{(t)}\right)^\top,$$

is an adaptable position shift, also called *velocity*, which is responsible for the particle's move. The third vector,

$$\mathbf{p}_i^{(t)} = \left(p_{i1}^{(t)}, p_{i2}^{(t)}, \ldots, p_{id}^{(t)}\right)^\top \in \mathcal{X},$$

is the particle's *best position*, i.e., the best position it has ever visited in $\mathcal{X}$ up to iteration t. For the reference optimization problem of Eq. (1), the best position corresponds to the particle's previous position with the smallest objective value, i.e.,

$$\mathbf{p}_i^{(t)} = \arg \min_{\left\{\mathbf{x}_i^{(k)},\, k \leqslant t\right\}} f\left(\mathbf{x}_i^{(k)}\right).$$

The best position plays the role of attractor that biases the velocity toward the discovered promising regions of $\mathcal{X}$. However, this sole information would render the particle an isolated agent with limited search capability that produces trajectories by oscillating around its best visited points.

For this reason, in addition to its own discoveries, each particle has an implicit information-exchange mechanism with a subset of the swarm, called its *neighborhood*. The neighborhood can be formally denoted as a subset of the indices set I,

$$NB_i^{(t)} \subseteq I,$$

and its cardinality is usually called the *neighborhood size*. In most PSO variants, the best position in the neighborhood of the particle, i.e.,

$$\mathbf{p}_{g_i}^{(t)} = \arg \min_{\left\{\mathbf{p}_j^{(t)},\, j \in NB_i^{(t)}\right\}} f\left(\mathbf{p}_j^{(t)}\right),$$

is used as a second attractor for biasing its velocity.

Based on the definitions above, at each iteration of the algorithm, the particles componentwisely update their current positions and velocities as follows:

$$v_{ij}^{(t+1)} = \chi\, v_{ij}^{(t)} + \mathcal{C}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + \mathcal{C}_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right), \tag{4}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{5}$$

where $i \in I$ and $j \in D$. The scalar parameter χ is called the *inertia weight* or *constriction coefficient*, and its role is discussed in section "Convergence and Parameter Setting". The stochastic terms $\mathcal{C}_1$ and $\mathcal{C}_2$ follow continuous uniform distributions,

$$\mathcal{C}_1 \sim \mathcal{U}(0, c_1), \qquad \mathcal{C}_2 \sim \mathcal{U}(0, c_2), \tag{6}$$

where c_1 and c_2 are user-defined parameters, called the *acceleration constants*, which control the magnitude of attraction toward $\mathbf{p}_i^{(t)}$ and $\mathbf{p}_{g_i}^{(t)}$, respectively. The first difference term $\left(p_{ij}^{(t)} - x_{ij}^{(t)} \right)$ is called the *cognitive term* because it involves only the particle's own information. Correspondingly, the second difference term $\left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right)$ is called the *social term* since it involves information provided by other particles.

After the current positions and velocities, the best positions are also updated as follows:

$$\mathbf{p}_i^{(t+1)} = \begin{cases} \mathbf{x}_i^{(t+1)}, & \text{if } f\left(\mathbf{x}_i^{(t+1)}\right) \leqslant f\left(\mathbf{p}_i^{(t)}\right), \\ \mathbf{p}_i^{(t)}, & \text{otherwise,} \end{cases} \qquad i \in I. \tag{7}$$

The algorithm is executed until a predefined stopping condition is fulfilled. The presented basic PSO model is also known as the *Canonical Particle Swarm* [59], and it is summarized in Algorithm 1. The discussion on initialization, boundary violation, and stopping conditions is postponed until the next section.

A number of issues arise regarding essential decisions that need to be made prior to the application of the basic PSO algorithm. Specifically, the practitioner shall address the following questions:

1. How shall the parameters N, χ, c_1, c_2, be set?
2. How shall the neighborhoods be defined?
3. How shall the boundaries violations be handled?
4. How shall the swarm be initialized?
5. What stopping conditions shall be used?

Algorithm 1: Canonical PSO

Require: Initialize PSO algorithm.

while (not stopping condition) **do**
  for $i = 1$ to N **do**
    for $j = 1$ to d **do**
      Update v_{ij} by using Eq. (4).
      Check for velocity boundaries violation and correct if needed.
      Update x_{ij} using Eq. (5).
      Check for search space boundaries violation and correct if needed.
    end for
  end for
  for $i = 1$ to N **do**
    Update $\mathbf{p}_i$ according to Eq. (7).
  end for
end while
Report best solution.

Apparently, each decision can have significant impact on the algorithm's performance. For this reason, careful settings that take into consideration the potential impact on PSO's dynamic are of vital importance.

A number of PSO variants have been developed on the basis of using alternative techniques for configuring the algorithm. These issues are discussed in the next section. Canonical PSO has constituted the starting point for further developments and enhancements for almost two decades. Yet, it still remains a popular piece of the state-of-the-art due to its verified effectiveness and efficiency in a plethora of applications [106, 112, 127].

Convergence and Parameter Setting

During the last two decades of PSO's development, a large number of variants have been proposed. In almost all cases, the motivation for further research has stemmed from the necessity for more efficient algorithms that tackle the weaknesses of previous variants. In the core of these developments lie the answers of the basic questions posed in section "Basic Model" with respect to PSO's configuration and parameter setting.

In the following paragraphs, the most significant developments are highlighted along with insights and suggestions for PSO's configuration.

Early Precursors

The early PSO precursors [33, 60] were not as sophisticated as the Canonical PSO approach presented in section "Basic Model". In fact, they were based on a simplified version of Eq. (4) with $\chi = 1$ and $NB_i = I$, i.e.,

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} + \mathcal{C}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + \mathcal{C}_2 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right), \tag{8}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{9}$$

with $i \in I$, $j \in D$, and $\mathbf{p}_g^{(t)}$ be the best position discovered by any particle (also called the swarm's *overall best*) up to iteration t, i.e.,

$$\mathbf{p}_g^{(t)} = \arg \min_{\left\{ \mathbf{p}_k^{(t)},\, k \in I \right\}} f\left(\mathbf{p}_k^{(t)} \right). \tag{10}$$

Obviously, the magnitude of the position shifts depends on the values of the parameters c_1 and c_2 that determine the stochastic terms $\mathcal{C}_1$ and $\mathcal{C}_2$ according to Eq. (6).

In [56] the trajectories of the particles were studied by simplifying the system. The use of the previous velocity $\mathbf{v}_i^{(t)}$ in the update equation of the current velocity results in an oscillatory move of the particle around the weighted average of the two best positions,

$$\bar{\mathbf{p}}_i^{(t)} = \frac{1}{\mathcal{C}_1 + \mathcal{C}_1} \left(\mathcal{C}_1 \mathbf{p}_i^{(t)} + \mathcal{C}_2 \mathbf{p}_g^{(t)} \right). \tag{11}$$

The swarm's overall best particle, for which it holds that $\mathbf{p}_i^{(t)} = \mathbf{p}_g^{(t)}$, updates its velocity as follows:

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} + \mathcal{C} \left(p_j^{(t)} - x_{ij}^{(t)} \right), \tag{12}$$

where $j \in D$, $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$, and $\mathbf{p}^{(t)} = \mathbf{p}_g^{(t)}$.

If we consider the trivial case of a single particle and a fixed best position $\mathbf{p}$, it was shown in [56] that the particle's trajectory becomes highly dependent on the values of $\mathcal{C}$, as shown in Fig. 2. Values higher than $\mathcal{C} = 4.0$ are detrimental for the algorithm's convergence, since the velocity can grow arbitrarily large (notice the scaling difference in the vertical axis for $\mathcal{C} = 4.0$ in Fig. 2). This problem was called the *swarm explosion effect*, and it was verified also in [87, 88], emphasizing the necessity for a mechanism to control the amplitude of the velocities.

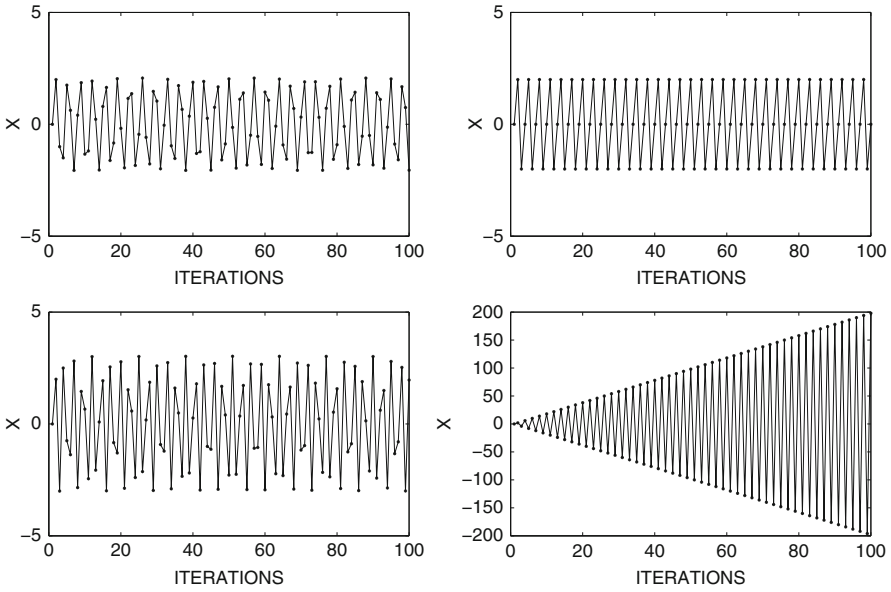


Fig. 2 Particle trajectory for fixed value $\mathcal{C} = 2.5$ (*upper left*), 3.0 (*upper right*), 3.5 (*lower left*), and 4.0 (*lower right*)

Velocity Clamping and Inertia Weight

The swarm explosion effect led to the first significant improvements of PSO, namely *velocity clamping* and the introduction of the *inertia weight*. Since the main problem was the arbitrary growth of the velocities, a straightforward solution was their explicit restriction in acceptable bounds. Specifically, a maximum value $v_j^{\max}$ was imposed on the absolute value of each velocity component, i.e.,

$$-v_j^{\max} \leqslant v_{ij} \leqslant v_j^{\max}, \quad \text{for all } \; i \in I, j \in D.$$

This way, if a component v_{ij} violates one of the boundaries, it is set equal to the violated boundary's value, prohibiting the particle from taking large steps away from the weighted average of best positions of Eq. (11). The user-defined value $v_j^{\max}$ is usually equal to a fraction of the search space along the j-th coordinate direction, i.e.,

$$v_j^{\max} = \lambda_j \left(x_j^{\max} - x_j^{\min} \right), \qquad j \in D, \; \lambda_j \in (0, 1). \tag{13}$$

Naturally, prior information regarding the search space facilitates proper setting of the maximum velocity. For example, in cases of extremely large number of minimizers or very narrow regions of attraction around them, smaller velocities

can offer better search accuracy [129]. Equation (13) is used also in modern PSO variants, usually assuming identical λ_j values for all $j \in D$. Velocity clamping is applied in line 5 of the Canonical PSO in Algorithm 1.

Although velocity clamping was effective in hindering divergence, it was proved to be inadequate to produce convergent behavior of the particles. Obviously, convergence to a point in the search space requires the gradual decrease of velocities such that the particles can perform decreasing oscillations around the attractors and, eventually, settle on a point. This was achieved by introducing an inertia weight w [128] on the velocity update of Eq. (8), i.e.,

$$v_{ij}^{(t+1)} = w v_{ij}^{(t)} + \mathcal{C}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + \mathcal{C}_2 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right), \tag{14}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}. \tag{15}$$

The parameter w shall be properly set such that the impact of the previous velocity $\mathbf{v}_i^{(t)}$ declines during the execution of the algorithm. This can be achieved by setting either a fixed value $w \in (0, 1)$ or assuming a decreasing value between two extreme values $w_{\max}$ and $w_{\min}$, during the algorithm's run. For example, if $t_{\max}$ is a predefined maximum number of iterations, the most common linearly decreasing inertia weight takes values as follows,

$$w^{(t)} = w_{\max} - \frac{t}{t_{\max}} \left(w_{\max} - w_{\min} \right).$$

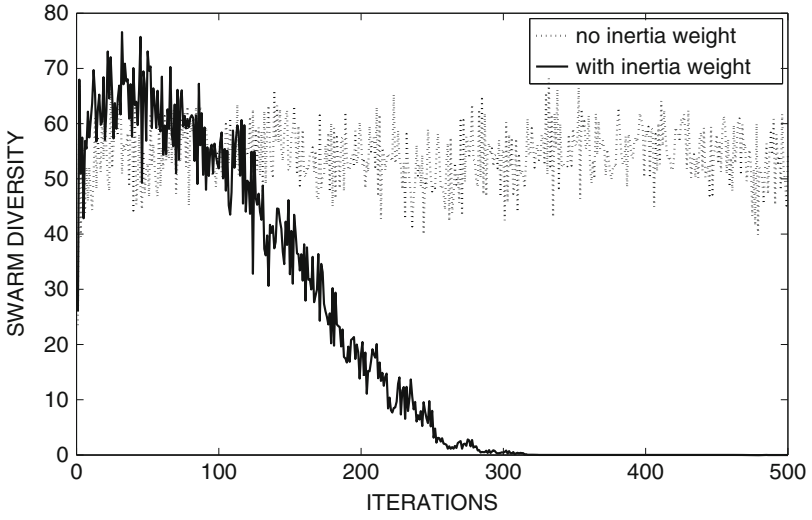


Fig. 3 Swarm's diversity with (*solid line*) and without (*dotted line*) inertia weight while minimizing the 2-dimensional Rosenbrock function using 20 particles with $c_1 = c_2 = 2$, $v^{\max} = 50$, $w_{\max} = 1.2$, and $w_{\min} = 0.1$

Figure 3 illustrates the swarm's diversity in terms of the average of the particles' standard deviations per coordinate direction, without inertia weight as well as with a linearly decreasing inertia weight, for a well known optimization problem.

Velocity clamping and the introduction of inertia weight boosted PSO research due to the improved convergence properties of the derived PSO variants. Both these improvements are still used in modern PSO approaches [106]. Further information and enhancements can be found in recent works such as [15, 155].

Stability Analysis

The empirical analyses of particles' trajectories in [56,87,88] and their implications in PSO's modeling and parameter setting motivated the systematic theoretical investigation of the algorithm. The first breakthrough appeared in 2002 due to M. Clerc and J. Kennedy who conducted a convergence and stability analysis of the algorithm in multidimensional search spaces [21].

The authors used as starting point the early model of Eqs. (8) and (9) with velocity clamping. The initial model was manipulated according to Eq. (12) and further simplified by assuming 1-dimensional particles without stochasticity, i.e., the two stochastic acceleration terms were assumed to be equal and fixed, $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$. After dropping some indices and making proper algebraic manipulations, the 1-dimensional swarm's update rules can be rewritten in the form [21],

$$v^{(t+1)} = v^{(t)} + \mathcal{C} y^{(t)}, \tag{16}$$

$$y^{(t+1)} = -v^{(t)} + (1 - \mathcal{C}) y^{(t)}, \tag{17}$$

where $y^{(t)} = \bar{p} - x^{(t)}$, with $\bar{p}$ being the (currently fixed) aggregate best position defined in Eq. (11) (note that all vectors currently collapse to scalars). This discrete system can be written also in matrix form,

$$P_{t+1} = M\, P_t = M^t\, P_0,$$

where,

$$P_t = \begin{pmatrix} v^{(t)} \\ y^{(t)} \end{pmatrix}, \qquad M = \begin{pmatrix} 1 & \mathcal{C} \\ -1 & 1 - \mathcal{C} \end{pmatrix}.$$

Then, the behavior of the system depends on the eigenvalues of M, which are given by,

$$\lambda_{1,2} = 1 - \frac{\mathcal{C}}{2} \pm \frac{\sqrt{\mathcal{C}^2 - 4\mathcal{C}}}{2}.$$

The previously identified critical value $\mathcal{C} = 4$ (recall Fig. 2) [56, 88], appears again as the limit between the case of two different real eigenvalues, one eigenvalue of multiplicity 2, and two complex conjugate eigenvalues $\lambda_{1,2}$.

For the case of $\mathcal{C} \in (0, 4)$, the eigenvalues become complex [21],

$$\lambda_1^t = \cos(t\theta) + i\ \sin(t\theta), \qquad \lambda_2^t = \cos(t\theta) - i\ \sin(t\theta),$$

and the system exhibits cyclic behavior for $\theta = (2\kappa\pi)/t$. On the other hand, values of $\mathcal{C} > 4$ produce no cyclic behavior and it is proved that P_t has monotonically increasing distance from the origin [21]. Finally, the limit case $\mathcal{C} = 4$ produces either oscillatory behavior, i.e., $P_{t+1} = -P_t$, if P_0 is an eigenvector of M, or linearly increasing or decreasing $\|P_t\|$ for $y_0 > 0$ or $y_0 < 0$, respectively.

The investigation was further extended to the continuous case by transforming the simplified model into the recurrent equation [21],

$$\mathbf{v}^{(t+2)} + (\mathcal{C} - 2)\mathbf{v}^{(t+1)} + \mathbf{v}^{(t)} = 0,$$

which in turn becomes a second-order differential equation,

$$\frac{\partial^2 \mathbf{v}}{\partial t^2} + \ln(\lambda_1\lambda_2)\frac{\partial \mathbf{v}}{\partial t} + \ln(\lambda_1)\ln(\lambda_2)\mathbf{v} = 0,$$

where λ_1 and λ_2 are the solutions of the polynomial,

$$\lambda^2 + (\mathcal{C} - 2)\lambda + 1 = 0.$$

Thus, the quantities $\mathbf{v}^{(t)}$ and $\mathbf{y}^{(t)}$ assume the general form [21],

$$\begin{aligned}\mathbf{v}^{(t)} &= c_1\lambda_1^t + c_2\lambda_2^t,\\ \mathbf{y}^{(t)} &= \left(c_1\lambda_1^t(\lambda_1 - 1) + c_2\lambda_2^t(\lambda_2 - 1)\right)/\mathcal{C}.\end{aligned}$$

The parameters c_1 and c_2 depend on the initial vectors $\mathbf{v}^{(0)}$ and $\mathbf{y}^{(0)}$. Similar analysis with the discrete case shows that the system's explosion depends on whether the condition,

$$\max\{|\lambda_1|, |\lambda_2|\} > 1,$$

holds or not [21].

After the analysis above, the employed simplified model was generalized in order to approximate the actual model of PSO. A number of extended models were accordingly developed and studied in [21]. The outcome of the study was an effective model, namely the Canonical PSO model of Eqs. (4) and (5), accompanied with a proposed parameter setting [21],

$$\chi = 0.729, \quad c_1 = c_2 = 1.49, \tag{18}$$

derived from closed-form formulae in order to retain the convergent behavior of the system. This parameter setting has become the most common choice for off-the-shelf PSO approaches in numerous applications [106].

The first theoretically sound convergence analysis of PSO in [21] was succeeded by a number of theoretical studies [49, 54, 113, 124, 143, 147]. These developments increased our understanding of PSO's dynamic. Also, they added merit that, along with its easy implementation and the reported excellent results in various applications, eventually placed PSO in a salient position among other established population-based optimization algorithms such as Genetic Algorithms and Evolution Strategies. Moreover, the dynamic of the Canonical PSO model motivated recent approaches that spare function evaluations by taking advantage of the particles' dynamic [148].

Concept of Neighborhood

Information exchange among the particles is a concept of major importance in PSO. In the Canonical PSO of Eqs. (4) and (5), the i-th particle adopts the best position $\mathbf{p}_{g_i}$ of its neighborhood NB_i as an attractor for its move, besides its own best position. This way, the neighborhood determines the communication channels among particles and, consequently, the information flow within the swarm. Without communication, collective behavior cannot emerge and the swarm is downgraded to a group of isolated search agents with limited search capabilities.

Obviously, the neighborhood's characteristics have direct impact on PSO's dynamic [57, 132]. Its structure defines the communication channels among particles, while its size controls the influence of the swarm on each particle. Since the particles are attracted toward the best positions of their neighbors, it is easily comprehended that neighborhoods with large number of particles and dense communication channels are more inclined toward intensification of search around the best detected positions in the search space. This is ascribed to the rapid diffusion of the discovered good solutions to the particles and their consequent attraction toward them.

However, this also renders the particles prone to get stuck in deep local minima that are possibly detected in early steps of the algorithm's execution. Such minima are typically updated less frequently than shallow ones. Hence, they can impose stronger attraction on the particles. On the other hand, less crowded neighborhoods with sparse connections among particles exhibit slower information diffusion in the swarm. In this case, the particles are gradually acquainted with good solutions, retaining higher diversity and exploration capability.

The discussion above suggests that neighborhood's configuration is highly responsible for the diversification/intensification (a.k.a. exploration/exploitation) trade-off of the algorithm. For this reason, it shall be carefully selected. To this end, prior knowledge on the studied problem can be valuable. For example, smooth functions with one or few minima can be properly handled through intensification-

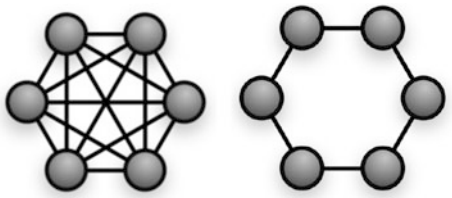


Fig. 4 Graphical representation of the fully connected (*left*) and the ring (*right*) neighborhood topology

oriented neighborhoods. On the other hand, rugged landscapes with a plethora of minimizers dispersed at distant parts of the search space usually require exploration-oriented approaches.

In early PSO variants, each particle was assumed to be connected with all the rest, i.e.,

$$NB_i^{(t)} = I, \quad \text{for all } \ i \in I,$$

and the best position for all neighborhoods was the overall best of the swarm, as defined in Eq. (10). Also, the neighborhoods were time-invariant, i.e., they remained unchanged throughout the execution of the algorithm (hence we can neglect t in $NB_i^{(t)}$ notation). This PSO model is also called the *global PSO model* or simply the *gbest model*. The connection scheme among the particles can be elegantly represented by undirected graphs, where nodes denote the particles and edges denote communication channels. The corresponding structure is called the *neighborhood's topology*. Apparently, the gbest model corresponds to a fully connected graph since all particles communicate with each other. This topology is illustrated in the left part of Fig. 4.

The tendency of the best model to rapidly converge toward the most promising detected solutions and easily get stuck in local minima led to further experimentation with sparsely connected models. The outcome of these efforts was the *local PSO model* or *lbest model*, which assumes neighborhoods that are proper subsets of I, i.e.,

$$NB_i \subset I, \quad \text{for all } \ i \in I.$$

The most effective and easily implemented topology of this form is the *ring*, which is depicted in the right part of Fig. 4. In this scheme, the particles are assumed to lie on a ring according to their indices, i.e, neighboring particles have neighboring indices. Then, each particle is set to communicate only with its immediate neighbors on the ring. The number of neighbors for each direction (front and back) is called the *neighborhood's radius*. Thus, a neighborhood of radius $r < N$ of the i-th particle is defined as the set,

$$NB_i = \{i - r, i - r + 1, \ldots, i - 1, i, i + 1, \ldots, i + r - 1, i + r\},$$

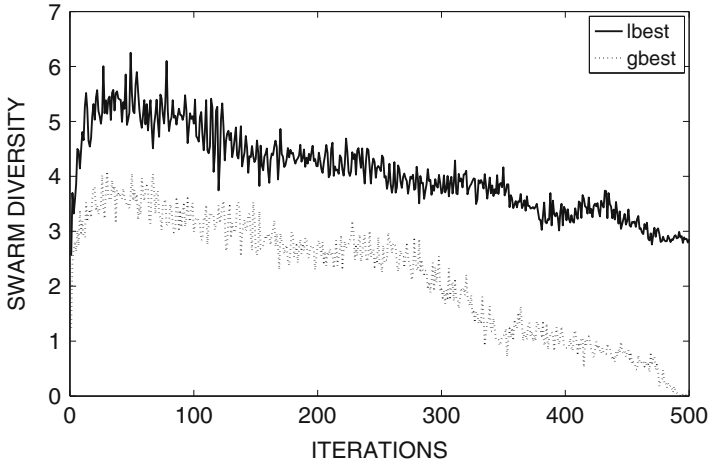


Fig. 5 Swarm's diversity for the lbest (*solid line*) and gbest (*dotted line*) PSO model on the 10-dimensional Rastrigin function using 20 particles

where the indices are assumed to recycle after index N, i.e.,

$$j = \begin{cases} j, & \text{if } \ 1 \leqslant j \leqslant N, \\ j \bmod N, & \text{if } \ j > N, \\ N - |j|, & \text{if } \ j < 1, \end{cases} \qquad \text{for all } \ j \in NB_i.$$

The diversity differences between the gbest model and the lbest with ring topology of radius $k = 1$ is indicatively depicted in Fig. 5 for a simple run on a well-known optimization test problem.

The ring topology has become a standard for lbest PSO implementations due to its successful application in various problems [106]. There is also a multitude of alternative topologies that have drawn attention, such as random [132], hierarchical [46], and dynamic [18], but with limited number of applications. Also, the concept of neighborhood has recently offered the ground for the development of new PSO variants with sophisticated computational budget allocation based on information carried by neighborhoods rather than the particles [130].

Initialization, Stopping Conditions, and Boundaries Violation

A basic requirement in the design and development of stochastic optimization algorithms such as PSO is their tolerance on perturbations of the initial conditions. In practice, this implies that mild perturbations on the initial positions of the particles shall correspond to similar performance profiles of the algorithm.

In benchmarking studies, it is commonly assumed that there is no information available regarding the optimization problem at hand. This is usually called the *black-box optimization problem*. In such cases, there is no reason for purposely biasing the particles' initialization in specific areas of the search space. Consequently, random and uniform initialization of the particles in the search space is the typical procedure followed in most studies, i.e.,

$$x_{ij}^{(0)} = p_{ij}^{(0)} \sim \mathcal{U}\left(x_j^{\min}, x_j^{\max}\right), \qquad \text{for all } \; i \in I, j \in D.$$

Accordingly, the velocities are randomly initialized as,

$$v_{ij}^{(0)} \sim \mathcal{U}\left(-v_j^{\max}, v_j^{\max}\right), \qquad \text{for all } \; i \in I, j \in D.$$

Naturally, if there is prior information regarding promising regions of the search space, the distributions can be properly adapted to favor the sampling of particles in these regions.

The effect of initialization has been studied in a number of works for different PSO variants [25, 31, 36, 100, 137, 157], offering further insight and suggestions in specific applications. Nevertheless, random initialization remains the standard approach also due to its minor implementation effort and the availability of uniform random number generators in almost all hardware platforms.

In contrast to initialization, which is based on the problem's characteristics, the termination conditions are rather user- and resources-dependent [106]. The following are the most common termination criteria:

1. Convergence in search space.
2. Convergence in function values.
3. Limitations in computational budget.
4. Search stagnation.

The first two criteria entail information on the position of the global minimizer or its value, respectively. Naturally, this information is generally unavailable. However, in some cases there are estimated bounds for the solution (or its value) and the user can set the algorithm to stop as soon as it reaches these bounds with a prespecified tolerance. In this case, the underlying stopping condition takes the form,

$$\texttt{IF} \quad \left(\left\| \mathbf{p}_g^{(t)} - \mathbf{x}^* \right\| \leq \varepsilon_x \quad \texttt{OR} \quad \left| f_g^{(t)} - f^* \right| \leq \varepsilon_f \right) \texttt{ THEN STOP}$$

where $\mathbf{x}^*$ and f^* are the targets in the search space and function values, respectively, and ε_x, ε_f, are the corresponding user-defined tolerances.

The other two stopping criteria are more common in PSO's literature. Computational budget limitations are typically imposed by the maximum available time that can be spent for solving a problem. This quantity can be explicitly expressed either

as wall-clock/CPU time or as the number of function evaluations performed by the algorithm. For benchmarking purposes, the latter is preferable since even the same algorithm, executed on the same machine at different time instances, may result in different running times due to irrelevant procedures that may be concurrently executed on the machine.

On the other hand, the search stagnation criterion can prematurely stop the algorithm even if the computational budget is not exceeded. Successful application of this criterion is based on the existence of a proper stagnation measure. Typically, the number of subsequent iterations without improvement of the best solution and/or the dispersion of the particles' current (or best) positions in the search space have been used as indicators of search stagnation.

Frequently, the aforementioned termination criteria are combined in forms such as,

$$\texttt{IF} \quad \left(t \geqslant t_{\max} \quad \texttt{OR} \quad t_{\text{fail}} \geqslant t_{\text{maxfail}} \quad \texttt{OR} \quad d\,v^{(t)} < d\,v_{\min}\right) \texttt{ THEN STOP}$$

where t stands for the iteration number, t_{fail} is the number of subsequent non-improving iterations, and $d\,v^{(t)}$ is a measure of dispersion of the particles (e.g., the average standard deviation per coordinate component). Moreover, the user shall pay special attention to the selection of stopping criteria in order to avoid unintentional premature termination of the algorithm. Recent developments on this issue can be found in [67].

Another topic of interest is the handling of search space boundaries violations. Specifically, after the update of a particle's current position with Eqs. (4) and (5), there is a possibility that some of the new position's components violate the corresponding boundaries of the search space. The most common approach to restrict the particle in the search space is to set the violated components of the particle equal to the value of the violated boundary, i.e.,

$$x_{ij}^{(t)} = \begin{cases} x_j^{\max}, & \text{if } x_{ij}^{(t)} > x_j^{\max}, \\ x_j^{\min}, & \text{if } x_{ij}^{(t)} < x_j^{\min}, \\ x_{ij}^{(t)}, & \text{otherwise}, \end{cases} \tag{19}$$

while simultaneously setting the corresponding velocity component $v_{ij}^{(t)}$ to zero. Different boundary handling techniques (absorbing, bouncing, cyclic search spaces) have been proposed, although with less popularity. A recent survey on such techniques can be found in [89].

Performance-Enhancing Techniques

The Canonical PSO has offered satisfactory performance in various problems. However, it may exhibit declining performance in special problems such as the detection of multiple local/global minima, constrained, or discrete problems. In

such cases, the user can either change the algorithm by introducing new, specialized operators or incorporate external techniques to tackle the problem's peculiarities.

There are established techniques that have been successfully used with Canonical PSO. Transformations of the objective function have been used for the alleviation of local minima and the detection of multiple minimizers. Rounding has been used for solving discrete optimization problems, while penalty functions can address constrained optimization problems. In the following paragraphs, basic techniques that have been combined with the Canonical PSO model are presented. Specialized variants of the algorithm with ad hoc operators for similar purposes are presented in subsequent sections.

Alleviating Local Minimizers

In problems with a multitude of local and/or global minimizers, stochastic optimization algorithms such as PSO may approximate a different minimizer in each independent run. Frequently, the detected solutions are sub-optimal, while the user either needs more than one such solution or requires the globally best one. *Multistart techniques* where the algorithm is subsequently restarted from different initial conditions have been proposed for these cases and discussed in classical optimization texts [142]. However, these approaches cannot guarantee that an already detected minimizer will be avoided after restarting the algorithm.

An alternative approach consists of transforming the objective function into a new one that excludes the already detected solutions. Well-known examples of this type are the *filled functions* [37]. Similar techniques have been recently developed and successfully used with PSO [95, 100]. The *Stretching* technique consists of a two-stage transformation of the objective function [95],

$$F_1(\mathbf{x}) = f(\mathbf{x}) + \gamma_1 \|\mathbf{x} - \mathbf{x}^*\| \left[1 + \text{sign}\left(f(\mathbf{x}) - f(\mathbf{x}^*)\right)\right], \tag{20}$$

$$F_2(\mathbf{x}) = F_1(\mathbf{x}) + \gamma_2 \frac{1 + \text{sign}\left(f(\mathbf{x}) - f(\mathbf{x}^*)\right)}{\tanh\left(\mu\left(F_1(\mathbf{x}) - F_1(\mathbf{x}^*)\right)\right)}, \tag{21}$$

where $f(\mathbf{x})$ is the original objective function, $\mathbf{x}^*$ is the best detected solution of the algorithm so far and,

$$\text{sign}(\mathbf{z}) = \begin{cases} -1, & \text{if } \mathbf{z} < 0, \\ 0, & \text{if } \mathbf{z} = 0, \\ +1, & \text{if } \mathbf{z} > 0, \end{cases}$$

is the three-valued sign function. As soon as a local minimizer (or generally a sub-optimal solution) is detected, the transformation $F_1(\mathbf{x})$ stretches the objective function upward, while $F_2(\mathbf{x})$ transforms the detected solution $\mathbf{x}^*$ into a local maximizer. Under proper parameter setting, Stretching has the ability to remove higher local minima than the detected solution $\mathbf{x}^*$, while leaving unchanged all lower minima as well as the global one. This is illustrated in Fig. 6 for an 1-dimensional instance of a well-known test function. Thus, it can be very useful in

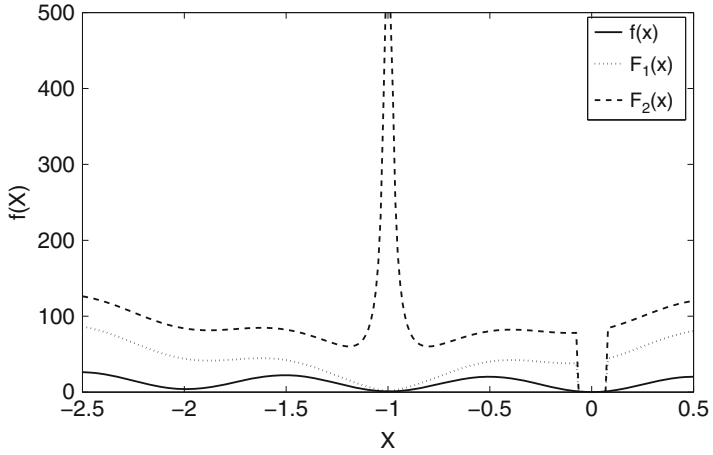


Fig. 6 The Stretching technique applied on the 1-dimensional Rastrigin function for the local minimizer $\mathbf{x}^* = -1$ and parameters $\gamma_1 = \gamma_2 = 20$, $\mu = 0.1$

problems with a multitude of local minima that can mislead the algorithm after its re-initialization. Of course, if a better solution is found in a subsequent application of the algorithm, it can simply replace $\mathbf{x}^*$ in Eqs. (20) and (21).

However, if Stretching is applied on a global minimizer, all other minimizers vanish. For this reason, Stretching is inappropriate for detecting multiple global minimizers (see next section for a relevant technique that tackles this problem). Also, similarly to its filled functions predecessors, Stretching may introduce new local minima around the point of application $\mathbf{x}^*$ [101]. This is also known as the *mexican hat effect* and has been addressed in [153] through proper parameterization. Naturally, the application of Stretching is not limited to the Canonical PSO. It can be incorporated to any PSO variant or even different algorithms [42].

Detecting Multiple Minimizers

Deflection [78] is a technique that works similarly to Stretching but it has only local effect on the objective function. Thus, it can be applied in cases where multiple (global or local) solutions are needed. Deflection has been used with PSO with promising results in detecting multiple global and/or local minimizers [101].

Let $f(\mathbf{x})$ be the objective function and $\mathbf{x}_1^*, \ldots, \mathbf{x}_k^*$, be k previously detected solutions. Then, Deflection transforms the objective function as follows [101],

$$F(\mathbf{x}) = \frac{f(\mathbf{x})}{\prod_{i=1}^{k} T_i\left(\mathbf{x}, \mathbf{x}_i^*, \lambda_i\right)}, \tag{22}$$

where T_i is defined as,

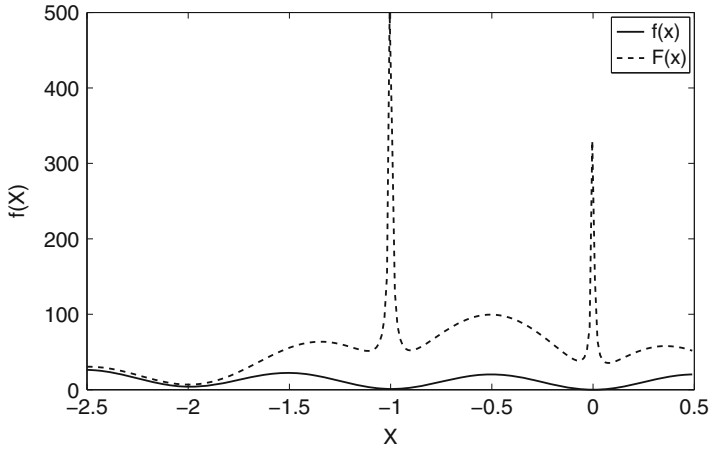


Fig. 7 The Deflection technique applied on the 1-dimensional Rastrigin function for the local minimizer $\mathbf{x}_1^* = -1$ as well as the global minimizer $\mathbf{x}_g^* = 0$ and parameters $\lambda_1 = \lambda_g = 1$

$$T_i\left(\mathbf{x}, \mathbf{x}_i^*, \lambda_i\right) = \tanh\left(\lambda_i \|\mathbf{x} - \mathbf{x}_i^*\|\right), \tag{23}$$

and λ_i are relaxation parameters, $i = 1, 2, \ldots, k$. The idea behind this transformation is the same as in Stretching, i.e., the transformation of detected minimizers into local maximizers. However, Deflection changes the objective function only locally, around the point of application. The magnitude of change depends on the parameters λ_i that need proper tuning. Also, Deflection requires strictly positive objective functions in order to achieve the desirable effect. In cases where the problem at hand is not strictly positive, it can be simply shifted to positive values by using $f(\mathbf{x}) = f(\mathbf{x}) + c$, with a sufficiently large bias $c > 0$, prior to the application of Deflection.

Figure 7 illustrates Deflection for the 1-dimensional instance of the Rastrigin test function. The mexican hat effect appears also in Deflection, although its impact can be controlled with proper selection of the parameters λ_i. Moreover, Deflection can be combined with a *repulsion*
technique that prevents the particles from visiting the neighborhoods of detected solutions and possibly get trapped in the artificially introduced local minima [101, 106]. Combined with PSO, Deflection offered previously undetected solutions in computationally demanding optimization problems [129].

Penalty Functions for Constrained Optimization Problems

Constrained optimization problems are accompanied by a set of constraints that need to be satisfied at the final solution. In general, such a problem can be defined as,

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \quad \text{subject to} \quad C_i(\mathbf{x}) \leqslant 0, \quad i = 1, 2, \ldots, k, \tag{24}$$

where $C_i(\mathbf{x})$ are inequality constraints. Different forms of constraints can be equivalently given in the form above as follows,

$$C_i(\mathbf{x}) \geqslant 0 \Leftrightarrow -C_i(\mathbf{x}) \leqslant 0,$$
$$C_i(\mathbf{x}) = 0 \Leftrightarrow C_i(\mathbf{x}) \geqslant 0 \text{ and } C_i(\mathbf{x}) \leqslant 0.$$

Penalty functions have been widely used in constrained problems. The main goal is to penalize all solutions that lie in the infeasible region, such that the algorithm will be directed again into the search space.

The general form of a penalty function for the problem of Eq. (24) is defined as,

$$f(\mathbf{x}) = f(\mathbf{x}) + P(\mathbf{x}),$$

where,

$$P(\mathbf{x}) = \begin{cases} \alpha > 0, & \text{if } C_i(\mathbf{x}) > 0 \text{ for at least one } i, \\ 0, & \text{otherwise.} \end{cases}$$

In the simplest case, the penalty term $P(\mathbf{x})$ can be constant for all infeasible solutions. However, this choice is not always the most suitable one, since it neglects the degree of violation and does not provide any information to the algorithm regarding the distance of the infeasible solutions from the feasible ones. Thus, the penalty term is recommended to take into consideration the number of violated constraints as well as the degree of violation for every infeasible solution. Moreover, the magnitude of penalties can be time-varying, starting with mild penalties in early stages and becoming strict in the last phase of the optimization procedure.

The Canonical PSO has been combined with such a penalty function defined as follows [98, 159],

$$f(\mathbf{x}) = f(\mathbf{x}) + h(t)H(\mathbf{x}), \tag{25}$$

where,

$$H(\mathbf{x}) = \sum_{i=1}^{k} \theta\left(q_i(\mathbf{x})\right) q_i(\mathbf{x})^{\gamma(q_i(\mathbf{x}))}, \tag{26}$$

and,

$$q_i(\mathbf{x}) = \max\left\{0, C_i(\mathbf{x})\right\}, \qquad i = 1, 2, \ldots, k.$$

The weight $h(t)$ controls the impact of the penalty term $H(\mathbf{x})$ with the number of iterations t. The degree of violation is accounted in $q_i(\mathbf{x})$ and manipulated with the power function $\gamma(q_i(\mathbf{x}))$ as well as with the multi-stage assignment function $\theta\left(q_i(\mathbf{x})\right)$. An alternative penalty function was proposed in [23],

$$f(\mathbf{x}) = f(\mathbf{x}) + H(\mathbf{x}),$$

with,

$$H(\mathbf{x}) = w_1 \, H_{\mathrm{NVC}}(\mathbf{x}) + w_2 \, H_{\mathrm{SVC}}(\mathbf{x}),$$

where $H_{\mathrm{NVC}}(\mathbf{x})$ is the number of violated constraints and,

$$H_{\mathrm{SVC}}(\mathbf{x}) = \sum_{i=1}^{k} \max\left\{0, C_i(\mathbf{x})\right\},$$

is the sum of violations. The weights w_1 and w_2 can be either fixed or time-varying and they determine the importance of each penalty term.

The Canonical PSO updates its best positions according to Eq. (7) by comparing objective values, solely. However, it is commonly desirable to allow only feasible best positions in order to avoid oscillations of the particles in the infeasible space. On top of that, the final solution shall be feasible and, thus, the particles shall eventually concentrate their search efforts in the feasible region. Yet, in the penalty functions defined above, it is still possible that an infeasible solution attains lower value than a feasible one and, therefore, be preferable for inclusion in the best positions.

In order to prevent such undesirable inclusions, a set of rules can be imposed in the selection procedure:

1. Between feasible solutions, the one with the smallest objective value is preferable.
2. Between a feasible and an infeasible solution, the feasible one is always preferable.
3. Between two infeasible solutions, the one with the smallest penalty term $H(\mathbf{x})$ is preferable.

These selection rules along with the penalty function approaches have been used with PSO in various constrained optimization problems with promising results [82, 111].

Tackling Discrete Problems

The Canonical PSO was initially introduced as a continuous optimization method and its operators are designed to work on real-valued search spaces. In Mathematical Programming literature, integer optimization problems can be tackled with continuous algorithms by extending the discrete problem to a continuous one and rounding the solutions to the nearest integers. Such approaches have been used with Branch and Bound algorithms combined with quadratic programming solvers [69, 79].

A similar approach was used also with PSO for solving integer problems [68]. Specifically, the particles are let to assume real vectors in their current positions,

although they are rounded to the nearest integer vector when evaluated with the objective function. Also, the rounded integer vectors are preferable as best positions, since the final solution shall be integer. Putting it formally, if $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\top$ is the current position of the i-th particle then its objective value $f(\mathbf{x}_i) = f(\mathbf{z}_i)$ is evaluated on the integer vector $\mathbf{z}_i = (z_{i1}, z_{i2}, \ldots, z_{id})^\top$ defined as,

$$z_{ij} = \lfloor x_{ij} + 0.5 \rfloor, \qquad i \in I,\ j \in D.$$

Rounding has effectively worked for various problems of integer and mixed integer type [68, 82, 92, 111]. Note that in mixed integer problems, no additional representation scheme is required for the real and the discrete parameters. The algorithm works by simply rounding the coordinate components that correspond to integer variables.

Yet, if PSO's velocities become small enough, it is probable that rounding will result in search stagnation due to the inability of producing different integer components. This potential deficiency can be tackled either by restarting the current positions of the particles (and possibly also the best positions except the overall best one) as soon as stagnation is identified or by applying gradual truncation of the decimal digits of the position vectors of the particles as the number of iterations increases [68].

Another problem category that involves discrete search spaces comprises of *permutation problems*. In such problems, the main goal is the detection of an optimal permutation of fixed elements (e.g., numbers, items, indices, etc.) and they are frequently met in Combinatorics and Operations Research [11]. Such problems can be tackled through real-valued algorithms by using the *smallest position value* (SPV) representation [138]. Specifically, let,

$$\mathcal{Z} = \{Z_1, \ldots, Z_d\}$$

be an ordered set of the elements of interest Z_i, $i \in D$. Then, the components of a real-valued particle are defined as numerical weights that denote the priority of the corresponding discrete elements according to a predefined mapping. For example, the components of the i-th particle $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\top$ can be mapped as follows,

$$\begin{array}{rccccc} \text{ordered list of (discrete) elements:} & Z_1 & Z_2 & \cdots & Z_d \\ & \updownarrow & \updownarrow & & \updownarrow \\ \text{particle's components (weights):} & x_{i1} & x_{i2} & \cdots & x_{id} \end{array}$$

Then, the weights (particle's components) are sorted in ascending order and the corresponding permutation is received by re-arranging the discrete elements accordingly, i.e.,

$$\begin{array}{rccccc} \text{sorted particle's components (weights):} & x_{ik_1} & x_{ik_2} & \cdots & x_{ik_d} \\ & \updownarrow & \updownarrow & & \updownarrow \\ \text{corresponding permutation:} & Z_{k_1} & Z_{k_2} & \cdots & Z_{k_d} \end{array}$$

Thus, each particle corresponds to a permutation received after sorting its components and PSO's goal is to find the appropriate weights that produce optimal or near-optimal permutations. The most common search space in such problems is $\mathcal{X} = [0, 1]^d$, i.e., the weights are all assumed to lie in the range $[0, 1]$. Obviously, there is an infinite number of weight vectors that correspond to a specific permutation, since it depends only on the relative ordering of the weights and not their actual values. SPV has been successfully combined with PSO in various permutation problems [65, 103, 139, 140].

Enhanced and Specialized PSO Variants

This section is devoted to PSO variants that stemmed from the Canonical PSO as improvements or specialized modifications for specific problem types. The pluralism of PSO variants in literature renders a complete presentation impossible. For this reason, a selection is made based on the novelty introduced by each method as well as the influence for further developments. Chronological order of appearance of the methods is partially retained.

Binary PSO

The first *Binary PSO* (BPSO) variant was developed in 1997 [61]. Instead of using rounding for transforming the real-valued parameters into binary ones, the algorithm introduced a new interpretation of the velocities. Specifically, each component of the velocity's vector is considered as the input of a sigmoid that determines the state (0 or 1) of the corresponding particle's position component. Thus, the velocity $\mathbf{v}_i$ of the i-th particle is updated according to Eq. (4), while the current position is updated as follows [61],

$$x_{ij}^{(t)} = \begin{cases} 1, & \text{if } \mathcal{R} < S\left(v_{ij}^{(t)}\right), \\ 0, & \text{otherwise,} \end{cases} \tag{27}$$

for all $i \in I$, $j \in D$, and $\mathcal{R} \sim \mathcal{U}(0, 1)$ is a random variable. The sigmoid is defined as,

$$S(x) = \frac{1}{1 + \exp(-x)},$$

clamping the velocity in the range $[0, 1]$ so it can be translated to probability. The rest of the algorithm follows the basic rules of the Canonical PSO.

The algorithm was demonstrated on a set of test problems with promising results. It also exhibits conceptual similarities with reinforcement learning approaches [73]. BPSO is still considered as the basis for the development of modern and ad hoc binary PSO variants for specific applications [9, 14, 52, 116, 158].

Guaranteed Convergence PSO

The *Guaranteed Convergence PSO* (GCPSO) was introduced in 2002 [145] based on the observation that small velocities can prohibit the convergence of the gbest PSO model with inertia weight. This is ascribed to the cancellation of the attraction forces of the overall best particle toward its best positions, along with small velocities that essentially immobilize it in the long run.

The problem was solved by modifying the position update of the overall best particle (denoted with the index g), as follows [145],

$$x_{gj}^{(t+1)} = p_{gj}^{(t)} + \chi\, v_{gj}^{(t)} + \rho^{(t)}(1 - 2\mathcal{R}), \tag{28}$$

where $j \in D$, $\rho^{(t)}$ is a *scaling factor*, and $\mathcal{R} \sim \mathcal{U}(0, 1)$ is a random variable. The rest of the particles are updated with the standard rules of Eqs. (4) and (5). The scaling factor is dynamically adjusted based on the number of consecutive successes and failures in improving the overall best, i.e.,

$$\rho^{(t+1)} = \begin{cases} 2\,\rho^{(t)}, & \text{if } m_{\text{suc}}^{(t)} > T_{\text{suc}}, \\ 0.5\,\rho^{(t)}, & \text{if } m_{\text{fail}}^{(t)} > T_{\text{fail}}, \\ \rho^{(t)}, & \text{otherwise}, \end{cases}$$

where $m_{\text{suc}}^{(t)}$ and $m_{\text{fail}}^{(t)}$ are counters of the consecutive successes and failures at iteration t, respectively, and T_{suc}, T_{fail}, are the corresponding user-defined thresholds. The counters are reset whenever the row of consecutive successes or failures is interrupted by a failure or success, respectively. Convergence of GCPSO was proved and the values $\rho^{(0)} = 1$, $T_{\text{suc}} = 15$, and $T_{\text{fail}} = 5$, were recommended in [145], where GCPSO was shown to be very promising on typical benchmark problems.

Bare Bones PSO

The *Bare Bones PSO* (BBPSO) was introduced in 2003 [58] as a simplification of the Canonical PSO. Its main feature is the elimination of the velocity update of Eq. (4) and its replacement with Gaussian sampling around the best positions. Specifically, the current position is updated by sampling the Gaussian distribution,

$$x_{ij}^{(t+1)} \sim \mathcal{N}\left(\mu_{ij}^{(t)}, \sigma_{ij}^{2\,(t)}\right), \tag{29}$$

where,

$$\mu_{ij}^{(t)} = \frac{1}{2}\left(p_{gj}^{(t)} + p_{ij}^{(t)}\right), \qquad \sigma_{ij}^{2\,(t)} = \left|p_{gj}^{(t)} - p_{ij}^{(t)}\right|,$$

and g stands for the index of the overall best particle. The rest of the Canonical PSO's procedures (such as best position update and boundary violations handling) are retained.

BBPSO aimed at offering a very simple, parameter-free PSO variant. Its convergence properties were studied in [10, 90, 115] and enhanced variants were proposed for applications in various scientific fields [66, 164–166].

Fully Informed PSO

The *Fully Informed PSO* (FIPS) was introduced in 2004 [80] as a variant of the Canonical PSO that extends the concept of neighbor influence. Specifically, in FIPS each particle is influenced by all its neighbors and not only the best one. The particles' update scheme of Eqs. (4) and (5) is modified in the following update rule,

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + \chi\left(x_{ij}^{(t)} - x_{ij}^{(t-1)}\right) + \sum_{k \in NB_i} \mathcal{C}_k\left(p_{kj}^{(t)} - x_{ij}^{(t)}\right), \tag{30}$$

where $i \in I$, $j \in D$, NB_i is the i-th particle's neighborhood of size s_i, and $\mathcal{C}_k \sim \mathcal{U}(0, c/s_i)$ is a random variable. Adapting the analysis of [21] in FIPS, the default parameter setting of Eq. (18) is used.

FIPS introduced a novel point of view for the concept of neighborhood. The social influence was enhanced, providing additional attractors to the particles. The outcome is their stochastic move around a stochastic average of the best positions of all neighbors. However, this also implies the dependency of the algorithm's performance on the selected neighborhood topology. For example, in the gbest PSO model where the whole swarm is the neighborhood of all particles, their move tends to be nearly random [59]. A multitude of neighborhood structures (more than 1000) were tested in [80] and interesting conclusions were derived regarding the potential of FIPS to improve the Canonical PSO.

Quantum PSO

Quantum PSO (QPSO) was introduced in 2004 [133], putting the algorithm in a new framework. In QPSO, the swarm is considered as a quantum system where each particle possesses a quantum state, while moving in a *Delta potential well* (DPW) toward a position **p**. The quantum state depends on the employed wave function.

Borrowing from previous PSO analyses such as [21], the aforementioned position $\mathbf{p}$ is defined as in Eq. (11). Depending on the considered potential, different variants of QPSO can be defined. Three established approaches are the following [81, 133, 135],

$$\text{Delta Potential Well:} \quad \mathbf{x}_i^{(t+1)} = \mathbf{p}^{(t)} \pm \frac{\ln\left(\frac{1}{\mathcal{R}}\right)}{2\,q\,\ln(\sqrt{2})} \left\| \mathbf{x}_i^{(t)} - \mathbf{p}^{(t)} \right\|, \tag{31}$$

$$\text{Harmonic Oscillator:} \quad \mathbf{x}_i^{(t+1)} = \mathbf{p}^{(t)} \pm \frac{\sqrt{\ln\left(\frac{1}{\mathcal{R}}\right)}}{0.47694\,q} \left\| \mathbf{x}_i^{(t)} - \mathbf{p}^{(t)} \right\|, \tag{32}$$

$$\text{Square Well:} \quad \mathbf{x}_i^{(t+1)} = \mathbf{p}^{(t)} + \frac{0.6574}{\xi\,q} \cos^{-1}\left(\pm\sqrt{\mathcal{R}}\right) \left\| \mathbf{x}_i^{(t)} - \mathbf{p}^{(t)} \right\|, \tag{33}$$

where $i \in I$, $\mathcal{R} \sim \mathcal{U}(0,1)$ is a random number, and ξ, q, are user-defined parameters. Despite the identified parameter sensitivity of the algorithm, it was embraced by the scientific community and extended in a number of interesting applications [17, 22, 39, 45, 50, 76, 163].

Unified PSO

The *Unified PSO* (UPSO) was introduced in 2004 [102] as a PSO variant that harnesses the gbest and lbest PSO models in a unified scheme. The motivation behind its development lies in the good intensification (exploitation) properties of the gbest model and the corresponding good diversification (exprolartion) properties of the lbest model. Their combination can form variants with different trade-offs of these two properties.

UPSO is based on the update equations of the Canonical PSO with constriction coefficient. Specifically, let,

$$\mathcal{L}_{ij}^{(t+1)} = \chi\, v_{ij}^{(t)} + \mathcal{C}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + \mathcal{C}_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right), \tag{34}$$

denote the velocity update of Eq. (4) for the lbest PSO model, where g_i is the index of the best neighbor of the i-th particle and $j \in D$. Also, let,

$$\mathcal{G}_{ij}^{(t+1)} = \chi\, v_{ij}^{(t)} + \mathcal{C}_3 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + \mathcal{C}_4 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right), \tag{35}$$

be the corresponding velocity update for the gbest PSO model, i.e., g denotes the overall best particle. Then, the basic UPSO scheme is defined as [102],

$$v_{ij}^{(t+1)} = u\,\mathcal{G}_{ij}^{(t+1)} + (1-u)\,\mathcal{L}_{ij}^{(t+1)}, \tag{36}$$

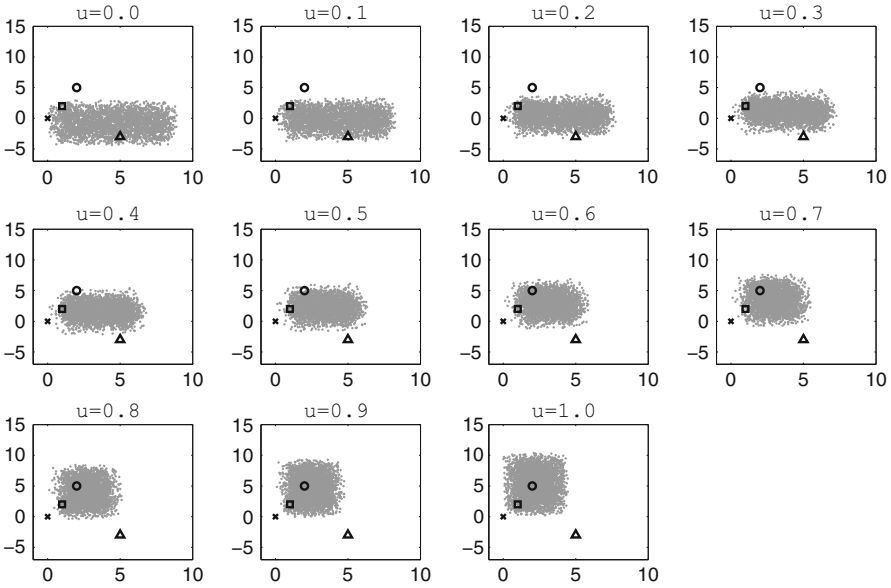


Fig. 8 The distribution of 3000 possible new positions (*light grey points*) of the 2-dimensional particle $\mathbf{x}_i = (0,0)^\top$ (*cross*) with own best position $\mathbf{p}_i = (1,2)^\top$ (*square*), neighborhood's best position $\mathbf{p}_{g_i} = (5,-3)^\top$ (*triangle*), and overall best $\mathbf{p}_g = (2,5)^\top$ (*circle*), for different values of $u \in [0,1]$. For simplicity, the velocity vector is set to $\mathbf{v}_i = (0,0)^\top$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{37}$$

where $i \in I$, $j \in D$, and $u \in [0,1]$ is a user-defined parameter called the *unification factor*, which controls the influence of the gbest and lbest term. Obviously, $u = 0$ corresponds to the lbest model, while $u = 1$ corresponds to the gbest model. All intermediate values define UPSO variants that combine the diversification/intensification properties of the two models. Figure 8 illustrates the distribution of new positions of a particle for different values of the unification factor.

In addition to the main UPSO model, an alternative with increased stochasticity was also proposed [102]. It came in two forms, namely,

$$v_{ij}^{(t+1)} = \mathcal{R}\, u\, \mathcal{G}_{ij}^{(t+1)} + (1-u)\, \mathcal{L}_{ij}^{(t+1)}, \tag{38}$$

which is mostly based on the lbest model, and,

$$v_{ij}^{(t+1)} = u\, \mathcal{G}_{ij}^{(t+1)} + \mathcal{R}\,(1-u)\, \mathcal{L}_{ij}^{(t+1)}, \tag{39}$$

which is based on the gbest model [102]. The stochastic parameter $\mathcal{R} \sim \mathcal{N}(\mu, \sigma^2)$ is normally distributed and imitates the mutation operators in Evolutionary Algorithms.

Theoretical properties of UPSO were studied in [102], while a thorough investigation of its parameter setting and adaptation was offered in [104]. Its performance has been studied on various problems [3, 38, 41, 65, 83, 84, 106, 144].

Cooperative PSO

Cooperative population-based algorithms [117] are based on the concept of cooperation between individuals toward a common goal. Cooperation can be either explicit through direct communication among them or implicit through a shared memory where information is deposited. Cooperation can be considered in a multi- or single-population framework. In the first case, each population usually operates on a subspace of the original search space, e.g., on one coordinate direction of the solution vector. Thus, its individuals carry partial solutions that are combined with those of the other populations, forming complete solutions. In the second case, the individuals usually carry complete solution information that is combined with the rest by using special recombination schemes [117].

In the context of PSO, one of the first notable attempts to design a *Cooperative PSO* (CPSO) took place in 2004 [146]. That version consists of a number d of swarms (equal to the problem's dimension), containing N_k particles each, $k \in D$, i.e., according to Eq. (3),

$$\mathcal{S}^{[1]} = \left\{A_1^{[1]}, \ldots, A_{N_1}^{[1]}\right\}, \quad \ldots, \quad \mathcal{S}^{[d]} = \left\{A_1^{[d]}, \ldots, A_{N_d}^{[d]}\right\}.$$

Each swarm probes only one coordinate direction of the solution vector, applying any PSO variant (Canonical PSO was used in the specific one). Yet, the evaluation of the particles with the objective function requires complete d-dimensional solutions. Thus, two main issues need to be addressed:

1. How shall particles from different swarms be selected to form complete solutions?
2. How shall particles be awarded or penalized for their contribution in solutions' quality?

The decisions on these crucial properties have direct impact on the algorithm's performance and, thus, require special attention.

In [146] two alternative schemes were proposed. The first scheme, denoted as CPSO-S$_k$, introduced a *context vector* $\mathbf{z}^*$ for the evaluation of the particles. This vector constitutes an external memory where each swarm $\mathcal{S}^{[k]}$ participates with its 1-dimensional overall best at the corresponding k-th direction component, i.e.,

$$\mathbf{z}^* = \left(\mathbf{p}_g^{[1]}, \mathbf{p}_g^{[2]}, \ldots, \mathbf{p}_g^{[d]}\right),$$

where $\mathbf{p}_g^{[k]}$ is the overall (1-dimensional) best of the k-th swarm [146]. Then, the evaluation of the i-th particle of the k-th swarm is done by replacing the k-th swarm's best in $\mathbf{z}^*$ with its own information, i.e.,

$$f\left(\mathbf{x}_i^{[k]}\right) = f\left(\mathbf{z}_{[i,k]}^*\right),$$

where,

$$\mathbf{z}_{[i,k]}^* = \left(\mathbf{p}_g^{[1]}, \ldots, \mathbf{p}_g^{[k-1]}, \mathbf{x}_i^{[k]}, \mathbf{p}_g^{[k+1]}, \ldots, \mathbf{p}_g^{[d]}\right),$$

where $\mathbf{x}_i^{[k]}$ is the current position of the i-th particle of the k-th swarm, which is under evaluation. Naturally, instead of the overall bests of the swarms, randomly selected best positions can be used in the context vector. Also, swarms of higher dimension can be used. However, both these alternatives can radically change the algorithm's performance. Obviously, the context vector $\mathbf{z}^*$ constitutes the best approximation of the problem's solution with CPSO-S_k.

The second variant presented in [146], denoted as CPSO-H_k, combines CPSO-S_k with the Canonical PSO and applies each algorithm alternatively in subsequent iterations. In addition, information exchange between the two algorithms was considered by sharing half of the discovered solutions between them. The experimental assessment revealed that both CPSO-S_k and CPSO-H_k are promising, opening the ground for further developments such as the ones in [48, 136, 152, 167].

Comprehensive Learning PSO

The *Comprehensive Learning PSO* (CLPSO) [72] was proposed in 2006 as an alternative for alleviating gbest PSO's premature convergence problem, which can be attributed to the use of the overall best position in the update equation of the velocities. In CLPSO, each particle can use the best position of any other particle to independently update its velocity, based on a probabilistic scheme.

Specifically, the velocity update of Eq. (4) is replaced with the following [72],

$$v_{ij}^{(t)} = \chi\, v_{ij}^{(t)} + \mathcal{C}\left(p_{q_{[i,j]}j} - x_{ij}^{(t)}\right), \tag{40}$$

where $j \in D$, $i \in I$, and $q_{[i,j]} \in I$ is the index of the particle that is used for the update of the j-th component of the i-th particle's velocity vector. Naturally, this particle can be either the i-th particle itself or another particle from the swarm. This decision is probabilistically made according to predefined probabilities $\rho_1, \rho_2, \ldots, \rho_d$, i.e.,

$$q_{[i,j]} = \begin{cases} i, & \text{if } \mathcal{R} \leqslant \rho_j, \\ \text{TOURN}\,(I'), & \text{otherwise,} \end{cases} \qquad \text{for all } \; j \in D,$$

where $\mathcal{R} \sim \mathcal{U}(0,1)$ is a uniformly distributed random variable, $I' = I \setminus \{i\}$, and TOURN (I') is an index selected from I' through *tournament selection* [72]. The latter procedure includes the random selection of two particles from the set I'. The best between them, i.e., the one with smallest objective value, is the winner and participates in the update of v_{ij}.

In case of $q_{[i,j]} = i$, for all $j \in D$, one of the components of v_{ij} is randomly selected and determined anew by using another particle. Also, the indices $q_{[i,j]}$ are updated for each particle after a number of non-improving iterations. CLPSO has been extensively studied in [72], while a number of improvements, modifications, and applications in various fields have been proposed in relevant literature [40, 43, 154, 161].

TRIBES

The *TRIBES* algorithm was proposed in 2006 [19] as a novel PSO variant with self-adaptation capability. It is based on a special communication scheme between neighborhoods and admits the update rules of any PSO variant. In TRIBES, the i-th particle is called *informant* of the j-th particle if it shares its best position for the update of the latter. Accordingly, a *tribe* can be defined as a subset of the swarm, where each one of its members is informant of all the rest in the same tribe. Obviously, the swarm is the union set of all tribes.

Each tribe must have at least one communication channel with another tribe. In other words, between two particles there shall be at least one path in the neighborhood's graph that connects them [19]. Also, the algorithm is self-adaptive, i.e., existing tribes can be deleted and new tribes can be generated. Hence, the communication channels between tribes also change dynamically. The goodness criterion for the tribes is related to the performance of their members-particles, which are characterized as *neutral* if they have not improved their best position in the last iteration, *good* if improvement was achieved in the last iteration, and *excellent* if improvement was achieved for more than one consecutive iterations. Accordingly, a tribe TR that contains s_{TR} particles is characterized as follows,

$$TR \;\text{ is }\; \begin{cases} \text{good}, & \text{if } \; N_{TR}^{\text{good}} > \mathcal{R}, \\ \text{bad}, & \text{otherwise,} \end{cases}$$

where N_{TR}^{good} is the number of good particles in tribe TR, and $\mathcal{R}$ is a randomly selected integer in $\{0, 1, \ldots, s_{TR}\}$. Moreover, additional rules are applied for the generation/deletion of particles and tribes as follows:

1. The worst particle of the best tribe can be eliminated, inheriting its communication channels to the best particle of its tribe.
2. If a tribe consists of only one particle, it is eliminated if it has an informant with better performance.
3. Each tribe that was characterized as "bad" generates two new particles. The first one is randomly generated within the whole search space. The second one is uniformly generated in the sphere with center the best position of the best informant of the tribe's best particle, and radius equal to the distance between the latter and the sphere's center.

Adaptations were recommended to occur after a number of iterations so that the algorithm can deploy its dynamic [19]. Promising results were received with TRIBES under various settings [19]. Although TRIBES is a rather controversial PSO variant, it has contributed toward the development of self-adaptation mechanisms [25–27] and has been applied on interesting problems [30].

Niching PSO

Niching algorithms are applied on multimodal optimization problems where the main goal is the identification of multiple global/local minima. In such problems, the algorithms must be capable of identifying minima and retaining them until the end of the search. Although transformation techniques such as the ones presented in section "Performance-Enhancing Techniques" can be used in these cases, alternative algorithmic models that do not use external procedures have been developed.

Two efficient *Niching PSO* approaches are the *Speciation-based PSO* (SPSO) [94] and the *Fitness Euclidean-distance Ratio PSO* (FERPSO) [70]. In SPSO, the swarm is divided into subswarms, which are considered as species represented by the dominant (best) particle in the subswarm. A niche radius is also specified to define the size of species. Special procedures are applied for determining species and their seeds, while the global best particle is replaced by the species best or species seed. Also, all particles in the same species use the same neighborhood best at each iteration.

On the other hand, FERPSO is based on the lbest PSO model of Eqs. (4) and (5) [70], where the neighborhood's best $\mathbf{p}_{g_i}$ is taken as the particle that maximizes the *fitness Euclidean-distance ratio* (FER), defined as,

$$\mathrm{FER}_{i,k} = \frac{\alpha \left(f\left(\mathbf{p}_i\right) - f\left(\mathbf{p}_k\right)\right)}{\left\|\mathbf{p}_i - \mathbf{p}_k\right\|}, \qquad k \in I,$$

where the scaling factor α is defined as,

$$\alpha = \frac{\sqrt{\sum_{l=1}^{d} \left(x_l^{\max} - x_l^{\min}\right)^2}}{f\left(\mathbf{p}_g\right) - f\left(\mathbf{p}_w\right)},$$

with $\mathbf{p}_g$ and $\mathbf{p}_w$ being the swarm's best and worst particles, respectively [70]. The effectiveness of both SPSO and FERPSO has led to further enhancements such as the ones in [71, 118, 125].

Standard PSO

The *Standard PSO* (SPSO) was introduced in an attempt to define a baseline for the development and assessment of new PSO variants. Although there have been various versions (2006, 2007, and 2011), only the latest one, SPSO-2011 [20], is considered here since it cures a number of deficiencies identified in the previous versions.

A characteristic feature of SPSO-2011 is the independence on the coordinate system. Let,

$$\mathcal{P}_{ij}^{(t)} = x_{ij}^{(t)} + \mathcal{C}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)}\right), \tag{41}$$

$$\mathcal{L}_{ij}^{(t)} = x_{ij}^{(t)} + \mathcal{C}_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)}\right), \tag{42}$$

with $j \in D$, define two points for the i-th particle that are a little "beyond" its own best position and its neighborhood's best position, respectively, at iteration t. Then, the center of gravity between the two points and the particle's current position is defined as,

$$\mathcal{G}_i^{(t)} = \frac{1}{3}\left(\mathbf{x}_i^{(t)} + \mathcal{P}_i^{(t)} + \mathcal{L}_i^{(t)}\right). \tag{43}$$

A new point $\mathbf{x'}_i^{(t)}$ is randomly (not necessarily uniformly) generated in the hypersphere $\mathcal{H}_i^{(t)}$ with center $\mathcal{G}_i^{(t)}$ and radius equal to its distance from the actual $\mathbf{x}_i^{(t)}$, i.e.,

$$\mathbf{x'}_i^{(t)} \in \mathcal{H}_i^{(t)}\left(\mathcal{G}_i^{(t)}, \left\|\mathcal{G}_i^{(t)} - \mathbf{x}_i^{(t)}\right\|\right). \tag{44}$$

Then, the update equations of SPSO-2011 are given as follows [20],

$$v_{ij}^{(t+1)} = \chi\, v_{ij}^{(t)} + x'^{(t)}_{ij} - x_i^{(t)}, \tag{45}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{46}$$

where $i \in I$ and $j \in D$. The rest of the algorithm follows the Canonical PSO approach. The particles are bounded in the search space according to Eq. (19). The algorithm's parameters were studied in [162], while a thorough theoretical analysis was provided in [13].

Memetic PSO

The term *Memetic Algorithm* [85] is used to describe hybrid algorithms that consist of population-based metaheuristics with additional local search or learning mechanisms. Early *Memetic PSO* (MPSO) schemes appeared in 2005 [74], hybridizing PSO with the Solis and Wets local search approach. Later, different schemes were proposed using alternative local search algorithms, such as the Random Walk with Direction Exploitation and the Hooke and Jeeves method [108, 109]. Recently, PSO was integrated with gradient-based optimization as well as direct search approaches in MEMPSODE, an efficient general-purpose software package [149]. Further enhancements and applications can be found in [5, 44, 150].

Besides the choice of an appropriate local search algorithm, which is mostly a problem-dependent decision, additional resolutions shall be made prior to the application of MPSO [106]:

1. When to apply local search?
2. Where to apply local search?
3. What computational budget shall be devoted to local search?

A straightforward choice is the application of local search on the best positions (if updated) and/or the current positions of the particles at each iteration, until a local minimum is found. Naturally, this approach would require excessive computational resources that are hardly available in practice.

For this reason, the following schemes were proposed in [108] after empirical evaluations on established test problems:

(S1) Application of local search on the overall best position $\mathbf{p}_g$, whenever it changes.
(S2) For each best position $\mathbf{p}_i$, $i \in I$, local search is applied with a predefined probability ρ.
(S3) Local search is applied both on $\mathbf{p}_g$ and some randomly selected best positions $\mathbf{p}_i$, $i \in I$.
(S4) Local search is applied on $\mathbf{p}_g$ as well as on the best positions $\mathbf{p}_i$ that lie in adequate distance from $\mathbf{p}_g$, e.g., $\|\mathbf{p}_g - \mathbf{p}_i\| > \varepsilon$, where $\varepsilon > 0$ is a predefined distance usually defined as a fraction of the search space diameter.

In addition, the *Shannon information entropy*, used as a measure of the swarm's information diversity, was employed in [107] along with the above schemes in order to make swarm-level decisions on the application of local search. Further details

and extensive results are given in [106], where it is shown that MPSO outperforms the Canonical PSO (as expected) but also its gbest model can outperform the lbest model of Canonical PSO. The latter result suggests that local search applied as above can be beneficial both for PSO's intensification and diversification properties.

Opposition-Based PSO

The opposition-based algorithms are grounded on the concept of *opposite point* [120]. If $\mathbf{x} = (x_1, \ldots, x_d)^\top$ is a point in the search space $\mathcal{X}$ defined as in Eq. (2), then its opposite is defined as a point $\mathbf{x}' = (x'_1, \ldots, x'_d)^\top$ with $x'_j = x_j^{\min} + x_j^{\max} - x_j$, for all $j \in D$. Evaluating both points simultaneously and keeping the best one can accelerate the optimization procedure according to the study in [120].

This scheme was recently adopted in the framework of PSO, producing the *Generalized Opposition-based PSO* (GOPSO) [151]. In GOPSO, there are two swarms, $\mathcal{S}$ and $\mathcal{S}'$, comprising the particles and their opposites, respectively. The initial positions $\mathbf{x}_i^{(0)}$, $i \in I$, of $\mathcal{S}$ are randomly initialized, while for $\mathcal{S}'$ the initial positions $\mathbf{x'}_i^{(0)}$ are obtained as follows [151],

$$x'^{(0)}_{ij} = \mathcal{R}\left(\alpha_j^{(0)} + \beta_j^{(0)}\right) - x_{ij}, \tag{47}$$

where $i \in I$, $j \in D$, $\mathcal{R} \sim \mathcal{U}(0,1)$, and $\alpha_j^{(0)} = x_j^{\min}$, $\beta_j^{(0)} = x_j^{\max}$. Subsequently, both swarms are evaluated and merged. The N best particles are then selected to form the initial swarm.

At each iteration, a probabilistic decision is taken. The algorithm, with a user-defined probability ρ, either chooses to update the boundaries $\alpha_j^{(t)}$, $\beta_j^{(t)}$, as follows [151],

$$\alpha_j^{(t)} = \min_i \left\{x_{ij}^{(t)}\right\}, \qquad \beta_j^{(t)} = \max_i \left\{x_{ij}^{(t)}\right\}, \tag{48}$$

or applies the update equations of the Canonical PSO defined in Eqs. (4) and (5). The procedure continues with the best positions update of Eq. (7). The new overall best undergoes also mutation, where its components are perturbed with random numbers following a Cauchy distribution [151].

Experimental results have shown that GOPSO can be competitive to other PSO variants [151]. A number of different opposition-based approaches have been proposed in various application fields [28, 55, 77, 168].

PSO in Noisy Environments

Noisy problems arise very often in engineering applications. The use of measurement instruments or approximations based on inaccurate mathematical models impose uncertainty on the objective function values. Thus, *noise-tolerance* is a desirable property for metaheuristic optimization algorithms such as PSO. In [97] the gbest PSO model was studied on a number of test problems contaminated by Gaussian noise, exhibiting promising behavior. Various other studies followed in subsequent years [8, 91, 119].

A common technique for tackling noisy problems is the re-evaluation of the objective function at each point. Specifically, if the objective function is given as,

$$f'(\mathbf{x}) = f(\mathbf{x}) + \mathcal{R},$$

where $\mathcal{R}$ is a random variable following a (usually Gaussian) distribution, then PSO evaluates the particles by using,

$$F(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} f'^{(m)}(\mathbf{x}),$$

where $f'^{(m)}(\mathbf{x})$ is the m-th re-evaluation of $\mathbf{x}$ using $f'(\mathbf{x})$. Re-evaluation serves as a mean for approximating the expected value of the noisy objective function, i.e., $F(\mathbf{x}) \approx \mathbb{E}(f'(\mathbf{x}))$. Accuracy increases with the number M of re-evaluations, although it also increases the computational cost.

Thus, the trade-off between better estimations of the objective values and the corresponding computational burden shall be tuned. In such cases, specialized techniques such as the *Optimal Computing Budget Allocation* (OCBA) [16] have been used to optimally allocate the re-evaluations budget in order to provide reliable evaluation and identification of the promising particles [91]. These techniques can be used along with proper parameter tuning [8] or learning strategies [110] for improved results. Also, they do not require the modification of the algorithm. Alternatively, specialized operators have been proposed with remarkable success [47].

Multiobjective PSO

Multiobjective optimization (MO) problems consist of a number of objective functions that need to be simultaneously optimized. In contrast to the definition of single-objective problems in Eq. (1), an MO problem is defined as the minimization of a vector function [24],

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x}))^\top,$$

possibly subject to constraints $C_i(\mathbf{x}) \leqslant 0$, $i = 1, 2, \ldots, m$. Typically, the objective functions $f_k(\mathbf{x})$ can be conflicting. Thus, it is highly improbable that a single solution that globally minimizes all of them can be found.

For this reason, the main interest in such problems is concentrated on the detection of *Pareto optimal* solutions. These solutions are nondominated by any other point in the search space, i.e., they are at least as good as any other point for all the objectives $f_k(\mathbf{x})$. Formally, if $\mathbf{x}$, $\mathbf{y}$, are two points in the search space $\mathcal{X}$, then $\mathbf{f}(\mathbf{x})$ is said to dominate $\mathbf{f}(\mathbf{y})$, and we denote $\mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{y})$, if it holds that,

$$f_k(\mathbf{x}) \leqslant f_k(\mathbf{y}), \text{ for all } k = 1, 2, \ldots, K,$$

and,

$$f_{k'}(\mathbf{x}) < f_{k'}(\mathbf{y}), \text{ for at least one } k' \in \{1, 2, \ldots, K\}.$$

Thus, $\mathbf{x}^* \in \mathcal{X}$ is a Pareto optimal point if there is no other point $\mathbf{y} \in \mathcal{X}$ such that $\mathbf{f}(\mathbf{y}) \prec \mathbf{f}(\mathbf{x}^*)$. Obviously, an (even infinite) set $\{\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots\}$ of Pareto optimal solutions may exist. The set $\{\mathbf{f}(\mathbf{x}_1^*), \mathbf{f}(\mathbf{x}_2^*), \ldots\}$ is called the *Pareto front*.

There are two main approaches for tackling MO problems. The first one aggregates the objectives into a single one and solves the problem with the typical methodologies for single-objective optimization. The second approach requires vector evaluated operators and it is based on the concept of Pareto dominance. In the context of PSO, early aggregation approaches appeared in 2002 [99], where the Canonical PSO was used for the minimization of a weighted aggregation of the objective functions,

$$F(\mathbf{x}) = \sum_{k=1}^{K} w_k \, f_k(\mathbf{x}), \qquad \sum_{k=1}^{K} w_k = 1.$$

Both a *conventional weighted aggregation* (CWA) approach with fixed weights as well as a *dynamic weighted aggregation* (DWA) approach [53] were investigated with promising results. Obviously, the detection of many Pareto optimal solutions through weighted aggregation requires multiple applications of PSO, since each run provides a single solution of $F(\mathbf{x})$. From the computational point of view, this is a drawback since the swarms can simultaneously evolve many solutions. Yet, it is still a popular approach in applications mostly due to its simplicity.

A *Vector Evaluated PSO* (VEPSO) was also proposed in [99] and parallelized later in [96]. VEPSO uses a number of K swarms, one for each objective f_k.

The k-th swarm $\mathcal{S}_k$ is evaluated only with the corresponding objective f_k, $k = 1, 2, \ldots, K$. The swarms are updated according to the gbest model of the Canonical PSO, although with a slight modification. Specifically, the overall best that is used for the velocity update of the particles in the k-th swarm comes from another swarm. Clearly, this is a migration scheme aiming at transferring information among swarms. The donator swarm can be either a neighbor of the k-th swarm in a ring topology scheme as the one described in section "Concept of Neighborhood" or it can be randomly selected [99]. VEPSO was studied on standard MO benchmark problems with promising results [96].

There is a large number of new developments and applications on multiobjective PSO approaches in literature [1, 2, 27, 29, 32, 51, 75, 156, 160, 169]. The interested reader can find comprehensive surveys in [105, 121].

Applications

It would be futile to even try to enumerate all applications of PSO that have been published so far. From 2005 and on, more than 400 papers with PSO's applications appear every year, spanning various scientific and technological fields. Electrical Engineering concentrates the majority of these works, especially in the fields of power systems, control, antenna design, electromagnetics, sensors, networks and communications. Artificial Intelligence also hosts a large number of PSO-based applications, especially in robotics, machine learning, and data mining. Bioinformatics and Operations Research follow closely, with numerous works in modeling, health-care systems, scheduling, routing, supply chain management, and forecasting.

A number of applications is cited in the previous sections. In addition, the interested reader can refer to devoted survey papers such as [112], which was probably the first notable attempt to collect and categorize PSO's applications. An analytical survey was published in [127], where a huge number of PSO-based applications was categorized along with more than 100 PSO variants. Further applications are reported in relevant books such as [106]. The Appendix at the end of the present work contains a number of sources for further inquiry on PSO-based developments.

Conclusions

PSO has been established as one of the most popular metaheuristic optimization algorithms. Its popularity emanates from its nice performance and adequate simplicity that renders it usable even by non-expert researchers. In the previous sections, a number of variants and improvements were presented. This is only a small fraction of the existing PSO-related literature, which counts thousands of papers. Thus, a reasonable question can be put regarding the room left for further developments on PSO. In the author's opinion, the answer is: a lot.

Despite the numerous research contributions, there are still many issues that need improvement to achieve the main goal of intelligent behavior and self-adaptation. Moreover, the evolution of computer and web technologies always introduces new challenges on the design and development of algorithms that can take full advantage of their properties to solve problems of increasing complexity. For example, the GPU computing paradigm and modern multicore desktop systems can offer computational power comparable to small- and medium-size clusters. Cloud computing and ubuquitous computing environments are other examples. Also, new ad hoc operators and procedures for specific problems are expected to boost PSO's performance.

Closing this chapter, the author would like to quote Albert Einstein's words to motivate new researchers toward unconventional thinking, which was the main ingredient for the development of PSO so far:

> You will never solve problems using the same thinking you created them with.

Cross-References

- ▸ Adaptive and Multilevel Metaheuristics
- ▸ Ant Colony Optimization: A Component-Wise Overview
- ▸ Hyper-heuristics
- ▸ Memetic Algorithms
- ▸ Multi-objective Optimization
- ▸ Multi-start Methods
- ▸ Parallel Metaheuristic Search

Appendix

A number of sources for further inquiry and experimentation with PSO is reported below.

Books

[19, 34, 62–64, 86, 93, 106, 134]

Survey papers

[4, 6, 7, 35, 112, 114, 122, 123, 127, 141]

Webpages

Particle Swarm Central

http://www.particleswarm.info/

M. Clerc's PSO page

http://clerc.maurice.free.fr/pso/

Software

PSO in C (code published in [149])

http://www.cpc.cs.qub.ac.uk/summaries/AELM_v1_0.html

PSO in Matlab

http://www.mathworks.com/matlabcentral/fileexchange/7506
http://psotoolbox.sourceforge.net/
PSO in Java
http://jswarm-pso.sourceforge.net/
http://gundog.lbl.gov/GO/jdoc/genopt/algorithm/PSOCC.html

References

1. Abido MA (2010) Multiobjective particle swarm optimization with nondominated local and global sets. Nat Comput 9(3):747–766
2. Agrawal S, Panigrahi BK, Tiwari MK (2008) Multiobjective particle swarm algorithm with fuzzy clustering for electrical power dispatch. IEEE Trans Evol Comput 12(5):529–541
3. Ahmadi MA (2012) Neural network based unified particle swarm optimization for prediction of asphaltene precipitation. Fluid Phase Equilib 314:46–51
4. Aote AS, Raghuwanshi MM, Malik L (2013) A brief review on particle swarm optimization: limitations & future directions. Int J Comput Sci Eng 2(5):196–200
5. Aziz M, Tayarani-N M-H (2014) An adaptive memetic particle swarm optimization algorithm for finding large-scale latin hypercube designs. Eng Appl Artif Intell 36:222–237
6. Banks A, Vincent J, Anyakoha C (2007) A review of particle swarm optimization. Part i: background and development. Nat Comput 6(4):467–484
7. Banks A, Vincent J, Anyakoha C (2008) A review of particle swarm optimization. Part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Nat Comput 7(1):109–124
8. T. Bartz-Beielstein, Blum D, Branke J (2007) Particle swarm optimization and sequential sampling in noisy environments. In: Doerner KF et al (ed) Metaheuristics: progress in complex systems optimization. Operations research/computer science interfaces series, vol 39. Springer, New York, pp 261–273
9. Bin W, Qinke P, Jing Z, Xiao C (2012) A binary particle swarm optimization algorithm inspired by multi-level organizational learning behavior. Eur J Oper Res 219(2):224–233
10. Blackwell T (2012) A study of collapse in bare bones particle swarm optimization. IEEE Trans Evol Comput 16(3):354–372
11. Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput J 11(6):4135–4151
12. Bonabeau E, Dorigo M, Théraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, New York
13. Bonyadi MR, Michalewicz Z (2014) SPSO2011 – analysis of stability, local convergence, and rotation sensitivity. In: GECCO 2014 – proceedings of the 2014 genetic and evolutionary computation conference, Vancouver, pp 9–15
14. Camci F (2009) Comparison of genetic and binary particle swarm optimization algorithms on system maintenance scheduling using prognostics information. Eng Optim 41(2):119–136
15. Chauhan P, Deep K, Pant M (2013) Novel inertia weight strategies for particle swarm optimization. Memet Comput 5(3):229–251
16. Chen C-H, Lin J, Yücesan E, Chick SE (2000) Simulation budget allocation for further enhancing the efficiency of ordinal optimization. Discr Event Dyn Syst Theory Appl 10(3):251–270
17. Chen J, Yang D, Feng Z (2012) A novel quantum particle swarm optimizer with dynamic adaptation. J Comput Inf Syst 8(12):5203–5210
18. Chen Z, He Z, Zhang C (2010) Particle swarm optimizer with self-adjusting neighborhoods. In: Proceedings of the 12th annual genetic and evolutionary computation conference (GECCO 2010), Portland, pp 909–916
19. Clerc M (2006) Particle swarm optimization. ISTELtd, London

20. Clerc M (2012) Standard particle swarm optimization. Technical report 2012, Particle Swarm Central
21. Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6(1):58–73
22. Coelho LdS (2008) A quantum particle swarm optimizer with chaotic mutation operator. Chaos Solitons Fractals 37(5):1409–1418
23. Coello Coello CA (1999) Self-adaptive penalties for GA-based optimization. In: Proceedings of the 1999 IEEE congress on evolutionary computation, Washington, vol 1, pp 573–580
24. Coello Coello CA, Van Veldhuizen DA, Lamont GB (2002) Evolutionary algorithms for solving multi-objective problems. Kluwer, New York
25. Cooren Y, Clerc M, Siarry P (2008) Initialization and displacement of the particles in TRIBES, a parameter-free particle swarm optimization algorithm. Stud Comput Intell 136:199–219
26. Cooren Y, Clerc M, Siarry P (2009) Performance evaluation of TRIBES, an adaptive particle swarm optimization algorithm. Swarm Intell 3(2):149–178
27. Cooren Y, Clerc M, Siarry P (2011) MO-TRIBES, an adaptive multiobjective particle swarm optimization algorithm. Comput Optim Appl 49(2):379–400
28. Dai Y, Liu L, Feng S (2014) On the identification of coupled pitch and heave motions using opposition-based particle swarm optimization. Math Probl Eng 2014(3):1–10
29. Daneshyari M, Yen GG (2011) Cultural-based multiobjective particle swarm optimization. IEEE Trans Syst Man Cybern Part B Cybern 41(2):553–567
30. Daoudi M, Boukra A, Ahmed-Nacer M (2011) Adapting TRIBES algorithm for traveling salesman problem. In: Proceedings of the 10th international symposium on programming and systems (ISPS' 2011), pp 163–168
31. Davarynejad M, Van Den Berg J, Rezaei J (2014) Evaluating center-seeking and initialization bias: the case of particle swarm and gravitational search algorithms. Inf Sci 278:802–821
32. Dos Santos Coelho L, Ayala HVH, Alotto P (2010) A multiobjective gaussian particle swarm approach applied to electromagnetic optimization. IEEE Trans Mag 46(8):3289–3292
33. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings sixth symposium on micro machine and human science, Piscataway, pp 39–43. IEEE Service Center
34. Engelbrecht AP (2006) Fundamentals of computational swarm intelligence. Wiley, Chichester
35. Eslami M, Shareef H, Khajehzadeh M, Mohamed A (2012) A survey of the state of the art in particle swarm optimization. R J Appl Sci Eng Technol 4(9):1181–1197
36. Gao W-F, Liu S-Y, Huang L-L (2012) Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique. Commun Nonlinear Sci Numer Simul 17(11):4316–4327
37. Ge RP, Qin YF (1987) A class of filled functions for finding global minimizers of a function of several variables. J Optim Theory Appl 54:241–252
38. Gholipour R, Khosravi A, Mojallali H (2013) Suppression of chaotic behavior in duffing-holmes system using backstepping controller optimized by unified particle swarm optimization algorithm. Int J Eng Trans B Appl 26(11):1299–1306
39. Gholizadeh S, Moghadas R (2014) Performance-based optimum design of steel frames by an improved quantum particle swarm optimization. Adv Struct Eng 17(2):143–156
40. Goudos SK, Moysiadou V, Samaras T, Siakavara K, Sahalos JN (2010) Application of a comprehensive learning particle swarm optimizer to unequally spaced linear array synthesis with sidelobe level suppression and null control. IEEE Antennas Wirel Propag Lett 9:125–129
41. He G, Wu B (2014) Unified particle swarm optimization with random ternary variables and its application to antenna array synthesis. J Electromag Waves Appl 28(6): 752–764
42. He J, Dai H, Song X (2014) The combination stretching function technique with simulated annealing algorithm for global optimization. Optim Methods Softw 29(3): 629–645

43. Hu Z, Bao Y, Xiong T (2014) Comprehensive learning particle swarm optimization based memetic algorithm for model selection in short-term load forecasting using support vector regression. Appl Soft Comput J 25:15–25
44. Huang K-W, Chen J-L, Yang C-S, Tsai C-W (2015) A memetic particle swarm optimization algorithm for solving the dna fragment assembly problem. Neural Comput Appl 26(3): 495–506
45. Jamalipour M, Gharib M, Sayareh R, Khoshahval F (2013) PWR power distribution flattening using quantum particle swarm intelligence. Ann Nucl Energy 56:143–150
46. Janson S, Middendorf M (2004) A hierarchical particle swarm optimizer for dynamic optimization problems. Lecture notes in computer science, vol 3005. Springer, Berlin/New York, pp 513–524
47. Janson S, Middendorf M (2006) A hierarchical particle swarm optimizer for noisy and dynamic environments. Genet Program Evol Mach 7(4):329–354
48. Jiang B, Wang N (2014) Cooperative bare-bone particle swarm optimization for data clustering. Soft Comput 18(6):1079–1091
49. Jiang M, Luo YP, Yang SY (2007) Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. Inf Process Lett 102:8–16
50. Jiao B, Yan S (2011) A cooperative co-evolutionary quantum particle swarm optimizer based on simulated annealing for job shop scheduling problem. Int J Artif Intell 7(11 A): 232–247
51. Jin N, Rahmat-Samii Y (2007) Advances in particle swarm optimization for antenna designs: real-number, binary, single-objective and multiobjective implementations. IEEE Trans Antennas Propag 55(3 I):556–567
52. Jin N, Rahmat-Samii Y (2010) Hybrid real-binary particle swarm optimization (HPSO) in engineering electromagnetics. IEEE Trans Antennas Propag 58(12):3786–3794
53. Jin Y, Olhofer M, Sendhoff B (2001) Evolutionary dynamic weighted aggregation for multiobjective optimization: why does it work and how? In: Proceedings GECCO 2001 conference, San Francisco, pp 1042–1049
54. Kadirkamanathan V, Selvarajah K, Fleming PJ (2006) Stability analysis of the particle dynamics in particle swarm optimizer. IEEE Trans Evol Comput 10(3):245–255
55. Kaucic M (2013) A multi-start opposition-based particle swarm optimization algorithm with adaptive velocity for bound constrained global optimization. J Glob Optim 55(1):165–188
56. Kennedy J (1998) The behavior of particles. In: Porto VW, Saravanan N, Waagen D, Eiben AE (eds) Evolutionary programming, vol VII. Springer, Berlin/New York, pp 581–590
57. Kennedy J (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the IEEE congress on evolutionary computation, Washington, DC. IEEE Press, pp 1931–1938
58. Kennedy J (2003) Bare bones particle swarms. In: Proceedings of the IEEE swarm intelligence symposium, Indianapolis. IEEE Press, pp 80–87
59. Kennedy J (2010) Particle swarm optimization. In: Sammut C, Webb G (eds) Encyclopedia of machine learning. Springer, Boston, pp 760–766
60. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceeding of the IEEE international conference neural networks, Piscataway, vol IV. IEEE Service Center, pp 1942–1948
61. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the conference on systems, man and cybernetics, Hyatt Orlando, pp 4104–4109
62. Kennedy J, Eberhart RC (2001) Swarm intelligence. Morgan Kaufmann Publishers, San Francisco
63. Kiranyaz S, Ince T, Gabbouj M (2014) Multidimensional particle swarm optimization for machine learning and pattern recognition. Springer, Berlin
64. Kishk A (2008) Particle swarm optimizaton: a physics-based approach. Morgan and Claypool Publishers, Arizona
65. Kotsireas IS, Koukouvinos C, Parsopoulos KE, Vrahatis MN (2006) Unified particle swarm optimization for Hadamard matrices of Williamson type. In: Proceedings of the 1st interna-

tional conference on mathematical aspects of computer and information sciences (MACIS 2006), Beijing, pp 113–121
66. Krohling RA, Campos M, Borges P (2010) Bare bones particle swarm applied to parameter estimation of mixed weibull distribution. Adv Intell Soft Comput 75:53–60
67. Kwok NM, Ha QP, Liu DK, Fang G, Tan KC (2007) Efficient particle swarm optimization: a termination condition based on the decision-making approach. In: Proceedings of the 2007 IEEE congress on evolutionary computation (CEC 2007), Singapore, pp 3353–3360
68. Laskari EC, Parsopoulos KE, Vrahatis MN (2002) Particle swarm optimization for integer programming. In: Proceedings of the IEEE 2002 congress on evolutionary computation (IEEE CEC 2002), Honolulu. IEEE Press, pp 1582–1587
69. Lawler EL, Wood DW (1966) Branch and bound methods: a survey. Oper Res 14:699–719
70. Li X (2007) A multimodal particle swarm optimizer based on fitness Euclidean-distance ratio. ACM, New York, pp 78–85
71. Li X (2010) Niching without Niching parameters: Particle swarm optimization using a ring topology. IEEE Trans Evol Comput 14(1):150–169
72. Liang JJ, Qin AK, Suganthan PN, Baskar S (2006) Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evol Comput 10(3):281–295
73. Likas A, Blekas K, Stafylopatis A (1996) Parallel recombinative reinforcement learning: a genetic approach. J Intell Syst 6(2):145–169
74. Liu B-F, Chen H-M, Chen J-H, Hwang S-F, Ho S-Y (2005) MeSwarm: memetic particle swarm optimization. ACM, New York, pp 267–268
75. Liu DS, Tan KC, Huang SY, Goh CK, Ho WK (2008) On solving multiobjective bin packing problems using evolutionary particle swarm optimization. Eur J Oper Res 190(2): 357–382
76. Liu R, Zhang P, Jiao L (2014) Quantum particle swarm optimization classification algorithm and its applications. Int J Pattern Recognit Artif Intell 28(2)
77. Lv L, Wang H, Li X, Xiao X, Zhang L (2014) Multi-swarm particle swarm optimization using opposition-based learning and application in coverage optimization of wireless sensor network. Sensor Lett 12(2):386–391
78. Magoulas GD, Vrahatis MN, Androulakis GS (1997) On the alleviation of local minima in backpropagation. Nonlinear Anal Theory Methods Appl 30(7):4545–4550
79. Manquinho VM, Marques Silva JP, Oliveira AL, Sakallah KA (1997) Branch and bound algorithms for highly constrained integer programs. Technical report, Cadence European Laboratories, Portugal
80. Mendes R, Kennedy J, Neves J (2004) The fully informed particle swarm: simpler, maybe better. IEEE Trans Evol Comput 8(3):204–210
81. Mikki SM, Kishk AA (2006) Quantum particle swarm optimization for electromagnetics. IEEE Trans Antennas Propag 54(10):2764–2775
82. Moustaki E, Parsopoulos KE, Konstantaras I, Skouri K, Ganas I (2013) A first study of particle swarm optimization on the dynamic lot sizing problem with product returns. In: XI Balkan conference on operational research (BALCOR 2013), Belgrade, pp 348–356
83. Nanda B, Maity D, Maiti DK (2014) Crack assessment in frame structures using modal data and unified particle swarm optimization technique. Adv Struct Eng 17(5):747–766
84. Nanda B, Maity D, Maiti DK (2014) Modal parameter based inverse approach for structural joint damage assessment using unified particle swarm optimization. Appl Math Comput 242:407–422
85. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol Comput 2:1–14
86. Olsson AE (ed) (2011) Particle swarm optimization: theory, techniques and applications. Nova Science Pub Inc., New York
87. Ozcan E, Mohan CK Analysis of a simple particle swarm optimization. In: Intelligent engineering systems through artificial neural networks, vol 8. ASME Press, New York, pp 253–258

88. Ozcan E, Mohan CK (1999) Particle swarm optimization: surfing the waves. In: Proceedings of the 1999 IEEE international conference on evolutionary computation, Washington, DC, pp 1939–1944
89. Padhye N, Deb K, Mittal P (2013) Boundary handling approaches in particle swarm optimization. In: Proceedings of seventh international conference on bio-inspired computing: theories and applications (BIC-TA 2012), Gwalior, vol 201, pp 287–298
90. Pan F, Hu X, Eberhart R, Chen Y (2008) An analysis of bare bones particle swarm. In: Proceedings of the 2008 IEEE swarm intelligence symposium, St. Louis
91. Pan H, Wang L, Liu B (2006) Particle swarm optimization for function optimization in noisy environment. Appl Math Comput 181(2):908–919
92. Pandremmenou K, Kondi LP, Parsopoulos KE, Bentley ES (2014) Game-theoretic solutions through intelligent optimization for efficient resource management in wireless visual sensor networks. Signal Process Image Commun 29(4):472–493
93. Parasuraman D (2012) Handbook of particle swarm optimization: concepts, principles & applications. Auris reference, Nottingham
94. Parrott D, Li X (2006) Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Trans Evol Comput 10(4):440–458
95. Parsopoulos KE, Plagianakos VP, Magoulas GD, Vrahatis MN (2001) Objective function "stretching" to alleviate convergence to local minima. Nonlinear Anal Theory Methods Appl 47(5):3419–3424
96. Parsopoulos KE, Tasoulis DK, Vrahatis MN (2004) Multiobjective optimization using parallel vector evaluated particle swarm optimization. In: Hamza MH (ed) Proceedings of the IASTED 2004 international conference on artificial intelligence and applications (AIA 2004), Innsbruck, vol 2. IASTED/ACTA Press, pp 823–828
97. Parsopoulos KE, Vrahatis MN (2001) Particle swarm optimizer in noisy and continuously changing environments. In: Hamza MH (ed) Artificial intelligence and soft computing. IASTED/ACTA Press, Anaheim, pp 289–294
98. Parsopoulos KE, Vrahatis MN (2002) Particle swarm optimization method for constrained optimization problems. In: Sincak P, Vascak J, Kvasnicka V, Pospichal J (eds) Intelligent technologies-theory and application: new trends in intelligent technologies. Frontiers in artificial intelligence and applications, vol 76. IOS Press, pp 214–220
99. Parsopoulos KE, Vrahatis MN (2002) Particle swarm optimization method in multiobjective problems. In: Proceedings of the ACM 2002 symposium on applied computing (SAC 2002), Madrid. ACM Press, pp 603–607
100. Parsopoulos KE, Vrahatis MN (2002) Recent approaches to global optimization problems through particle swarm optimization. Nat Comput 1(2-3):235–306
101. Parsopoulos KE, Vrahatis MN (2004) On the computation of all global minimizers through particle swarm optimization. IEEE Trans Evol Comput 8(3):211–224
102. Parsopoulos KE, Vrahatis MN (2004) UPSO: a unified particle swarm optimization scheme. In: Proceedings of the international conference of computational methods in sciences and engineering (ICCMSE 2004). Lecture series on computer and computational sciences, vol 1. VSP International Science Publishers, Zeist, pp 868–873
103. Parsopoulos KE, Vrahatis MN (2006) Studying the performance of unified particle swarm optimization on the single machine total weighted tardiness problem. In: Sattar A, Kang BH (eds) Lecture notes in artificial intelligence (LNAI), vol 4304. Springer, Berlin/New York, pp 760–769
104. Parsopoulos KE, Vrahatis MN (2007) Parameter selection and adaptation in unified particle swarm optimization. Math Comput Model 46(1–2):198–213
105. Parsopoulos KE, Vrahatis MN (2008) Multi-objective particles swarm optimization approaches. In Bui LT, Alam S (eds) Multi-objective optimization in computational intelligence: theory and practice. Premier reference source, chapter 2. Information Science Reference (IGI Global), Hershey, pp 20–42
106. Parsopoulos KE, Vrahatis MN (2010) Particle swarm optimization and intelligence: advances and applications. Inf Sci Publ (IGI Glob)

107. Petalas YG, Parsopoulos KE, Vrahatis MN (2007) Entropy-based memetic particle swarm optimization for computing periodic orbits of nonlinear mappings. In: IEEE 2007 congress on evolutionary computation (IEEE CEC 2007), Singapore. IEEE Press, pp 2040–2047
108. Petalas YG, Parsopoulos KE, Vrahatis MN (2007) Memetic particle swarm optimization. Ann Oper Res 156(1):99–127
109. Petalas YG, Parsopoulos KE, Vrahatis MN (2009) Improving fuzzy cognitive maps learning through memetic particle swarm optimization. Soft Comput 13(1):77–94
110. Piperagkas GS, Georgoulas G, Parsopoulos KE, Stylios CD, Likas CA (2012) Integrating particle swarm optimization with reinforcement learning in noisy problems. In: Genetic and evolutionary computation conference 2012 (GECCO 2012), Philadelphia. ACM, pp 65–72
111. Piperagkas GS, Konstantaras I, Skouri K, Parsopoulos KE (2012) Solving the stochastic dynamic lot-sizing problem through nature-inspired heuristics. Comput Oper Res 39(7):1555–1565
112. Poli R (2008) Analysis of the publications on the applications of particle swarm optimisation. J Artif Evol Appl 2008(3):1–10
113. Poli R (2008) Dynamic and stability of the sampling distribution of particle swarm optimisers via moment analysis. J Artif Evol Appl 2008(3):10010
114. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intell 1(1):33–57
115. Poli R, Langdon WB (2007) Markov chain models of bare-bones particle swarm optimizers. ACM, New York, pp 142–149
116. Pookpunt S, Ongsakul W (2013) Optimal placement of wind turbines within wind farm using binary particle swarm optimization with time-varying acceleration coefficients. Renew Energy 55:266–276
117. Potter MA, De Jong K (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29
118. Qu BY, Liang JJ, Suganthan PN (2012) Niching particle swarm optimization with local search for multi-modal optimization. Inf Sci 197:131–143
119. Rada-Vilela J, Johnston M, Zhang M (2014) Population statistics for particle swarm optimization: resampling methods in noisy optimization problems. Swarm Evol Comput 17:37–59
120. Rahnamayan RS, Tizhoosh HR, Salama MMA (2008) Opposition-based differential evolution. IEEE Trans Evol Comput 12(1):64–79
121. Reyes-Sierra M, Coello Coello CA (2006) Multi-objective particle swarm optimizers: a survey of the state-of-the-art. Int J Comput Intell Res 2(3):287–308
122. Rezaee Jordehi A, Jasni J (2013) Parameter selection in particle swarm optimisation: a survey. J Exp Theor Artif Intell 25(4):527–542
123. Rini DP, Shamsuddin SM, Yuhaniz SS (2014) Particle swarm optimization: technique, system and challenges. Int J Comput Appl 14(1):19–27
124. Schmitt M, Wanka R (2015) Particle swarm optimization almost surely finds local optima. Theor Comput Sci Part A 561:57–72
125. Schoeman IL, Engelbrecht AP (2010) A novel particle swarm niching technique based on extensive vector operations. Nat Comput 9(3):683–701
126. Schwefel H-P (1995) Evolution and optimum seeking. Wiley, New York
127. Sedighizadeh D, Masehian E (2009) Particle swarm optimization methods, taxonomy and applications. Int J Comput Theory Eng 1(5):486–502
128. Shi Y, Eberhart RC (1998) A modified particle swarm optimizer. In: Proceedings IEEE conference on evolutionary computation, Anchorage. IEEE Service Center, pp 69–73
129. Skokos Ch, Parsopoulos KE, Patsis PA, Vrahatis MN (2005) Particle swarm optimization: an efficient method for tracing periodic orbits in 3D galactic potentials. Mon Not R Astron Soc 359:251–260
130. Souravlias D, Parsopoulos KE (2016) Particle swarm optimization with neighborhood-based budget allocation. Int J Mach Learn Cybern 7(3):451–477. Springer
131. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359
132. Suganthan PN (1999) Particle swarm optimizer with neighborhood operator. In: Proceedings of the IEEE congress on evolutionary computation, Washington, DC, pp 1958–1961

133. Sun J, Feng B, Xu W (2004) Particle swarm optimization with particles having quantum behavior. In: Proceedings of the IEEE congress on evolutionary computation 2004 (IEEE CEC'04), Portland (OR), pp 325–331
134. Sun J, Lai C-H, Wu X-J (2011) Particle swarm optimisation: classical and quantum perspectives. CRC Press, Boca Raton
135. Sun J, Xu W, Feng B (2004) A global search strategy for quantum-behaved particle swarm optimization. In: Proceedings of the 2004 IEEE conference on cybernetics and intelligent systems, Singapore, pp 111–116
136. Sun S, Li J (2014) A two-swarm cooperative particle swarms optimization. Swarm Evol Comput 15:1–18
137. Sutton AM, Whitley D, Lunacek M, Howe A (2006) PSO and multi-funnel landscapes: how cooperation might limit exploration. In: Proceedings of the 8th annual conference on genetic and evolutionary computation (GECCO'06), Seattle, pp 75–82
138. Tasgetiren F, Chen A, Gencyilmaz G, Gattoufi S (2009) Smallest position value approach. Stud Comput Intell 175:121–138
139. Tasgetiren MF, Liang Y-C, Sevkli M, Gencyilmaz G (2006) Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. Int J Prod Res 44(22):4737–4754
140. Tasgetiren MF, Liang Y-C, Sevkli M, Gencyilmaz G (2007) A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. Eur J Oper Res 177(3):1930–1947
141. Thangaraj R, Pant M, Abraham A, Bouvry P (2011) Particle swarm optimization: hybridization perspectives and experimental illustrations. Appl Math Comput 217(12):5208–5226
142. Törn A, Žilinskas A (1989) Global optimization. Springer, Berlin
143. Trelea IC (2003) The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf Process Lett 85:317–325
144. Tsai H-C (2010) Predicting strengths of concrete-type specimens using hybrid multilayer perceptrons with center-unified particle swarm optimization. Expert Syst Appl 37(2): 1104–1112
145. Van den Bergh F, Engelbrecht AP (2002) A new locally convergent particle swarm optimiser. In: Proceedings of the 2002 IEEE international conference on systems, man and cybernetics, vol 3, pp 94–99
146. Van den Bergh F, Engelbrecht AP (2004) A cooperative approach to particle swarm optimization. IEEE Trans Evol Comput 8(3):225–239
147. Van den Bergh F, Engelbrecht AP (2006) A study of particle swarm optimization particle trajectories. Inf Sci 176:937–971
148. Voglis C, Parsopoulos KE, Lagaris IE (2012) Particle swarm optimization with deliberate loss of information. Soft Comput 16(8):1373–1392
149. Voglis C, Parsopoulos KE, Papageorgiou DG, Lagaris IE, Vrahatis MN (2012) MEMPSODE: a global optimization software based on hybridization of population-based algorithms and local searches. Comput Phys Commun 183(5):1139–1154
150. Wang H, Moon I, Yang S, Wang D (2012) A memetic particle swarm optimization algorithm for multimodal optimization problems. Inf Sci 197:38–52
151. Wang H, Wu Z, Rahnamayan S, Liu Y, Ventresca M (2011) Enhancing particle swarm optimization using generalized opposition-based learning. Inf Sci 181(20):4699–4714
152. Wang H, Zhao X, Wang K, Xia K, Tu X (2014) Cooperative velocity updating model based particle swarm optimization. Appl Intell 40(2):322–342
153. Wang Y-J, Zhang J-S (2008) A new constructing auxiliary function method for global optimization. Math Comput Modell 47(11–12):1396–1410
154. Wu H, Geng J, Jin R, Qiu J, Liu W, Chen J, Liu S (2009) An improved comprehensive learning particle swarm optimization and its application to the semiautomatic design of antennas. IEEE Trans Antennas Propag 57(10 PART 2):3018–3028
155. Xianfeng Y, Li LS (2014) Dynamic adjustment strategies of inertia weight in particle swarm optimization algorithm. Int J Control Autom 7(5):353–364

156. Xu W, Duan BY, Li P, Hu N, Qiu Y (2014) Multiobjective particle swarm optimization of boresight error and transmission loss for airborne radomes. IEEE Trans Antennas Propag 62(11):5880–5885
157. Xue B, Zhang M, Browne WN (2014) Particle swarm optimisation for feature selection in classification: novel initialisation and updating mechanisms. Appl Soft Comput J 18: 261–276
158. Yang J, Zhang H, Ling Y, Pan C, Sun W (2014) Task allocation for wireless sensor network using modified binary particle swarm optimization. IEEE Sens J 14(3):882–892
159. Yang J-M, Chen Y-P, Horng J-T, Kao C-Y (1997) Applying family competition to evolution strategies for constrained optimization. Lecture notes in mathematics, vol 1213. Springer, Berlin/New York, pp 201–211
160. Yen GG, Leong WF (2009) Dynamic multiple swarms in multiobjective particle swarm optimization. IEEE Trans Syst Man Cybern Part A Syst Hum 39(4):890–911
161. Yu X, Zhang X (2014) Enhanced comprehensive learning particle swarm optimization. Appl Math Comput 242:265–276
162. Zambrano-Bigiarini M, Clerc M, Rojas R (2013) Standard particle swarm optimisation 2011 at CEC-2013: a baseline for future PSO improvements. In: 2013 IEEE congress on evolutionary computation, Cancún, pp 2337–2344
163. Zhang Q, Wang Z, Tao F, Sarker BR, Cheng L (2014) Design of optimal attack-angle for RLV reentry based on quantum particle swarm optimization. Adv Mech Eng 6:352983
164. Zhang Y, Gong D, Hu Y, Zhang W (2015) Feature selection algorithm based on bare bones particle swarm optimization. Neurocomputing 148:150–157
165. Zhang Y, Gong D-W, Ding Z (2012) A bare-bones multi-objective particle swarm optimization algorithm for environmental/economic dispatch. Inf Sci 192:213–227
166. Zhang Y, Gong D-W, Sun X-Y, Geng N (2014) Adaptive bare-bones particle swarm optimization algorithm and its convergence analysis. Soft Comput 18(7):1337–1352
167. Zhao F, Li G, Yang C, Abraham A, Liu H (2014) A human-computer cooperative particle swarm optimization based immune algorithm for layout design. Neurocomputing 132: 68–78
168. Zhao J, Lv L, Fan T, Wang H, Li C, Fu P (2014) Particle swarm optimization using elite opposition-based learning and application in wireless sensor network. Sens Lett 12(2): 404–408
169. Zheng Y-J, Ling H-F, Xue J-Y, Chen S-Y (2014) Population classification in fire evacuation: a multiobjective particle swarm optimization approach. IEEE Trans Evol Comput 18(1):70–81

POPMUSIC

22

Éric D. Taillard and Stefan Voß

Contents

Introduction 688
POPMUSIC Template 688
POPMUSIC Adaptation for Location-Routing 691
Initial Solution 691
Part and Subproblem Definitions 693
Seed-Part Selection 693
Empirical Complexity 693
POPMUSIC Applications 694
Application of POPMUSIC to Clustering 694
Application of POPMUSIC to Max Weight Stable Set Problems 695
Application of POPMUSIC to Berth Allocation Problems 696
Related Approaches 698
Conclusion 699
References 700

Abstract

This chapter presents POPMUSIC, a general decomposition-based framework within the realm of metaheuristics and matheuristics that has been successfully applied to various combinatorial optimization problems. POPMUSIC is especially useful for designing heuristic methods for large combinatorial problems

É. D. Taillard (✉)
Embedded Information Systems, HEIG-VD, University of Applied Sciences and Arts of Western Switzerland, Yverdon-les-Bains, Delémont, Switzerland
e-mail: eric.taillard@heig-vd.ch

S. Voß
Faculty of Business Administration, Institute of Information Systems, University of Hamburg, Hamburg, Germany
e-mail: stefan.voss@uni-hamburg.de

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_31

that can be partially optimized. The basic idea is to optimize subparts of solutions until a local optimum is reached. Implementations of the technique to various problems show its broad applicability and efficiency for tackling especially large-size instances.

Keywords
Decomposition algorithm · Fix-and-optimize method · Large Neighbourhood Search · Large-scale optimization · Matheuristics · Metaheuristics

Introduction

A natural way to solve large optimization problems is to decompose them into independent subproblems that are solved with an appropriate method. However, such approaches may lead to solutions of moderate quality since the subproblems might have been created in a somewhat arbitrary fashion. While it is not necessarily easy to find an appropriate way to decompose a problem a priori, the basic idea of Partial OPtimization Metaheuristic Under Special Intensification Conditions (POPMUSIC) is to locally optimize subparts of a solution, a posteriori, once a solution to the problem is available. These local optimizations are repeated until a local optimum is found (and possibly beyond). POPMUSIC stands for "partial optimization metaheuristic under special intensification conditions" [24]. It is a template, or framework, for creating heuristics repeating partial optimization on a solution. To apply this template, special conditions must be fulfilled. Among these conditions, it is supposed that a solution is composed of parts that can be optimized almost independently. Thus, POPMUSIC may be seen as a "general purpose" framework which does not need too much problem-specific adaptations if a reasonable way of decomposing a problem, or a given incumbent solution, is at hand so that it can be readily applied to a wide spectrum of problem classes.

POPMUSIC typically applies to large-size problem instances, but the template was also applied successfully to smaller instances. Typical real-life problems usually contain a very large number of elements. The POPMUSIC template proposes a solution for dealing with problem instances of that size.

Next, we describe the POPMUSIC template followed by a survey of several successful applications. Moreover, we provide an overview on a few related approaches and close with some conclusions and options for future research.

POPMUSIC Template

The idea of decomposing problems of large size into smaller subproblems easier to solve is certainly very old [8]. In the context of metaheuristics and matheuristics, this can be accomplished, for instance, by using given incumbent solutions and optimizing parts of it. In general terms, one may distinguish soft fixing and hard fixing to attain problems of reduced size that may be solved separately. While in *hard fixing* some of the decision variables of a given problem are directly excluded

(e.g., by fixing them to a certain value), *soft fixing* refers to adding, say linear, constraints to the model to cut out all solutions from a problem that are beyond a certain distance from a given solution. In that way, the search can be intensified around those parts of a solution space that are not cut out.

To exemplify the POPMUSIC template in detail, we resort to the capacitated vehicle routing problem (CVRP) as one of the first specific applications of the template [25]. We emphasize this as POPMUSIC that can be well illustrated on the CVRP. In its simplest version, it consists of finding a set of vehicle tours, starting from a depot, servicing customers asking given quantities of goods and coming back to the depot. Knowing the vehicle capacity and all distances between customers, a set of tours, visiting each customer exactly once, is searched in such a way that the total distance of the tours is minimized.

If a solution is available, a tour can be considered as a part of the solution. Each tour can be interpreted as being independent from the others and can be optimized independently. Optimizing a tour consists of finding an optimal trip (or tour) of a traveling salesman problem (TSP). Since the number of customers on a tour of a CVRP is limited and exact codes for the TSP are available and efficient up to few dozens of cities and beyond [9], the optimization of single CVRP tours can be considered as a simple task.

Now, if we consider the subset of customers delivered by a subset of tours of a given solution, it is also possible to optimize independently the delivery of this subset of customers. So, a subset of parts of a solution creates a subproblem that can be optimized independently from the remaining of the problem. For a CVRP, such a subproblem is a CVRP of reduced size.

If an optimization method is available that can treat CVRP's with up to r tours efficiently, larger instances can be tackled by repeating optimizations on subproblems containing up to r tours. The POPMUSIC template suggests a tactic for building subproblems that can be potentially improved. Instead of generating any subproblem with r parts, as done by matheuristics or metaheuristics like large neighborhood search (LNS), POPMUSIC introduces a notion of proximity between parts. Subproblems are built by first selecting a part called a *seed-part* and r of its closest parts. Figure 1 illustrates the principle of the generation of a subproblem containing six tours.

Given a solution composed of p parts, there are only p subproblems composed of $r < p$ parts that can be so built, instead of the $\frac{p!}{(p-r)!\cdot r!}$ possible ones. In contrast to the LNS template where the user must choose at each step how to build a subproblem and when to stop the optimization process, POPMUSIC-based approaches have a natural stopping criterion: when the p subproblems are solved optimally or with satisfaction, the process can be stopped.

To be specific, in the POPMUSIC template, it is supposed that a solution s can be decomposed into p parts $s_1, \ldots, s_p$ and that a proximity measure between two parts has been defined. Let U with $|U| =: p$ be the set of parts that have not been used as seed-part for building a subproblem. Initially, U contains all p parts and U is empty at the end. Let $m(R)$ be an optimization method that allows to optimize a subproblem R composed of r parts. With these hypothesis, the POPMUSIC template can be expressed by Algorithm 1. (Note that in line 8, a slightly modified

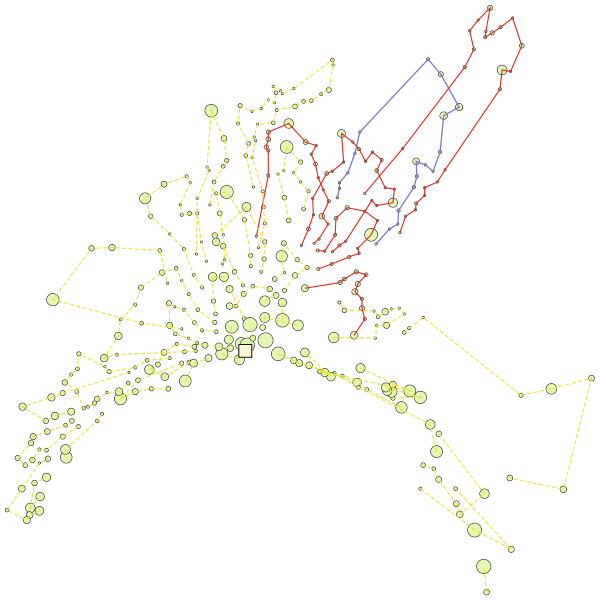


Fig. 1 For the vehicle routing problem, the definition of a part can be a vehicle tour. In this figure, the size of the disks that represents the customers' demand and the trips from and to the depot (*square*) are not drawn. Here, the proximity between parts is measured by the distance of their center of gravity. A subproblem is a smaller VRP composed of the customers serviced by six tours

Algorithm 1: POPMUSIC template.

Input: Initial solution s composed of p parts $s_1, \ldots, s_p$; subproblem optimization method m; ; sub-problem size r
Output: Improved solution s
$U = \{s_1, \ldots, s_p\}$
while $U \neq \emptyset$ **do**
  Select $s_g \in U$ `// seed-part`
  Build subproblem R with the r closest parts to s_g Optimize R with method m
  if R *improved* **then**
    Update solution s Remove from U the parts that do not belong to s anymore
    Insert in U the parts of optimized subproblem R
  else `//` R `not improved`
    Remove s_g of U
  end
end

version of the template may be obtained as follows. If one improves the solution, one can either remove all of the parts of s or insert them again.)

POPMUSIC can be seen from different perspectives: as mentioned above, it can be considered as a large neighborhood search [22] or as a soft fixing approach.

In this case, the neighborhood considered consists of finding the best way to reshape the elements of r parts. The neighborhood is qualified as large, in contrast to local search that modifies only few elements of a solution at each step, allowing explicitly a complete examination of the neighborhood. When all the elements of r parts must be optimized, it is not possible to proceed to an explicit examination of the whole neighborhood, since its size is much too large. So, either an exact method is used, making POPMUSIC a matheuristic (see [19]), or an approximate algorithm is used, typically based on a metaheuristic.

To derive a specific POPMUSIC, implementation is described by means of an explicit application in the next section.

POPMUSIC Adaptation for Location-Routing

The POPMUSIC template requires the programmer to define five components: the *initial solution*, a *part*, the *proximity* between parts, the *seed-part* choice, and an *optimization method*. This section illustrates how these components can be implemented for a location-routing problem (LRP). Very briefly, an LRP is a vehicle routing problem with several depots where the location of the depots must be decided in addition to the building of vehicle tours.

Initial Solution

A key point for the treatment of large problem instances is to analyze the algorithmic complexity of each step of the solution method. Examining the POPMUSIC template, the first question is about the number of times the `while` loop at line of Algorithm 1 is repeated. Numerical experiments on different problems – unsupervised clustering [26], map labeling [3], location-routing [4], and berth allocation [15, 16] – have shown that this loop is repeated a number of times that grows quasi-linearly with the problem size. This behavior is typical for local searches based on standard neighborhoods where the theoretical exponential complexity is not observed. To lower the algorithmic complexity, the other steps of the template must be carefully designed.

The step that looks most complex is naturally the optimization of subproblems at line . Indeed, the computational time for optimizing a subproblem composed of r parts grows very fast with r, for instance, exponentially if an exact method is used. The user can control the computational time by modifying the unique parameter r of the POPMUSIC template. Let us suppose that the user has fixed the value of r to a value for which the subproblems are sufficiently large for getting solutions of good quality while limiting the computational time of the optimization method. If r is fixed, solving a subproblem takes a constant time – maybe large, but constant. So, if the use of an efficient optimization method is a key point of the POPMUSIC template, this is not an issue for limiting the algorithmic complexity.

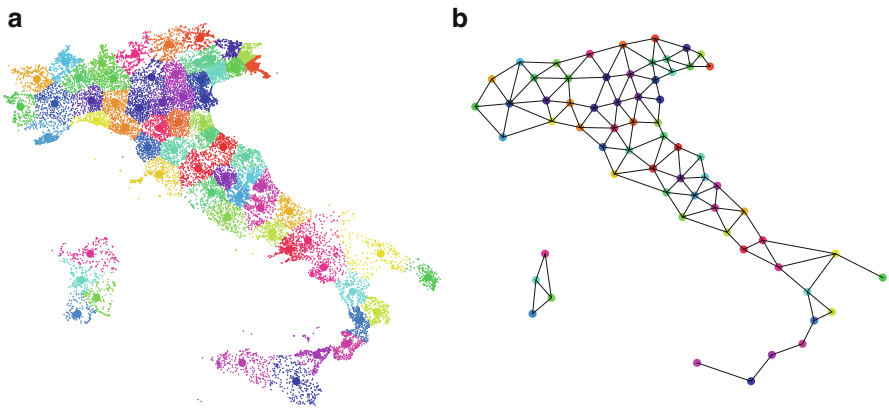


Fig. 2 Illustration of the decomposition of a problem into $\sqrt{n}$ clusters and building a proximity network between the clusters. (**a**) Problem decomposition into clusters. (**b**) Building proximity network

The choice of the seed-part can be implemented in constant time, for instance, by making a random choice or by storing set U as a stack or a queue. So, the key points for limiting the algorithmic complexity of a POPMUSIC implementation are to use an appropriate technique for producing the initial solution and for building subproblems from a seed-part. Indeed, without an appropriate data structure, building a subproblem can take a time proportional to the problem size, leading to a global complexity being at least quadratic.

Alvim and Taillard [4] presented how to build in $O(n^{3/2})$ a solution to a location-routing problem while building a data structure allowing to build a subproblem in $O(r)$. The idea of the construction is the following: Let us suppose that it is possible to advise a distance measure between the entities or solution components of the problem. For the LRP, this distance is part of the problem data: it is simply the distance between customers. The entities of the problem are grouped into $\sqrt{n}$ clusters by means of a heuristic method for the p-median problem. Although being NP-hard, the p-median problem can be approximately solved with an empirical complexity of $\overline{O}((p \cdot n + (\frac{n}{p})^2)$ where p is the number of desired clusters. In this problem, the clusters are built by choosing p central elements and assigning all remaining elements to the closest center. By choosing $p = \sqrt{n}$, it is possible to cut a problem of size n into $\sqrt{n}$ clusters containing approximately $\sqrt{n}$ elements in $\overline{O}(n^{3/2})$. This method is illustrated in Fig. 2.

Then, all centers can be compared 2 by 2 to create proximity relations. Figure 2b illustrates the proximity relation obtained by considering that clusters a and b are neighbors if an entity of one of these clusters has the center of the other one as the second closest center. Each cluster can then be decomposed into $O(\sqrt{n})$ clusters to create a total of $O(n)$ small clusters containing a constant number of elements. So, it is possible to create proximity relations between every entity of a problem of size n in $O(n^{3/2})$. In the context of the LRP, a vehicle tour can be assigned to the

entities of a small cluster. The proximity relations can be exploited in POPMUSIC for finding efficiently the closest r parts to a given seed-part.

However, if no versatile method for finding incumbent solutions is at hand, any type of heuristic or even a randomized approach may be used to derive them.

Part and Subproblem Definitions

The definition of a part strongly depends on the problem under consideration and on the procedure available for optimizing subproblems. For vehicle routing problems, [4, 20, 25] have defined the customers serviced by a vehicle tour as a part. This definition is justified by the fact that the subproblems built around a seed-part are smaller VRPs or multi-depot VRPs that can be efficiently optimized with a metaheuristic-based algorithm, like a basic tabu search or genetic algorithm hybridized with a variable neighborhood search. Several guidelines or proposals have been adopted for the proximity between parts: for Euclidean instances, the distance between the center of gravity of tours can be used; generally, the technique presented in section "Initial Solution" can be used for defining the proximity.

Seed-Part Selection

Very few works have studied the impact of the seed-part selection procedure. In many applications, subproblems are generated around a seed-part arbitrarily chosen in U (next available, random one, ...). Preliminary results in [4] have shown that a set U managed as a stack (last part entered in U is selected) seems to be better than an arbitrary selection.

Empirical Complexity

In [4], problem instances 10,000 times larger than those usually treated in the literature have been tackled. As mentioned above, the most complex part is the generation of an initial solution in $O(n^{3/2})$. The subproblem optimization is less complex, almost linear. Figure 3 shows the evolution of the computational effort as a function of problem size for different POPMUSIC phases (or iterations). The increase of computational time is higher for the steps of generating the initial solution (solving a p-median problem with two levels, as presented in section "Initial Solution") than for the subproblem optimization. However, this last phase still takes most of the computational effort, even with instances with two millions of entities.

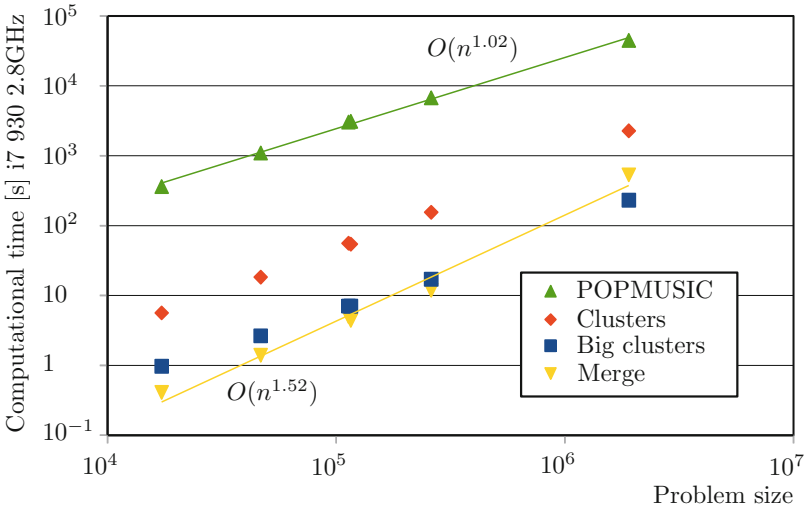


Fig. 3 Evolution of the computational effort as a function of problem size for various parts of the algorithm: building large clusters, decomposing large clusters into small ones containing subsets of customers that can be delivered by the same vehicle, finding positions for the depots, and optimizing subproblems

POPMUSIC Applications

While we have used the CVRP and a location-routing problem to clarify concepts, in this section, we summarize a few additional successful applications of POPMUSIC. Other applications not specifically mentioned refer, e.g., to the turbine runner balancing problem [24] or the p-cable trench problem [17].

Application of POPMUSIC to Clustering

The POPMUSIC template is particularly well adapted to solve large clustering problem instances. The goal of clustering is to create groups of entities as well separated as possible while containing entities as homogeneous as possible. This means that metrics are available for measuring the similarity or the dissimilarity between elements. If the number of clusters is relatively large, using the POPMUSIC template is interesting. In this case, a part of a solution can be constituted as the set of entities belonging to the same cluster. The measure of the separation between groups can be used for defining the proximity between parts.

This technique was successfully applied to unsupervised clustering with centroids (sum-of-squares clustering, p-median, multisource Weber) in [26]. Figure 4 illustrates the construction of a subproblem from a solution to a p-median problem.

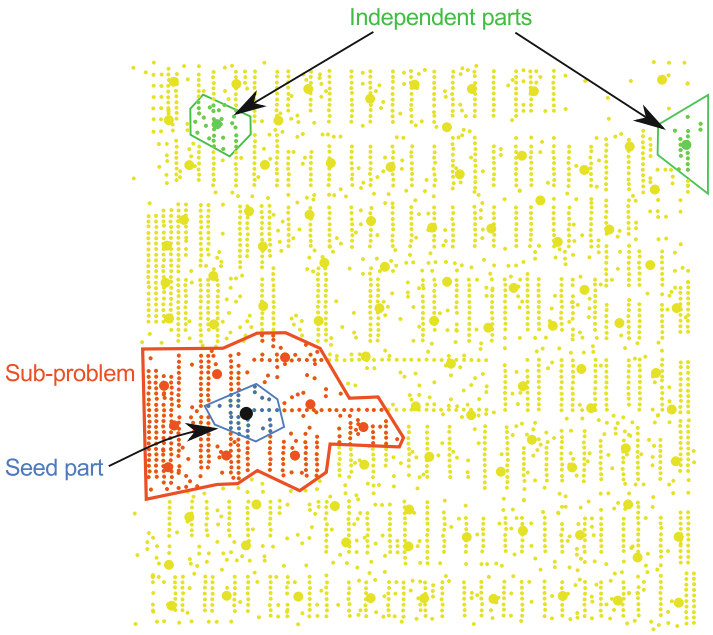


Fig. 4 Building a subproblem from a seed-part in the case of a p-median problem. Clusters that are well separated can be considered as independent and should not be involved in the same subproblem

Application of POPMUSIC to Max Weight Stable Set Problems

The POPMUSIC template is also well suited for the search of a stable set with maximum weight in a graph. Several practical problems can be modeled under this form. For instance, the map labeling problem can be formulated as a stable set with maximum weight. The problem is the following: given objects on the Euclidean plane, one wants to place a label around them to identify them. For instance, when drawing a map, an object can be the top of a mountain and label the name of the mountain. The label can be placed at several predefined positions. The problem consists of finding a position for each label in such a way that no label partially covers another one. If it is not possible to label all objects without superposition, it is searched to maximize the number of labels correctly placed. For typographic reasons, a different weight is associated to each label position, reflecting the positioning preferences or the importance of the object.

The problem can be formulated as the search of a stable set with maximum weight in a graph. The graph is built as follows: a node is associated with every label position, with a weight corresponding to the preference. An edge connects two nodes if the corresponding labels are incompatible: either they are associated with the same object (every object must be labeled at most once) or they overlap. Figure 5 illustrates the construction of a graph for labeling three objects with four possible label positions for each of them.

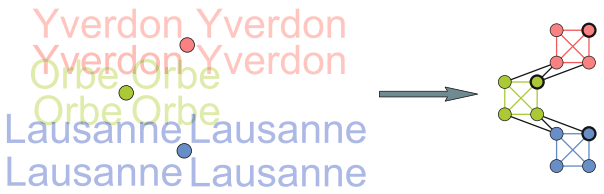


Fig. 5 Formulation of the problem of map labeling under a stable set

Applying the POPMUSIC template to this problem can be done as follows: a part is an object to label, so, the set of all nodes representing the different possible label positions for this object. The distance between two objects (interpreted as parts) is the minimum number of edges needed to connect two nodes associated to labels of the objects they identify. A subproblem is constituted by r objects for which the label position can be modified. A subproblem must also consider other labels of the current solution that may overlap, but without searching to modify the position of these labels.

The optimization procedure is a re-implementation of a tabu search proposed by [28] for which the parameters were tuned for solving at best problem instances with few dozens of objects. [2] have shown that excellent solutions can be obtained for problem instances with one thousand objects and that the method can still be used for instances with millions of objects.

This work was continued by [18] for allowing the labeling of cartographic maps containing non-punctual objects like lines (streets, rivers) or polygons (states, countries). The subproblem optimization used in this reference is based on ejection chains. The algorithms developed by [18] have been integrated in the QGIS open-source software (http://www.qgis.org/en/site/).

Various problems in the domain of transportation can be formulated as a map labeling (or a maximum weight stable set). The assignment of cruising levels of commercial aircraft is one of them. Knowing the departure hour and horizontal trajectory of every aircraft at a continent level, it is required to assign each of them a cruising level in order to avoid the collision between aircraft. The label shape is determined in this case by the possible position of the aircraft for a given time interval. The exact position of the aircraft cannot be determined, since the departure hour may fluctuate (traffic delay in the airport) and the weather conditions may affect the speed and position of the aircraft (wind, clouds, storm, etc.). For a time interval, it is required to find a level for each label in such a way that no label covers another one.

Application of POPMUSIC to Berth Allocation Problems

The POPMUSIC template was used in [15, 16] for the dynamic berth allocation problem with and without the consideration of tidal constraints. In those works, the optimization procedure is an exact method, thus applying POPMUSIC as a matheuristic.

The berth allocation problem (BAP) is a well-known optimization problem within maritime shipping. It aims at assigning and scheduling incoming vessels to berthing positions along the quay of a container terminal. Lalla-Ruiz and Voß[15] propose two POPMUSIC approaches that incorporate an existing mathematical programming formulation based on modeling the problem as a set partitioning problem. The computational experiments reveal state-of-the-art results outperforming all previous approaches regarding solution quality.

To be specific, [15] study the application of POPMUSIC for solving the discrete dynamic berth allocation problem (DBAP). In the DBAP, one is given a set of incoming container vessels, N, and a set of berths, M. Each container vessel, $i \in N$, must be assigned to an empty berth, $k \in M$, within its time window (TW), $[t_i, t_i']$, and the assigned berth time window, $[s_k, e_k]$. A simplified assumption is that each berth can handle at most one vessel at a time. For each container vessel, $i \in N$, its handling time, ρ_{ik}, depends on the berth $k \in M$ where it is assigned to, that is, the service time of a given vessel differs from one berth to another. Moreover, some vessels may have forbidden berths in order to model water-depth or maintenance constraints. Finally, each vessel $i \in N$ has a given service priority, denoted as v_i, according to its contractual agreement with the terminal.

A natural way to define parts is by means of the berths themselves. If a restricted number of berths (and assigned vessels) can be handled with an exact approach, this allows for efficient subproblem solving. Moreover, it is also easy to define the proximity between berths, taking real distances.

An interesting observation deduced from solving the DBAP and extensions by means of POPMUSIC also reveals some general lessons learned regarding the definition of parts. Let us assume a problem extension as the berth allocation problem under time-dependent limitations (BAP-TL) [16]. In this problem, in order to assign a vessel to a berth, terminal managers might have to take into account not only the berthing place and vessel draft but also the arrival and berthing time of an incoming vessel while observing changing environments due to tidal changes. Note that in the DBAP, the decisions about to which berth the vessel should be assigned to are relevant since there are different handling times depending on the berth. This is relaxed in the BAP-TL, i.e., all the berths provide the same handling time, having also an important implication on the proper definition of parts. Note also that in the BAP-TL, the berthing time is important due to the tidal constraints. Thus, it makes sense to define the parts not necessarily following the berths delimitation, but defining the parts as intervals of time. Numerical results are given in [16].

To end this subsection, we should point out an interesting analogy which may be drawn between the pure (and possibly simplified) berth allocation problem and the map labeling problem. In this case, the width of a label corresponds to the time interval during which the vessel must be at the berth for being loaded/unloaded. The height of the label is the vessel length. A label can be placed at different positions along the berth. Figure 6 provides an example of the transformation of a small problem with four vessels (represented by different colors or shades of gray) arriving at different times and that can be placed at few different positions along a berth.

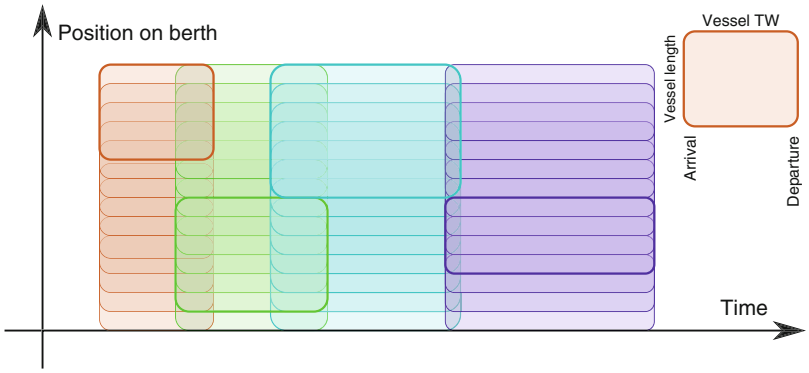


Fig. 6 Transformation of a berth allocation problem instance to a map labeling one

Related Approaches

For large optimization problems, it is often possible to interpret a solution as composed of parts or chunks [27] as can also be found under the term vocabulary building. Suppose that a solution can be represented as a set of parts as seen above, some parts are more in relation with some other parts so that a corresponding heuristic measure can be defined between two parts.

The *corridor method* (CM) has been presented by [23] as a hybrid metaheuristic, linking mathematical programming techniques with heuristic schemes. The basic idea of the CM relies on the use of an exact method over restricted portions of the solution space of a given problem. Given an optimization problem, the basic ingredients of the method are a very large feasible space and an exact method that could easily solve the problem if the feasible space was not large.

The basic concept of a *corridor* is introduced to delimit a portion of the solution space around the incumbent solution. The optimization method is then applied within the neighborhood defined by the corridor with the aim of finding an improved solution. Consequently, the CM defines method-based neighborhoods, in which a neighborhood is built taking into account the method used to explore it.

As mentioned before, POPMUSIC may be seen as a soft fixing approach as does the CM. Food for thought might be considered putting POPMUSIC into perspective regarding other methods like large neighborhood search, adaptive randomized decomposition, kernel search, etc. An older research which may serve as motivation relates to solving the job shop problem by means of the shifting bottleneck heuristic [1]. The idea of developing heuristics identifying a small/moderate size subset of variables in order to intensify the search in a promising region of the solution space has been used in other contexts. In the knapsack problem family, for instance, [7] propose the idea of selecting a small subset of items (called the core) and solving exactly a restricted problem on that subset. The use of an expanding method to modify the size of the core during the algorithm execution is proposed by [21].

A heuristic framework called kernel search has been proposed for the solution of mixed integer linear problems in [5, 6]. The kernel search framework is based on the idea of exhaustively exploring promising portions of the solution space. Kernel search is similar to POPMUSIC since "buckets," analogous to parts, are defined at the beginning of the search. During the execution, the algorithm revises the definition of the core problem, called "kernel," by adding/removing buckets to/from the current problem. Once the current kernel is defined, an exact method is applied to the restricted problem.

Local branching [10] is another soft fixing technique, in which the introduction of linear constraints is used to cut solutions from the feasible space. The feasible space of the constrained problem includes only solutions that are not too far away from the incumbent. When presented, local branching was used within a branch and bound framework. A generalized local branching concept has been proposed by [14].

Variable neighborhood decomposition search is in line with the strategy employed by POPMUSIC. Neighborhoods around an incumbent are defined using a distance metric and explored via any method. If a local optimum better than the incumbent is found in the current neighborhood, the neighborhood is re-centered around the new incumbent, and the mechanism moves on to the exploration of the new neighborhood. If no improvement occurs, the search moves to the next neighborhood defined around the same incumbent. An interesting modification of variable neighborhood decomposition is provided in [13], where variable neighborhood search is coupled with local branching. Neighborhoods are defined imposing linear constraints on a mixed integer linear programming model, as done in local branching, and then explored using a general purpose solver as black box.

Fischetti et al. [11] propose a similar method called diversification, refining, and tight-refining (DRT). It is aimed at solving problems with two-level variables, in which fixing the value of the first-level variables leads to an easier to solve, but still hard, subproblem. Finally, another way to see POPMUSIC is to exploit the proximate optimality principle (POP, see, e.g., [12]). In tabu search, the POP notion is exploited by performing a number of iterations at a given level before restoring the best solution found to initiate the search at the next level. In that context, a level corresponds to the optimization of a subproblem in the POPMUSIC terminology.

Conclusion

The main strengths of POPMUSIC are its simplicity, the fact that it has a unique parameter, and its ability to solve large problem instances. So, the main effort when implementing a POPMUSIC-based procedure is not devoted to parameter tuning as it can be the case for other metaheuristics. But there is no free lunch. The requisite of POPMUSIC is that an initial solution with a structure adapted to the template must be available as well as an optimization procedure for the subproblems. These points may make POPMUSIC less general and more problem-specific than other metaheuristics. However, this can be outweighed by the fact that exact approaches

may be applicable within the framework. Adapting POPMUSIC to various hard problems which are not easy to decompose is a first research avenue.

Up to now, very few works have been devoted to study the influence of POPMUSIC options, such as the way the seed-part is chosen or the procedure used for optimizing subproblems. These are other promising research areas for the future. It can also be interesting to revisit existing methods that are somewhat based on decomposition principles under the form of the POPMUSIC template. (An example would be the well-known Lagrangean decomposition concept.) This could lead to simplifications and/or efficiency improvements.

The POPMUSIC template may be adapted to parallel implementations. Our first works on the vehicle routing problem were primarily devoted to parallel implementations of metaheuristics. A relatively easy issue is to use the few cores of CPU in parallel. However, implementing a POPMUSIC with a number of processors dependent on the problem size is less trivial. Another research avenue is to see how to exploit the large number of processors of graphic process units in the context of POPMUSIC.

References

1. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manag Sci 34:391–401
2. Alvim ACF, Taillard ÉD (2007) An efficient POPMUSIC based approach to the point feature label placement problem. In: Metaheuristic International Conference (MIC'07) Proceedings.
3. Alvim ACF, Taillard ÉD (2009) POPMUSIC for the point feature label placement problem. Eur J Oper Res 192(2):396–413
4. Alvim ACF, Taillard ÉD (2013) POPMUSIC for the world location routing problem. EURO J Transp Logist 2:231–254
5. Angelelli E, Mansini R, Speranza M (2010) Kernel search: a general heuristic for the multi-dimensional knapsack problem. Comput Oper Res 37(11):2017–2026
6. Angelelli E, Mansini R, Speranza M (2012) Kernel search: a new heuristic framework for portfolio selection. Comput Optim Appl 51(1):345–361.
7. Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problems. Oper Res 28(5):1130–1154
8. Ball MO (2011) Heuristics based on mathematical programming. Surv Oper Res Manag Sci 16(1):21–38
9. Concorde (2015) Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde/index.html
10. Fischetti M, Lodi A (2003) Local branching. Math Program B 98:23–47
11. Fischetti M, Polo C, Scantamburlo M (2004) A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. Networks 44(2):61–72
12. Glover F, Laguna M (1997) Tabu search. Kluwer, Dordrecht
13. Hansen P, Mladenović N, Urosević D (2006) Variable neighborhood search and local branching. Comput Oper Res 33(10):3034–3045
14. Hill A, Voß S (2015) Generalized local branching heuristics and the capacitated ring tree problem. Working paper, IWI, University of Hamburg
15. Lalla-Ruiz E, Voß S (2016) POPMUSIC as a matheuristic for the berth allocation problem. Ann Math Artif Intell 76:173–189

16. Lalla-Ruiz E, Voß S, Exposito-Izquierdo C, Melian-Batista B, Moreno-Vega JM (2015) A POPMUSIC-based approach for the berth allocation problem under time-dependent limitations. Ann Oper Res 1–27. doi:10.1007/s10479-015-2055-6, ISSN:1572-9338. Page online available http://dx.doi.org/10.1007/s10479-015-2055-6
17. Lalla-Ruiz E, Schwarze S, Voß S (2016) A matheuristic approach for the p-cable trench problem. Working paper, IWI, University of Hamburg
18. Laurent M, Taillard ÉD, Ertz O, Grin F, Rappo D, Roh S (2009) From point feature label placement to map labelling. In: Metaheuristic International Conference (MIC'09) Proceedings.
19. Maniezzo V, Stützle T, Voß S (eds) (2009) Matheuristics: hybridizing metaheuristics and mathematical programming. Springer, Berlin
20. Ostertag A, Doerner KF, Hartl RF, Taillard ÉD, Waelti P (2009) POPMUSIC for a real-world large-scale vehicle routing problem with time windows. J Oper Res Soc 60(7):934–943
21. Pisinger D (1999) Core problems in knapsack algorithms. Oper Res 47(4):570–575
22. Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Comput Oper Res 34(8):2403–2435
23. Sniedovich M, Voß S (2006) The corridor method: a dynamic programming inspired metaheuristic. Control Cybern 35:551–578
24. Taillard E, Voß S (2002) POPMUSIC—partial optimization metaheuristic under special intensification conditions. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer, Boston, pp 613–629
25. Taillard ÉD (1993) Parallel iterative search methods for vehicle routing problems. Networks 23(8):661–673
26. Taillard ÉD (2003) Heuristic methods for large centroid clustering problems. J Heuristics 9(1):51–73. Old technical report IDSIA-96-96
27. Woodruff D (1998) Proposals for chunking and tabu search. Eur J Oper Res 106:585–598
28. Yamamoto M, Camara G, Lorena L (2002) Tabu search heuristic for point-feature cartographic label placement. GeoInformatica 6:77–90

Random-Key Genetic Algorithms

23

José Fernando Gonçalves and Mauricio G. C. Resende

Contents

Introduction 704
Biased Random-Key Genetic Algorithms 705
A Model for the Implementation of a BRKGA 705
Restarting a Random-Key Genetic Algorithm 707
RKGA with Multiple Populations 709
Specifying a RKGA 710
API for BRKGA 711
Conclusions 711
Cross-References 712
References 712

Abstract

A random-key genetic algorithm is an evolutionary metaheuristic for discrete and global optimization. Each solution is encoded as an array of n random keys, where a random key is a real number, randomly generated, in the continuous interval $[0, 1)$. A decoder maps each array of random keys to a solution of the optimization problem being solved and computes its cost. The algorithm starts with a population of p arrays of random keys. At each iteration, the arrays are partitioned into two sets, a smaller set of high-valued elite solutions and the remaining nonelite solutions. All elite elements are copied, without change, to the

J. F. Gonçalves (✉)
INESC TEC, Porto, Portugal

Faculdade de Economia da, Universidade do Porto, Porto, Portugal
e-mail: jfgoncal@fep.up.pt

M. G. C. Resende
Amazon.com, Inc. and University of Washington, Seattle, WA, USA
e-mail: resendem@amazon.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_30

next population. A small number of random-key arrays (the mutants) are added to the population of the next iteration. The remaining elements of the population of the next iteration are generated by combining, with the parametrized uniform crossover of Spears and DeJong (On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms, San Mateo, pp 230–236, 1991), pairs of arrays. This chapter reviews random-key genetic algorithms and describes an effective variant called biased random-key genetic algorithms.

Keywords
Random keys · Biased · Genetic algorithms

Introduction

Bean [6] described a new class of genetic algorithms for combinatorial optimization problems whose solutions can be represented by a permutation. These algorithms, called *random-key genetic algorithms* (RKGA), represent a solution of the optimization problem as an array of random keys. A *random key* is a real number, generated at random in the continuous interval $[0, 1)$.

A *decoder* is a procedure that maps an array of random keys into a solution of the optimization problem and computes the cost of this solution. The decoder proposed by Bean [6] simply orders the elements of the array of random keys, thus producing a permutation corresponding to the indices of the sorted elements.

A RKGA evolves a population, or set, of p arrays of random keys applying the Darwinian principle of survival of the fittest, where the fittest individuals (or solutions) of a population are more likely to find a mate and pass on their genetic material to future generations. The algorithm starts with an initial population of p arrays of n random keys and produces a series of populations. In the k-th generation, the p arrays of the population are partitioned into a small set of $p_e < p/2$ arrays corresponding to the best solutions (this set is called the *elite* set) and another set with the remainder of the population (called the *nonelite* set). All elite arrays are copied, unchanged, to the population of the $k + 1$-st generation. This elitism characterizes the Darwinian principle in an RKGA. Next, p_m arrays of random keys are introduced into the population of the $k + 1$-st generation. These arrays, called *mutants* or *immigrants*, play the same role as the mutation operators of classical genetic algorithms, i.e., they help avoid convergence of the population to a non-global local optimum. To complete the p elements of the population of the $k + 1$-st generation, $p - p_e - p_m$ arrays are generated, combining pairs of arrays from the population of the k-th generation with a parametrized uniform crossover [59]. Let a and b be the arrays chosen for mating and let c be the offspring produced. In the crossover of Spears and DeJong [59], $c[i]$, the i-th component of the offspring array receives the i-th key of one of its parents. It receives the key $a[i]$ with probability ρ_a and $b[i]$ with probability $\rho_b = 1 - \rho_a$.

Biased Random-Key Genetic Algorithms

As seen in section "Introduction" of this chapter, Bean's algorithm limits itself to elitism to simulate Darwinism. A *biased random-key genetic algorithm* (or BRKGA [24]), on the other hand, not only uses elitism to simulate survival of the fittest but also makes use of mating. A BRKGA differs from Bean's algorithm in the way parents are selected for crossover and how crossover is applied.

Both algorithms choose parents at random and with replacement. This way a parent can have more than one offspring per generation. While in Bean's algorithm both parents are chosen from the entire population, in a BRKGA one parent is always chosen from the elite set, while the other is chosen from the nonelite set (or, in some cases, from the entire population). Since $p_e < p/2$, an elite array in a BRKGA has a probability of $1/p_e$ of being selected for each crossover. This is greater than $1/(p - p_e)$, the probability that a nonelite array has on being selected. For the same reason, the probability that a specific elite array is chosen in a BRKGA is greater than $1/p$, the probability that a given elite array is chosen in Bean's algorithm.

Both algorithms combine parents a and b using parametrized uniform crossover [59] to produce the offspring c. While in Bean's algorithm each parent can be parent a or b, in a BRKGA a is always the elite parent, and b is the nonelite parent. Since $\rho_a > 1/2$, in a BRKGA the offspring c has greater probability of inheriting the keys of the elite parent, while in Bean's algorithm this is not necessarily true.

This small difference between the two algorithms almost always results in BRKGA outperforming Bean's algorithm [34]. Figure 1 compares iteration count distributions to a given target value for a BRKGA and an implementation of Bean's for a covering by pairs problem [7]. The figure clearly shows the dominance of the biased variant of the RKGA over the unbiased variant on this instance and for this target value. Though there has been at least one instance where the unbiased variant was slightly superior to the biased variant, the dominance of the biased variant over the unbiased variant is well established [34].

A Model for the Implementation of a BRKGA

Algorithm 1 shows a pseudo-code of a BRKGA for the minimization of $f(x)$, where $x \in X$ and X is a discrete set of solutions and $f : X \to \mathbb{R}$. This implementation is a multi-start variant of a BRKGA where several populations are evolved in sequence and a best solution among all in the population is returned as the output of the algorithm. After describing the pseudo-code, we will justify its multi-start nature.

In line 2, the value f^* of the best solution found is initialized to a large value, i.e., not smaller than $f(x^0)$, where $x^0 \in X$ is some feasible solution to the problem. Evolution of each population is done in lines 3–28. The algorithm halts when some stopping criterion in line 3 is satisfied. This criterion can be, for example, number of evolved populations, total time, or quality of the best solution found.

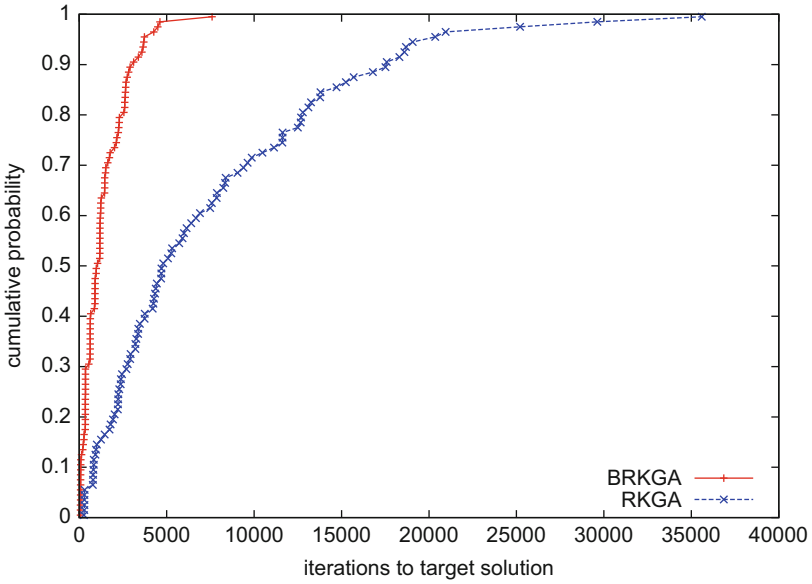


Fig. 1 Iteration count distributions to a given target solution value for a BRKGA and Bean's algorithm

In line 4, the population being evolved is initialized with $p = |\mathcal{P}|$ arrays of random keys. Evolution of population $\mathcal{P}$ takes place in lines 5–27. This evolution ends when a restart criterion is satisfied in line 5. This criterion can be, for example, a maximum number of generations without improvement in the value of the best solution in $\mathcal{P}$. At each generation, or iteration, the following operations are carried out: in line 6 all new solutions (offspring and mutants) are decoded and their costs evaluated. Note that each decoding and evaluation in this step can be computed simultaneously, i.e., in parallel. In line 7, population $\mathcal{P}$ is partitioned into two subpopulations $\mathcal{P}_e$ (elite) and $\mathcal{P}_{\bar{e}}$ (nonelite), where $\mathcal{P}_e$ is such that $|\mathcal{P}_e| < |\mathcal{P}|/2$ and contains $|\mathcal{P}_e|$ of the best solutions in $\mathcal{P}$ and $\mathcal{P}_{\bar{e}}$ consists of the remaining solutions in $\mathcal{P}$, that is, $\mathcal{P}_{\bar{e}} = \mathcal{P} \setminus \mathcal{P}_e$. $\mathcal{P}^+$ is the population of the next generation. It is initialized in line 8 with the elite solutions of the current generation. In line 9, the mutant subpopulation $\mathcal{P}_m$ is generated. Each mutant is an array of n random keys. The number of generated mutants in general is such that $|\mathcal{P}_m| < |\mathcal{P}|/2$. This subpopulation is added to population $\mathcal{P}^+$ of the next generation in line 10.

With $|\mathcal{P}_e| + |\mathcal{P}_m|$ arrays inserted in population $\mathcal{P}^+$, it is necessary to generate $|\mathcal{P}| - |\mathcal{P}_2| - |\mathcal{P}_m|$ new offspring to complete the $|\mathcal{P}|$ arrays that form population $\mathcal{P}^+$. This is done in lines 11–20. In lines 12 and 13, parents a and b are chosen, respectively, at random from subpopulations $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$. The generation of offspring c from parents a and b takes place in lines 14–18. A biased coin (with probability $\rho_a > 1/2$ of flipping to heads) is thrown n times. If the i-th toss is a heads, the offspring inherits the i-th key of parent a. Otherwise, it inherits the i-th key of parent b. After the offspring is generated, c is added to population $\mathcal{P}^+$ in line 19.

Algorithm 1: Model for biased random-key genetic algorithm with restart

```
BRKGA(|P|, |P_e|, |P_m|, n, ρ_a)
Initialize value of the best solution found: f* ← ∞;
while stopping criterion not satisfied do
    Generate a population P with n arrays of random keys;
    while restart criterion not satisfied do
        Evaluate the cost of each new solution in P;
        Partition P into two sets: P_e and P_ē;
        Initialize population of next generation: P+ ← P_e;
        Generate set P_m of mutants, each mutant with n random keys;
        Add P_m to population of next generation: P+ ← P+ ∪ P_m;
        foreach i ← 1 to |P| − |P_e| − |P_m| do
            Select parent a at random from P_e;
            Select parent b at random from P_ē;
            foreach j ← 1 to n do
                Throw a biased coin with probability ρ_a > 0.5 of resulting heads;
                if heads then c[j] ← a[j] ;
                else c[j] ← b[j];
            end
            Add offspring c to population of next generation: P+ ← P+ ∪ {c};
        end
        Update population: P ← P+;
        Find best solution χ+ in P: χ+ ← argmin{f(χ) | χ ∈ P};
        if f(χ+) < f* then
            χ* ← χ+;
            f* ← f(χ+);
        end
    end
end
return χ*
```

The generation of $\mathcal{P}^+$ ends when it consists of $|\mathcal{P}|$ elements. In line 21, $\mathcal{P}^+$ is copied to $\mathcal{P}$ to start a new generation. The best solution in the current population in evolution is computed in line 22 and if its value is better than all solutions examined so far, the solution and its cost are saved in lines 24 and 25 as χ^* and f^*, respectively. χ^*, the best solution found over all populations is returned by the algorithm in line 29.

Restarting a Random-Key Genetic Algorithm

As with most stochastic search methods, the continuous random variable *time to target solution* of a RKGA has an empirical distribution that approximates a shifted exponential distribution. The discrete random variable *iterations to target solution*, on the other hand, has an empirical shifted geometric distribution.

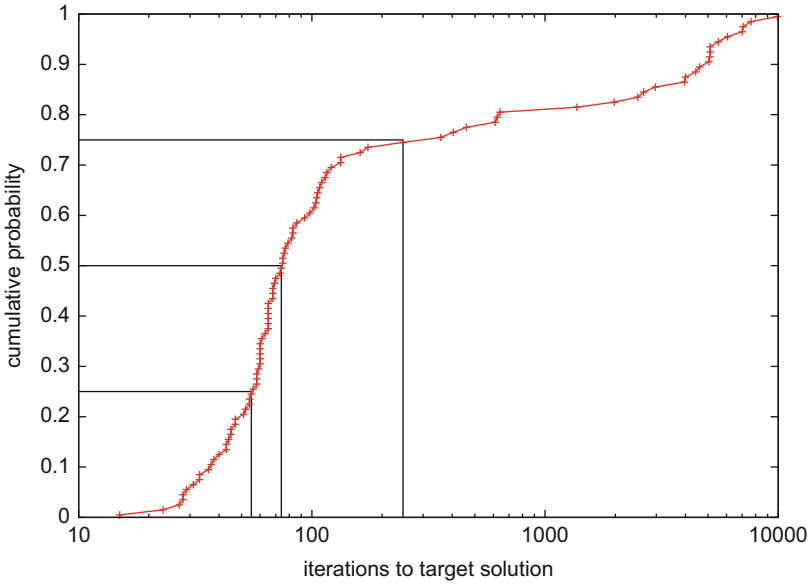


Fig. 2 Iteration count distribution to target solution of a BRKGA without restart

Consider, in Fig. 2, the empirical distribution of number of iterations of a BRKGA to find an optimal solution of instance of Steiner triple covering problem `stn243` [51].

The *iterations-to-target-solution plot* [1] is generated by running the BRKGA 100 times, each time using a different seed for the random number generator and recording the number of iterations that the algorithm took to find a solution as least as good as the target (in this case an optimal solution). The figure shows that 25% of the runs needed no more than 55 iterations to find an optimal solution, 50% took at most 74 iterations, and 75 % at most 245. However, 10% of the runs required more than 4597 iterations, 5% more than 5532 iterations, 2% more than 7061, and the longest run took 9903 iterations. This is the typical behavior of a random variable com a shifted geometric distribution.

Let I be the random variable number of iterations to a given target solution. For instance `stn243`, a visual examination of Fig. 2 suggests that $\Pr(I \geq 246) \approx 1/4$. Restarting the algorithm after 246 iterations and assuming independence of the runs, $\Pr(I \geq 492 \mid I \geq 246) \approx 1/4$. Therefore, $\Pr(I \geq 492) = \Pr(I \geq 246) \times \Pr(I \geq 492 \mid I \geq 246) \approx 1/4^2$. One can easily show, by induction, that the probability that the algorithm will take fewer than k cycles of 246 iterations is approximately $1/4^k$. For example, the probability that the algorithm with restart will take more than 1230 iterations (five cycles of 246 iterations between restarts) is approximately $1/4^5 = 1/1024 \approx 0.1\%$. This probability is considerably smaller than the approximately 20% probability that the algorithm without restart will take more than 1230 iterations.

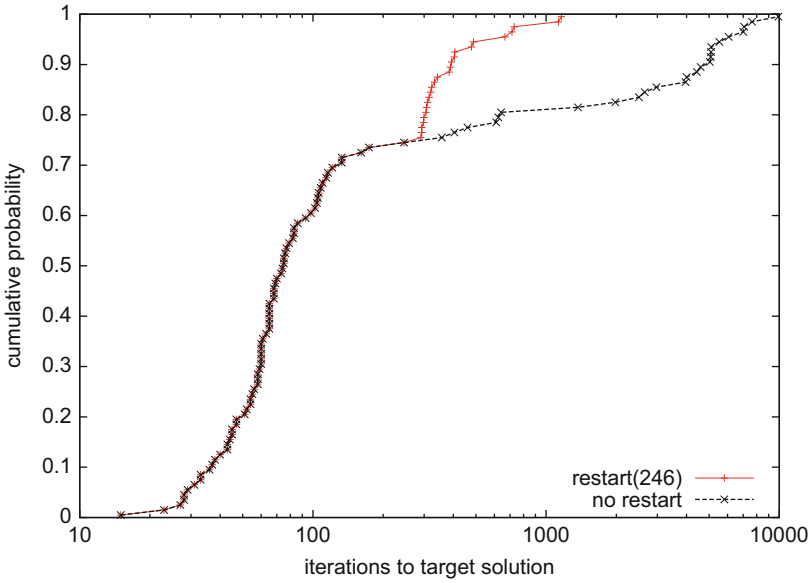


Fig. 3 Iteration count distribution to a target (optimal) solution of variants of BRKGA with and without restart on Steiner triple covering instance `stn243`

The above analysis uses a restart strategy that differs slightly from the one proposed here for random-key genetic algorithms. In the proposed strategy, similar to the restart strategy for GRASP with path relinking proposed by Resende and Ribeiro [50], instead of restarting each k iterations, it restarts after k_r iterations without improvement of the value of the best solution found since the previous restart.

Figure 3 compares a BRKGA without restart with one which restarts every 246 iterations without improvement of the value of the best solution found on Steiner triple covering instance `stn243`. The figure clearly shows that both the average number of iterations to an optimum and the corresponding standard deviation are smaller in the variant with restart than in the one without restart.

RKGA with Multiple Populations

The description of random-key genetic algorithms so far involved a single population. However, it is possible to implement a RKGA with more than one population [25].

Suppose that the RKGA has π populations, $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\pi$, each with p arrays of random keys. In this case, the π populations are initially populated, each independently of the others, with p arrays of random keys in line 4 of

the pseudo-code of Algorithm 1, and the loop in lines 6–26 is applied to each of these π populations. The populations exchange information every k_p iterations of the loop in lines 5–27 of Algorithm 1. In this exchange, the k_m best solutions from each population replace the $(\pi - 1)k_m$ worst solutions of each population.

Specifying a RKGA

The specification of a RKGA requires defining how a solution is represented, or encoded, how decoding is done, and what are the parameters of the algorithm.

Since each solution is represented as an array of n random keys, it is necessary only to specify a value for n.

The decoder is a deterministic algorithm that takes as input an array of n random keys and produces as output a solution of the optimization problem as well as its corresponding cost. A decoder is, in general, a heuristic. If it makes use of local search, then it is recommended but not strictly necessary that an array adjustment procedure be specified such that when the decoder is applied to an array of random keys corresponding to a local optimum, the decoder will produce the local optimum without applying the local search phase of the decoder. See Resende et al. [51] for a simple example of array adjustment.

Several parameters need to be specified. Table 1 lists these parameters and offers value ranges which in practice have proven to be satisfactory [24].

Table 1 Parameters and recommended values

Parameter	Recommended value
p: size of population	$p = \max\{3, \lfloor \kappa_p \times n \rfloor\}$, where $\kappa_p > 0$
p_e: size of elite partition of population	$p_e = \max\{1, \lfloor \kappa_e \times p \rfloor\}$, where $\kappa_e \in [0.10, 0.25]$
p_m: size of mutant partition of population	$p_m = \max\{1, \lfloor \kappa_m \times p \rfloor\}$, where $\kappa_m \in [0.05, 0.20]$
ρ_a: probability of inheriting key from elite parent	$\rho_a > 1/2$
k_r: iterations without improvement for restart	$k_r = \text{argmin}\ \{\Pr(k \text{ iterations to target solution}) \geq 0.75\}$
π: number of parallel populations	$\pi \in \{1, \ldots, 5\}$
k_p: frequency for population interchange	$k_p \in \{50, \ldots, 100\}$
k_m: number of exchanged solutions	$k_m \in \{1, 2, 3\}$
Stopping criterion (examples)	Running time
	Maximum number of iterations
	Maximum number of restarts
	Finding a solutions as good

API for BRKGA

To simplify the implementation of BRKGAs, Toso and Resende [62] proposed an *application programming interface (API)*, or C++ library, for BRKGA. The API is efficient and easy to use. The library is portable and automatically deals with several aspects of the BRKGA, such as management of the population and of the evolutionary dynamics. The API is implemented in C++ and uses an object-oriented architecture. In systems with available *OpenMP* [45], the API enables parallel decoding of random-key arrays. The user only needs to implement the decoder and specify the stopping criteria, restart and population exchange mechanisms, as well as the parameters of the algorithm.

The API is open source and can be downloaded from http://github.com/rfrancotoso/brkgaAPI.

Conclusions

This chapter reviewed random-key genetic algorithms, covering both their unbiased and the biased variants. After introducing the algorithm of Bean [6], on which the BRKGA is based, the chapter points to two small differences between the two variants that lead to improved performance of the BRKGA with respect to Bean's original random-key genetic algorithm. A model for the implementation of a BRKGA is described, and issues such as restart and use of multiple populations are discussed. The chapter ends by illustrating how a BRKGA is specified and presents an C++ API for BRKGA that allows for easy implementation of the algorithm.

The BRKGA metaheuristic has been applied to many optimization problems, such as:

- *Telecommunications:* Ericsson et al. [15], Buriol et al. [8], Noronha et al. [44], Reis et al. [48], Ruiz et al. [53], Pedrola et al. [47], Goulart et al. [35], Resende [49], Morán-Mirabal et al. [41], Pedrola et al. [46], Duarte et al. [14], and Andrade et al. [3].
- *Transportation:* Buriol et al. [10], Grasas et al. [36], Stefanello et al. [60], and Lalla-Ruiz et al. [37].
- *Scheduling:* Gonçalves et al. [30, 31, 32], Valente et al. [64], Valente and Gonçalves [63], Mendes et al. [39], Tangpattanakul et al. [61], Gonçalves and Resende [28], and Marques et al. [38].
- *Packing/layout:* Gonçalves [20], Gonçalves and Resende [25, 26, 27, 29].
- *Clustering:* Festa [16] and Andrade et al. [5].
- *Covering:* Breslau et al. [7] and Resende et al. [51].
- *Network optimization:* Andrade et al. [2, 3], Buriol et al. [9], Fontes and Gonçalves [18, 19], Coco et al. [12], Ruiz et al. [54], and Coco et al. [13].
- *Power systems:* Roque et al. [52].

- *Industrial engineering:* Gonçalves and Beirão [22], Gonçalves and Almeida [21], Gonçalves and Resende [23], Moreira et al. [43], Morán-Mirabal et al. [42], Gonçalves et al. [33], and Chan et al. [11].
- *Automatic tuning of parameters in heuristics:* Festa et al. [17] and Morán-Mirabal et al. [40].
- *Combinatorial auctions:* Andrade et al. [4].
- *Global continuous optimization:* Silva et al. [55,56,57,58].

Cross-References

▸ Biased Random-Key Genetic Progamming
▸ Genetic Algorithms
▸ GRASP
▸ Multi-start Methods

Acknowledgments The first author was partially supported by funds granted by the ERDF through the program COMPETE and by the Portuguese government through the FCT – Foundation for Science and Technology, project PTDC/ EGE-GES/ 117692/ 2010.

References

1. Aiex RM, Resende MGC, Ribeiro CC (2007) TTTPLOTS: a perl program to create time-to-target plots. Optim Lett 1:355–366
2. Andrade DV, Buriol LS, Resende MGC, Thorup M (2006) Survivable composite-link IP network design with OSPF routing. In: Proceedings of the eighth INFORMS telecommunications conference, Dallas
3. Andrade CE, Miyazawa FK, Resende MGC (2013) Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem. In: Proceedings of genetic and evolutionary computation conference (GECCO). ACM, Amsterdam
4. Andrade CE, Miyazawa FK, Resende MGC, Toso RF (2013) Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. Technical report, AT&T Labs Research, Florham Park
5. Andrade CE, Resende MGC, Karloff HJ, Miyazawa FK (2014) Evolutionary algorithms for overlapping correlation clustering. In: Proceedings of genetic and evolutionary computation conference (GECCO'14), Vancouver, pp 405–412
6. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6:154–160
7. Breslau L, Diakonikolas I, Duffield N, Gu Y, Hajiaghayi M, Johnson DS, Karloff H, Resende MGC, Sen S (2011) Disjoint-path facility location: theory and practice. In: Proceedings of the thirteenth workshop of algorithm engineering and experiments (ALENEX11), San Francisco, pp 68–74
8. Buriol LS, Resende MGC, Ribeiro CC, Thorup M (2005) A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 46:36–56
9. Buriol LS, Resende MGC, Thorup M (2007) Survivable IP network design with OSPF routing. Networks 49:51–64
10. Buriol LS, Hirsch MJ, Querido T, Pardalos PM, Resende MGC, Ritt M (2010) A biased random-key genetic algorithm for road congestion minimization. Optim Lett 4:619–633

11. Chan FTS, Tibrewal RK, Prakash A, Tiwari MK (2015) A biased random key genetic algorithm approach for inventory-based multi-item lot-sizing problem. Proc Inst Mech Eng Part B J Eng Manuf 229(1):157–171
12. Coco AA, Noronha TF, Santos AC (2012) A biased random-key genetic algorithm for the robust shortest path problem. In: Proceedings of global optimization workshop (GO2012), Natal, pp 53–56
13. Coco AA, Abreu JCA Jr, Noronha TF, Santos AC (2014) An integer linear programming formulation and heuristics for the minmax relative regret robust shortest path problem. J Glob Optim 60(2):265–287
14. Duarte A, Martí R, Resende MGC, Silva RMA (2014) Improved heuristics for the regenerator location problem. Int Trans Oper Res 21:541–558
15. Ericsson M, Resende MGC, Pardalos PM (2002) A genetic algorithm for the weight setting problem in OSPF routing. J Comb Optim 6:299–333
16. Festa P (2013) A biased random-key genetic algorithm for data clustering. Math Biosci 245:76–85
17. Festa P, Gonçalves JF, Resende MGC, Silva RMA (2010) Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm. In: Festa P (ed) Experimental algorithms. Lecture notes in computer science, vol 6049. Springer, Berlin/Heidelberg, pp 338–349
18. Fontes DBMM, Gonçalves JF (2007) Heuristic solutions for general concave minimum cost network flow problems. Networks 50:67–76
19. Fontes DBMM, Gonçalves JF (2013) A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. Optim Lett 7(6): 1303–1324
20. Gonçalves JF (2007) A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. Eur J Oper Res 183:1212–1229
21. Gonçalves JF, Almeida J (2002) A hybrid genetic algorithm for assembly line balancing. J Heuristics 8:629–642
22. Gonçalves JF, Beirão NC (1999) Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional 19:123–137
23. Gonçalves JF, Resende MGC (2004) An evolutionary algorithm for manufacturing cell formation. Comput Ind Eng 47:247–273
24. Gonçalves JF, Resende MGC (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17:487–525
25. Gonçalves JF, Resende MGC (2011) A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. J Comb Optim 22:180–201
26. Gonçalves JF, Resende MGC (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 29:179–190
27. Gonçalves JF, Resende MGC (2013) A biased random-key genetic algorithm for a 2D and 3D bin packing problem. Int J Prod Econ 145:500–510
28. Gonçalves JF, Resende MGC (2014) An extended Akers graphical minimization method with a biased random-key genetic algorithm for job-shop scheduling. Int Tran Oper Res 21:215–246
29. Gonçalves JF, Resende MGC (2015) A biased random-key genetic algorithm for the unequal area facility layout problem. Eur J Oper Res 246(1):86–107
30. Gonçalves JF, Mendes JJM, Resende MGC (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167:77–95
31. Gonçalves JF, Mendes JJM, Resende MGC (2008) A genetic algorithm for the resource constrained multi-project scheduling problem. Eur J Oper Res 189:1171–1190
32. Gonçalves JF, Resende MGC, Mendes JJM (2011) A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. J Heuristics 17:467–486
33. Gonçalves JF, Resende MGC, Costa MD (2016) A biased random-key genetic algorithm for the minimization of open stacks problem. Int Trans Oper Res 23(1–2):25–46

34. Gonçalves JF, Resende MGC, Toso RF (2014) An experimental comparison of biased and unbiased random-key genetic algorithms. Pesquisa Operacional 34:143–164
35. Goulart N, de Souza SR, Dias LGS, Noronha TF (2011) Biased random-key genetic algorithm for fiber installation in optical network optimization. In: IEEE congress on evolutionary computation (CEC 2011). IEEE, New Orleans, pp 2267–2271
36. Grasas A, Lourenço HR, Pessoa LS, Resende MGC, Caballé I, Barba N (2014) On the improvement of blood sample collection at clinical laboratories. BMC Health Serv Res 14:Article 12
37. Lalla-Ruiz E, González-Velarde JL, Melián-Batista B, Moreno-Vega JM (2014) Biased random key genetic algorithm for the tactical berth allocation problem. Appl Soft Comput 22:60–76
38. Marques I, Captivo ME, Vaz Pato M (2014) Scheduling elective surgeries in a portuguese hospital using a genetic heuristic. Oper Res Health Care 3:59–72
39. Mendes JJM, Gonçalves JF, Resende MGC (2009) A random key based genetic algorithm for the resource constrained project scheduling problem. Comput Oper Res 36:92–109
40. Morán-Mirabal LF, González-Velarde JL, Resende MGC (2013) Automatic tuning of GRASP with evolutionary path-relinking. In: Proceedings of hybrid metaheuristics 2013 (HM 2013). Lecture notes in computer science, vol 7919. Springer, Ischia, pp 62–77
41. Morán-Mirabal LF, González-Velarde JL, Resende MGC, Silva RMA (2013) Randomized heuristics for handover minimization in mobility networks. J Heuristics 19:845–880
42. Morán-Mirabal LF, González-Velarde JL, Resende MGC (2014) Randomized heuristics for the family traveling salesperson problem. Int Trans Oper Res 21:41–57
43. Moreira MCO, Ritt M, Costa AM, Chaves AA (2012) Simple heuristics for the assembly line worker assignment and balancing problem. J Heuristics 18:505–524
44. Noronha TF, Resende MGC, Ribeiro CC (2011) A biased random-key genetic algorithm for routing and wavelength assignment. J Glob Optim 50:503–518
45. OpenMP (2013) http://openmp.org/wp/. Last visted on 11 May 2013
46. Pedrola O, Careglio D, Klinkowski M, Velasco L, Bergman K, Solé-Pareta J (2013) Metaheuristic hybridizations for the regenerator placement and dimensioning problem in sub-wavelength switching optical networks. Eur J Oper Res 224:614–624
47. Pedrola O, Ruiz M, Velasco L, Careglio D, González de Dios O, Comellas J (2013) A GRASP with path-relinking heuristic for the survivable IP/MPLS-over-WSON multi-layer network optimization problem. Comput Oper Res 40:3174–3187
48. Reis R, Ritt M, Buriol LS,, Resende MGC (2011) A biased random-key genetic algorithm for OSPF and DEFT routing to minimize network congestion. Int Trans Oper Res 18:401–423
49. Resende MGC (2012) Biased random-key genetic algorithms with applications in telecommunications. TOP 20:120–153
50. Resende MGC, Ribeiro CC (2011) Restart strategies for GRASP with path-relinking heuristics. Optim Lett 5:467–478
51. Resende MGC, Toso RF, Gonçalves JF, Silva RMA (2012) A biased random-key genetic algorithm for the Steiner triple covering problem. Optim Lett 6:605–619
52. Roque LAC, Fontes DBMM, Fontes FACC (2014) A hybrid biased random key genetic algorithm approach for the unit commitment problem. J Comb Optim 28:140–166
53. Ruiz M, Pedrola O, Velasco L, Careglio D, Fernández-Palacios J, Junyent G (2011) Survivable IP/MPLS-over-WSON multilayer network optimization. J Optic Commun Netw 3:629–640
54. Ruiz E, Albareda-Sambola M, Fernández E, Resende MGC (2013) A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. Technical report, AT&T Labs Research Technical, Florham Park
55. Silva RMA, Resende MGC, Pardalos PM, Gonçalves JF (2012) Biased random-key genetic algorithm for bound-constrained global optimization. In: Proceedings of global optimization workshop (GO2012), Natal, pp 133–136
56. Silva RMA, Resende MGC, Pardalos PM (2014) Finding multiple roots of box-constrained system of nonlinear equations with a biased random-key genetic algorithm. J Glob Optim 60(2):289–306

57. Silva RMA, Resende MGC, Pardalos PM (2015) A Python/C++ library for bound-constrained global optimization using biased random-key genetic algorithm. J Comb Optim 30(3):710–728
58. Silva RMA, Resende MGC, Pardalos PM, Facó JLD (2013) Biased random-key genetic algorithm for non-linearly constrained global optimization. In: Proceedings of the 2013 IEEE congress on evolutionary computation (CEC), Cancun, pp 2201–2206
59. Spears WM, DeJong KA (1991) On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms, San Mateo, pp 230–236
60. Stefanello F, Buriol LS, Hirsch MJ, Pardalos PM, Querido T, Resende MGC, Ritt M (2013) On the minimization of traffic congestion in road networks with tolls. Technical report, AT&T Labs Research, Florham Park
61. Tangpattanakul P, Jozefowiez N, Lopez P (2012) Multi-objective optimization for selecting and scheduling observations by agile earth observing satellites. In: Parallel problem solving from nature – PPSN XII. Lecture notes in computer science, vol 7492. Springer, Berlin/New York, pp 112–121
62. Toso RF, Resende MGC (2015) A C++ application programming interface for biased random key genetic algorithms. Optim Methods Softw 30(1):81–93
63. Valente JMS, Gonçalves JF (2008) A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. Comput Oper Res 35:3696–3713
64. Valente JMS, Gonçalves JF, Alves RAFS (2006) A hybrid genetic algorithm for the early/tardy scheduling problem. Asia-Pac J Oper Res 23:393–405

Scatter Search

24

Rafael Martí, Ángel Corberán, and Juanjo Peiró

Contents

Introduction . . . 718
Past and Present . . . 719
The Five Main Components . . . 721
The Uncapacitated r-Allocation p-Hub Median Problem . . . 725
A Small Example with Real Data . . . 727
Scatter Search for the p-Hub Problem . . . 728
The Diversification Generator Method . . . 729
Reference Set Initialization . . . 731
The Subset Generation Method . . . 731
The Solution Combination Method . . . 731
The Reference Set Update Method . . . 732
The Improvement Method . . . 732
Computational Experiments . . . 733
Parameter Calibration . . . 734
Comparison with a GRASP Algorithm . . . 736
Conclusions . . . 737
Cross-References . . . 737
References . . . 737

Abstract

Scatter search (SS) is a population-based metaheuristic that has been shown to yield high-quality outcomes for hard combinatorial optimization problems.

R. Martí (✉)
Statistics and Operations Research Department, University of Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

Á. Corberán · J. Peiró
Facultat de Ciències Matemàtiques, Departament d'Estadística i Investigació Operativa, Universitat de València, València, Spain
e-mail: angel.corberan@uv.es; juanjo.peiro@uv.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_20

It uses strategies for combining solution vectors, making limited use of randomization, that have proved effective in a variety of problem settings. The fundamental concepts and principles were first proposed in the 1960s and 1970s as an extension of mathematical relaxation techniques for combinatorial optimization problems. Its framework is flexible, allowing the development of implementations with varying degrees of sophistication.

This chapter provides a grounding in the scatter search methodology that will allow readers to create successful applications of their own. To illustrate this, we present a scatter search implementation for a $\mathcal{NP}$-hard variant of the classic p-hub median problem, for which we describe search elements, mechanisms, and strategies to generate, combine, and improve solutions.

Keywords
Scatter search · Metaheuristic · Combinatorial optimization · p-hub

Introduction

Scatter search (SS) was conceived as an extension of a heuristic in the area of mathematical relaxation, which was designed for the solution of integer programming problems: surrogate constraint relaxation. The following three operations come from the area of mathematical relaxation, and they are the core of most evolutionary optimization methods including SS and genetic algorithms (GAs):

- Building, maintaining, and working with a population of elements (coded as vectors).
- Creating new elements by combining existing elements.
- Determining which elements are retained based on a measure of quality.

Two of the best-known mathematical relaxation procedures are Lagrangean relaxation [9] and surrogate constraint relaxation [12]. While Lagrangean approaches absorb "difficult" constraints into the objective function by creating linear combinations of them, surrogate constraint relaxations generate new constraints to replace those considered problematic. The generation of surrogate constraints also involves the combination of existing constraints using a vector of weights. In both cases, these relaxation procedures search for the best combination in an iterative manner.

Scatter search is more intimately related to surrogate relaxation procedures, because not only surrogate relaxation includes the three operations outlined above but also has the goal of generating information from the application of these operations. In the case of surrogate relaxation, the goal is to generate information that cannot be extracted from the original constraints. Scatter search takes on the same approach, by generating information through the combination of two or more solutions. It strategically explores the solution space of an optimization problem by evolving a set of *reference* points. These points define a set, known as *reference set* (*RefSet*), which consists of good solutions obtained by prior solving efforts.

SS and GAs were both introduced in the seventies. While Holland [20] introduced genetic algorithms and the notion of imitating nature and the "survival of the fittest" paradigm, Glover [13] introduced scatter search as a heuristic for integer programming that expanded on the concept of surrogate constraints as mentioned above. Both methods fall in the category of evolutionary optimization procedures, since they build, maintain, and evolve a set (population) of solutions throughout the search. Although the population-based approach makes SS and GAs part of the so-called evolutionary methods, there are fundamental differences between the two methodologies. Typical differences between the scatter search methodology and other evolutionary methods rely on the use of randomization and the size of the population of solutions. Specifically, SS mainly implements deterministic strategies to generate, combine, and improve solutions, while other evolutionary methods, such as GAs, generally introduce random elements in their strategies. On the other hand, in SS, the *RefSet* tends to be small, usually around 10 solutions, while in GAs the population tends to be much larger, usually around 100.

The following principles can be seen as the foundations of the scatter search methodology:

- Useful information about the characteristics of optimal solutions is typically contained in a suitable diverse collection of elite solutions.
- This information is exploited by the combination of the elite solutions.
- The purpose of the combinations is to incorporate to the elite set both diversity (in terms of solutions' attributes) and quality (in terms of objective value).
- Search mechanisms and strategies are not limited to combinations but include solution generation and improvement combination methods.

The scatter search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication. It is based on the idea of limiting the scope of the search to a selective group of combination types, as a mechanism for controlling the number of possible combinations in a given reference set. A comprehensive examination of this methodology can be found in the book of Laguna and Martí in [32]. In the following subsection, we review the most relevant recent applications to illustrate the impact of this methodology.

Past and Present

As mentioned before, scatter search was first introduced in 1977 [13] as a heuristic for integer programming. However, it seems that it was never applied or studied again until 1990, when it was presented at the EPFL Seminar on Operations Research and Artificial Intelligence Search Methods (Lausanne, Switzerland). An article based on this presentation was published in 1994 [14], in which the range of applications was expanded to nonlinear (continuous) optimization problems, binary and permutation problems. The algorithmic principles and the structure of the method were finally proposed in the so-called scatter search template [15]. In a way,

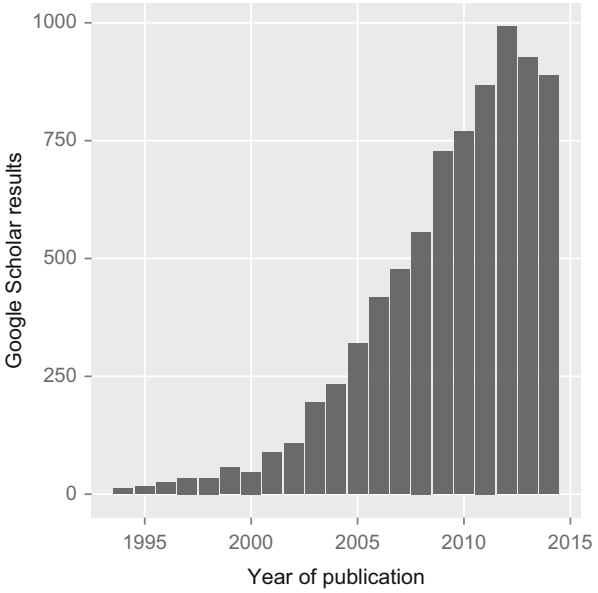


Fig. 1 Publication counts for the query "scatter search" on Google Scholar

this template was a simplification of the previous descriptions and served as the main reference for most of the scatter search implementations up to date. There are two important milestones in the scatter search publications since then. In 2002, a book on this methodology was published by Kluwer [32], which included implementations in C to help the reader to create his/her own applications. In 2006, a special issue of the *European Journal of Operational Research* [1] was devoted to successful scatter search applications.

The scatter search methodology has become the method of choice for the design of solution procedures for many $\mathcal{NP}$-hard combinatorial optimization problems. Its use has been steadily increasing as shown in Figs. 1 and 2. Those figures show the number of yearly publications from 1994 to 2014 on scatter search. Figure 1 shows the number of cites or references for the query "scatter search" on Google Scholar (http://scholar.google.com), while Fig. 2 shows the counts for the same query on the Science Citation Index database of Thomson Reuter's Web of Science, which corresponds to the publications in this topic (http://thomsonreuters.com/thomson-reuters-web-of-science/). To restrict our search, we looked up on the Scopus website (http://www.scopus.com) the number of publications with the query "scatter search" in the title and found that it was 298. We have shortlisted the most recent ones (from 2012 to 2014) that refer to journal papers and deal with practical applications solved with scatter search. Table 1 summarizes the most relevant articles classified into six categories, according to the class of problem being solved. In particular we consider classical problems such as TSP, VRP, and knapsack; scheduling and sequencing problems such as flowshop, jobshop, and project scheduling; manufac-

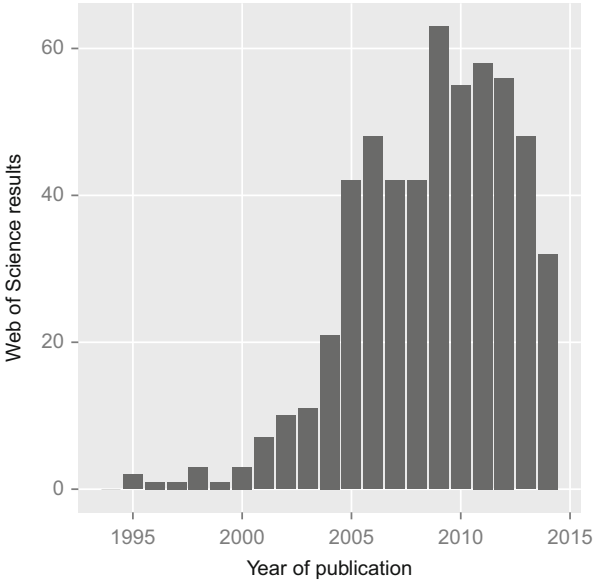


Fig. 2 Publication counts for the query "scatter search" on the Web Of Science

turing problems ranging from NoC mapping to coal and steel production; planning problems such as territory design and location problems; statistics and finance problems such as credit scoring and protein information prediction; and image registration problems such as image registration and segmentation and craniofacial superimposition.

Although we can consider that the scatter search methodology is nowadays well established, and that it has been used in many applications, if we compare SS with other metaheuristics, we can see that there is still significant room for improvement. In particular, a Scopus search with the query "genetic algorithm" returned almost 40,000 entries, while "tabu search" gave 2,273 hits. Consequently, one of this chapter's goals is to trigger the interest of researchers and practitioners to apply the scatter search methodology.

The Five Main Components

In scatter search, the search process is performed to capture information not contained separately in the original solutions. It takes advantage of auxiliary heuristic methods for selecting the elements to be combined and generating new solutions. The combination strategy is devised with the belief that this information can be better exploited when integrated rather than treated in isolation. In general, the decision rules created from such combination strategies produce better empirical outcomes than the standard applications of local decision rules.

Table 1 Journal papers published between 2012 and 2014

Classical problems	
TSP	[46]
VRP	[2, 54, 59, 61]
Linear ordering	[22]
Knapsack	[36]
4-color mapping	[51]
Cutwidth minimization	[47]
Global optimization and black-box	[23, 31]
Scheduling and sequencing	
Flowshop	[19, 44]
Assembly lines	[39]
Jobshop	[7]
Task scheduling	[27, 53]
Project scheduling	[55]
Manufacturing	
NoC mapping	[33, 34]
Aircraft conflict resolution	[37, 56]
Cellular manufacturing systems	[25]
Grid resources selection	[4]
Coal production planning	[49]
Steel industry	[41]
Disassembly sequence	[17]
SONET problems	[3]
Environmental–economic dispatch	[6]
Planning	
Layout	[26, 29, 30]
Transmission expansion planning	[18, 43]
Commercial territory design	[52]
Location	[35, 42]
Statistics and finance	
Credit scoring	[58]
Distribution fitting	[21]
Parameter determination	[38]
Protein information prediction	[28]
Image registration	
Intensity-based image registration	[57]
Image segmentation	[5]
Craniofacial superimposition	[24]

As mentioned, scatter search operates on a set of solutions, *RefSet*, combining them to create new ones. The method basically performs iterations over the *RefSet* and can be summarized in three steps: select the solutions, combine them, and update the *RefSet* with the resulting solutions. As in many other metaheuristics, an

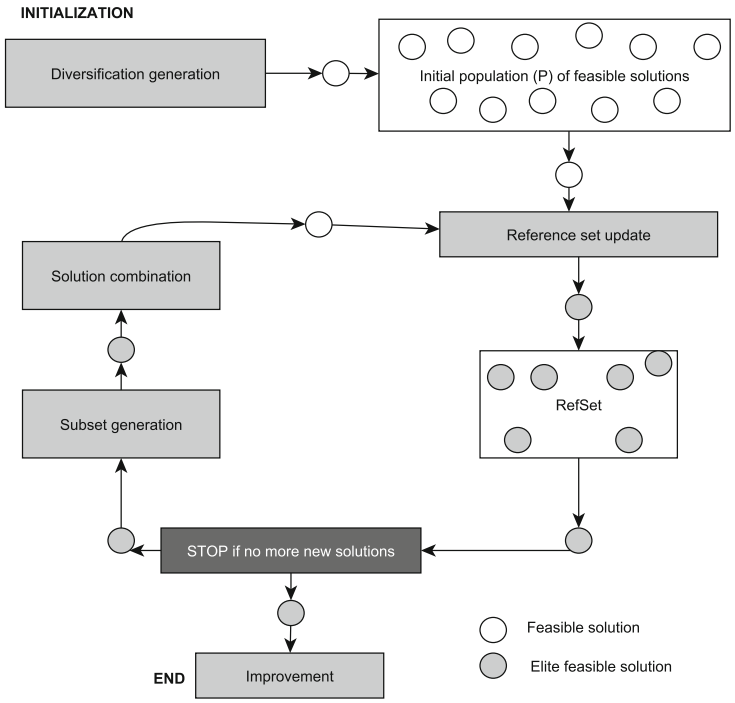


Fig. 3 Selective scatter search scheme

improvement method, or local search, can also be applied to improve the generated solutions. Here, the outcome of the combination method can be sent to a local search procedure to obtain a local optima.

Although the scatter search origins are linked with the mathematical programming area, in which convex combinations are very popular, convex and non-convex combinations are used in this method to generate new solutions. The non-convex combinations project new "centers" into regions of the solution space that are external to the original *RefSet* solutions, a strategy that is based on the principle of input diversity in the search, to avoid getting trapped in local optima.

The scatter search template provides a concise and direct description about how to implement the methodology. Figure 3 shows the scheme of our selective scatter search in which the improvement method is only applied at the end of the search. The methodology basically consists of five elements or methods and their associated strategies and may be sketched as follows:

- A diversification generation method (DGM):
  The DGM can be seen as the mechanism that creates the initial set of solutions P of an optimization problem. This is what we would call "a first generation" in the evolutionary terminology. If the attributes of this first generation are good, there is some confidence of getting better solutions in the following

iterations after strategically combining these initial solutions. The DGM typically implements frequency-based memory to generate P and employs controlled randomization. As opposed to the GAs, which usually rely in randomization to generate the initial population, the DGM samples the solution space in a systematic fashion. The size of P, $|P|$, is denoted as $Psize$.

- An improvement method (IM):

  The IM transforms a given solution of the problem, s, into an enhanced solution s'. Usually, both solutions are expected to be feasible, but this is not a requirement of the methodology. The IM generally relies on local search (LS) procedures (also known as *iterative improvement* procedures). As it is well known, given s, the local search tries to improve it by making "small changes" in its structure. These changes typically consist of adding elements to s, removing elements from s, changing the way in which elements are grouped in s, or changing their order, to mention a few. If an improvement is achieved in this way, then a new solution s' is obtained. This process of *small changes* is continued until no further improvement can be obtained or a time bound is elapsed. If no improvement of s was found, s would be the output.
- A reference set update method (RSUM):

  This method builds *RefSet* for the first time using a reference set creation subroutine (RSCsr) and updates *RefSet* during the search process. In the RSCsr, a given solution from P enters *RefSet* according to its quality or diversity. Many of the implementations of SS indicate that a good trade-off is to build 50% of *RefSet* by quality criterion, and the remaining 50% by a diversity criterion, but these proportions can be modified depending on the problem being solved.

  A standard mechanism to build the initial *RefSet*, whose size will be denoted by b, follows. The construction starts with the selection of the best $b/2$ solutions from P. For each solution in $P \setminus RefSet$, the minimum of the distances to the solutions in *RefSet* is computed. Then, the solution that maximizes the minimal distances is selected. This solution is added to *RefSet* and deleted from P, and the minimal distances are updated. This process is repeated $b/2$ times. The resulting reference set has $b/2$ high-quality solutions and $b/2$ diverse solutions.

  The update operation consists of maintaining a record of the b best solutions found, where the value of b is treated as a constant search parameter.
- A subset generation method (SGM):

  The SGM creates subsets of two or more solutions belonging to *RefSet*. These subsets will be subjected to the next SS process, the combination method. The general SS framework considers the generation of subsets with two, three, and four solutions but only generates a given subset if its solutions are being used to create this subset for the first time. This situation differs from those considered in the context of genetic algorithms, where the combinations are typically determined by the spin of a roulette wheel and are usually restricted to the combination of only two solutions.
- A solution combination method (SCM):

  The SCM creates new solutions from any subset of solutions previously created by the SGM. Although the process of creating new solutions that derive

from other solutions can be achieved by a black-box procedure, it is generally more effective to design the SCM as a problem-specific mechanism, because it is directly related to the way we represent a solution in a problem. Depending on the specific form of the SCM, each subset can create one or more new solutions. This creation can be done in different ways, but the most common ones are systematic combinations of decision rules, path-relinking, and randomization. We refer the reader to [16, 50] for some combination methods within SS.

In this chapter, we illustrate the SS methodology by implementing its underlying framework to solve the uncapacitated r-allocation p-hub median problem (UrApHMP). Although we target here a specific variant of the well-known p-hub median problem, our results show that the scatter search methodology is well suited for solving this type of location problems. In line with our comment above, i.e., to encourage the reader to apply this methodology, here we can find a large family of problems in which the application of SS can result in a promising research area.

The Uncapacitated *r*-Allocation *p*-Hub Median Problem

Let $G = (V, E)$ be a network with set of nodes V and set of edges E. In the uncapacitated r-allocation p-hub median problem (UrApHMP), for each pair of nodes i and $j \in V$, there is an amount of traffic t_{ij} (generally of people or goods) that needs to be transported using this network. The cost of sending a unit of traffic from i to j is denoted by c_{ij}. It is assumed that direct transportation between i and j is not possible. This well-known assumption is based on the empirical evidence that it is not physically and/or economically viable, for example, to schedule a flight between any two pairs of cities or to add a link between any two computers for data transmission, since this would require a large amount of resources. Therefore, the traffic t_{ij} is routed along a path $i \to k \to l \to j$, where nodes k and $l \in V$ are used as intermediate points for this transportation. Examples of these intermediate nodes are some connecting airports such as JFK in New York, ORD in Chicago, and LHR in London; they all serve as transit points where passengers can connect from one flight to another in order to reach their final destinations. The UrApHMP consists of choosing a set H of nodes, ($H \subseteq V$, $|H| = p$), assigning r nodes in H to each node, and minimizing the total transportation cost of all the traffics of the network. The nodes in H, which can be used as intermediate transfer points between any pair of nodes in G, are commonly called *distribution centers* or *hub nodes*. The other nodes in the network are known as *terminal nodes*. For the sake of simplicity, we call them *hubs* and *terminals*, respectively.

Three optimization subproblems arise when solving the UrApHMP: a *location* problem to choose the best locations for the hubs, an *assignment* problem of each terminal to r of the hubs, and a *routing* problem to obtain the minimum cost route to use for transporting the traffics between any given pair of nodes. Regarding the allocation strategy in the assignment process, the UrApHMP generalizes two extensively studied versions of the p-hub location problem: the *single* and *multiple*

versions. In the single version ($r = 1$), each terminal is assigned to only one of the p hubs, thus allowing to send and receive the traffics through this hub. In contrast, in the multiple version ($r = p$), each terminal can send and receive traffics through any of the p hubs. In the version considered here, the *r-allocation*, each terminal is allowed to be allocated to r of the p hubs. The motivation of this version comes from the fact that the single allocation version is too restricted for real-world situations, while the multiple allocation version results in high fixed costs and complicated networks, which does not reflect the real models either. Yaman presented in [60] a study of allocation strategies and introduced the r-allocation version of the problem. The uncapacitated r-allocation p-hub median problem is formulated in [60] in terms of the following variables: Given a node $k \in V$, $z_{kk} = 1$ if node k is set to be a hub (i.e., if a hub is located at this node), and $z_{kk} = 0$ otherwise. Given a non-hub node $i \in V$, $z_{ik} = 1$ if node i is assigned to node k and 0 otherwise. Finally, f_{ijkl} is the proportion of the traffic t_{ij} from node i to node j that travels along the path $i \to k \to l \to j$, where k and l are the nodes that will be used as hubs. With these variables, the problem is formulated as follows:

$$\text{Min} \quad \sum_{i\in V}\sum_{j\in V}\sum_{k\in V}\sum_{l\in V} t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}) f_{ijkl} \tag{1}$$

$$\sum_{k\in V} z_{ik} \le r, \qquad \forall i \in V \tag{2}$$

$$z_{ik} \le z_{kk}, \qquad \forall i,k \in V \tag{3}$$

$$\sum_{k\in V} z_{kk} = p \tag{4}$$

$$\sum_{k\in V}\sum_{l\in V} f_{ijkl} = 1, \qquad \forall i,j \in V \tag{5}$$

$$\sum_{l\in V} f_{ijkl} \le z_{ik}, \qquad \forall i,j,k \in V \tag{6}$$

$$\sum_{k\in V} f_{ijkl} \le z_{jl}, \qquad \forall i,j,l \in V \tag{7}$$

$$f_{ijkl} \ge 0, \qquad \forall i,j,k,l \in V \tag{8}$$

$$z_{ik} \in \{0,1\}, \qquad \forall i,k \in V, \tag{9}$$

where χ, α, and δ are unit rates for collection (origin-hub), transfer (hub-hub) and distribution (hub-destination), respectively. Note that constraints (2) ensure that each node is allocated to at most r hubs, where hubs are selected according to (3). In addition, constraint (4) limits to p the number of hubs. Finally, constraints (5), (6), and (7) are associated with the routing of the traffic between each pair of nodes i, j through their corresponding hubs k, l.

A Small Example with Real Data

With the aim of illustrating the UrApHMP in detail, we introduce a small instance of a network with ten nodes that can be used as the input data. Table 2 contains the traffics t_{ij} and the unitary transportation costs c_{ij} for any pair of nodes (i, j) of this network. The number of nodes to be used as hubs, p, is an input of the problem, as well as r, the number of hubs a given terminal can be assigned to. Assume that $p = 3$ and $r = 2$ and that nodes 3, 6, and 8 will be used as hubs, i.e., $H = \{3, 6, 8\}$. For any pair of nodes, for example, nodes 2 and 5, $t_{2,5} = 18$ is obtained from the table, which means that there are 18 passengers that need to be transported from node 2 to node 5. To compute the cost of transporting $t_{2,5}$, we first need to assign each terminal to r of the p hubs and each hub to all the hubs including itself. For this example, let terminal 2 be assigned to hubs 3 and 6 and terminal 5 to hubs 3 and 8. If we denote by H^i the set of hubs to which node i is assigned to, $H^2 = \{3, 6\}$ and $H^5 = \{3, 8\}$. Then, to transport the corresponding traffics, we need to explore all possible paths in order to know the most inexpensive one. This information is summarized in Table 3, where costs are separated into the three terms described above: c_{ik}, c_{kl}, and c_{lj}. Then, the cost is computed for each path, by applying the unit rate coefficients $\chi = 3$, $\alpha = 0.75$, and $\delta = 2$. As can be seen in this table, path $2 \rightarrow 6 \rightarrow 3 \rightarrow 5$ is the one with the minimum transportation cost (73.25).

Table 2 Example of a traffic and cost matrices derived from an AP instance

	i, j	1	2	3	4	5	6	7	8	9	10
t_{ij}	1	75	37	55	19	20	18	57	17	19	16
	2	26	38	25	27	18	23	38	20	12	17
	3	67	39	51	22	24	21	71	20	23	19
	4	17	33	19	25	16	23	40	23	11	19
	5	17	25	21	19	23	18	73	20	24	19
	6	12	18	12	16	11	17	37	22	10	18
	7	82	78	90	68	95	73	312	99	110	110
	8	35	42	36	48	36	58	173	90	41	92
	9	12	12	15	10	19	11	68	15	24	20
	10	22	25	23	28	23	33	109	51	30	63
c_{ij}	1	0	20	16	23	23	30	32	33	36	36
	2	20	0	20	13	25	15	30	25	38	31
	3	16	20	0	14	7	19	16	18	21	21
	4	23	13	14	0	14	7	18	13	27	18
	5	23	25	7	14	0	16	9	13	14	14
	6	30	15	19	7	16	0	16	8	25	13
	7	32	30	16	18	9	16	0	9	10	7
	8	33	25	18	13	13	8	9	0	18	5
	9	36	38	21	27	14	25	10	18	0	14
	10	36	31	21	18	14	13	7	5	14	0

Table 3 Possible paths and their associated costs between terminals 2 and 5

Path	Collect	Transfer	Dist.	Cost computation
$2 \rightarrow 3 \rightarrow 8 \rightarrow 5$	20	18	13	$(3 \times 20) + (0.75 \times 18) + (2 \times 13) = 99.50$
$2 \rightarrow 3 \rightarrow 3 \rightarrow 5$	20	0	7	$(3 \times 20) + (0.75 \times 0) + (2 \times 7) = 74.00$
$2 \rightarrow 6 \rightarrow 3 \rightarrow 5$	15	19	7	$(3 \times 15) + (0.75 \times 19) + (2 \times 7) = 73.25$
$2 \rightarrow 6 \rightarrow 8 \rightarrow 5$	15	8	13	$(3 \times 15) + (0.75 \times 8) + (2 \times 13) = 77.00$

It is therefore the selected path to route the traffic $t_{2,5}$. Note that there is a path $(2 \rightarrow 3 \rightarrow 3 \rightarrow 5)$ in which both terminals share a common hub, but it is not the most inexpensive one.

The process of checking the paths between any two pair of nodes $i, j \in V$ needs to be computed for any possible assignments and for any possible set of hubs, in order to find the combination minimizing the total cost of transporting all the traffics in the network. This massive calculation gives an idea of the combinatorial nature of the problem. Even when the set of hubs is given, the subproblem of assignment of terminals to hubs is also $\mathcal{NP}$-hard [40].

To illustrate how large is the search space that we are exploring in this problem, consider a medium size instance in which $n = 100$, $p = 5$, and $r = 3$. The location problem, in which we have to select the 5 hubs out of the 100 nodes, gives us 75,287,520 combinations. Then, for each combination we have to solve the assignment problem, in which we have to assign each node to 3 of the 5 hubs selected, giving 10 combinations for each node, which makes 950 possibilities. Finally, for each combination, and each assignment, we have to solve the routing problem. To do that, we have to consider that each node is assigned to 3 hubs, and, therefore, for each pair of nodes, we have 9 routes, which makes a total of 89,100 possible routes in this network with 100 nodes. The total number of combinations, or solutions in the search space, is therefore $75{,}287{,}520 \times 950 \times 89{,}100 = 6.3 \times 10^{15}$. This example also illustrates that the main source of difficulty comes from the first problem, the location, which contributes with the largest factor in this computation.

Scatter Search for the *p*-Hub Problem

As described in section "The Diversification Generator Method", and shown in Fig. 3, the SS method starts by generating a set P of diverse solutions. Then, b of them are selected to create *RefSet*. In our implementation for the UrApHMP, we follow the standard design of selecting the best $\frac{b}{2}$ solutions in P and then the most diverse $\frac{b}{2}$ solutions with respect to the solutions already in *RefSet* (see section "Reference Set Initialization"). The main loop of the method consists of applying the combination method to all pairs of solutions in *RefSet*, and update this set with the new solutions obtained from these combinations. The method finishes when no new solutions are added to *RefSet*.

The Diversification Generator Method

Recall that three optimization subproblems arise when solving the UrApHMP. We base our solution representation in these three subproblems: location, assignment, and routing. Specifically, a solution s is coded as:

- An array H containing the set of nodes to be used as hubs.
- For any node i of the network, an array $H^i \subseteq H$ containing the hubs to which node i is assigned to.
- For any given pair of nodes (i, j), the pair of hubs that have been used to route t_{ij}.

With this representation, it is easy to see that a straightforward approach to construct a feasible solution for the problem is to select first the nodes to be used as hubs, to assign then each terminal to r of the p hubs, and, finally, to route the traffics through the network. Following this rationale, we create a population P of feasible solutions. To do this, we have developed and tested three constructive methods. Two of them are based on a greedy randomized construction [10,11] based on different expressions for the evaluation of the candidate nodes to be hubs. The third one is simply a random construction to provide diversity to P. Each hub of a solution is selected by evaluating all nodes with respect to a greedy function g that measures their attractiveness to be hub. Only the q elements with best g values are placed in a restricted candidate list (RCL), where q is a search parameter. Lower q values favor greedy selection. Then, an element in the RCL is randomly selected, according to a uniform distribution, to become part of the solution. The values of g are updated at each iteration to reflect the changes brought on by the selection of previous elements.

Let $h \in V$ be a candidate node to be a hub. In that case, it would be used for the transportation of the traffics among some terminals, let say the $\lfloor \frac{n}{p} \rfloor$ terminals $(i_1, \ldots, i_{\lfloor \frac{n}{p} \rfloor})$ with the lowest assignment cost to h. We then compute $g(h)$ as

$$g(h) = \sum_{s=1}^{s=\lfloor \frac{n}{p} \rfloor} \text{cost}(i_s, h), \quad \forall h \in V,$$

where $\text{cost}(i, h)$ represents the assignment cost of terminal i to hub h, and can be computed in two ways, leading to constructive methods DGM1 and DGM2.

DGM1 In this method, $\text{cost}(i, h)$ is computed as the cost of sending and receiving the traffics of i through h, i.e., $\text{cost}(i, h) = c_{ih}\overrightarrow{T_i} + c_{hi}\overleftarrow{T_i}$, where $\overrightarrow{T_i} = \sum_{j \in V} t_{ij}$ is the sum of all the traffics from i to all nodes j and $\overleftarrow{T_i} = \sum_{j \in V} t_{ji}$ is the sum of all the traffics from all nodes j to node i.

DGM2 In this method, $\text{cost}(i,h)$ is computed considering the discounting factors that are usually present in the UrApHMP. To do this, we modify the cost expression to $\text{cost}(i,h) = \chi c_{ih}\overrightarrow{T_i} + \dfrac{\alpha+\delta}{2} c_{hi}\overleftarrow{T_i}$.

DGM3 The third method to generate solutions is based on the notion of constructing solutions by simply generating random sets of p hubs. Its purpose is to create a limited amount of solutions not guided by an evaluation function to just bring diversity into P.

Once the p hubs are selected, the next step is to allocate r of the p hubs to each terminal. For any terminal i, we compute the following estimation of the assignment cost of i to a hub h as

$$\text{assignment}(i,h) = c_{ih}\overrightarrow{T_i} + \sum_{j\in V} c_{hj}t_{ij}.$$

Then, we assign i to the hub h_a with the lowest assignment cost, and the above expression is updated to reflect previous assignments:

$$\text{assignment}(i,h) = c_{ih}\overrightarrow{T_i} + \sum_{j\in V\setminus H^i} c_{hj}t_{ij} - \sum_{u\in H^i} c_{iu}t_{iu}, \quad \forall h \in H \setminus H^i.$$

This process is performed in a greedy way, selecting at each iteration the lowest assignment cost. Note that the assignment expressions are estimations, because they assume that there is only one hub h in the path between any pair of nodes i and j, which is not necessarily true.

Finally, we route all the traffics at their minimum cost. For each pair of nodes i and j, we have to determine the hubs $k \in H^i$ and $l \in H^j$ minimizing the routing cost of the traffic sent from i to j. In mathematical terms, given $k \in H^i$ and $l \in H^j$, we denote as $\text{routecost}_{ij}(k,l)$ the cost of transporting the traffics from i to j through hubs k and l, i.e.,

$$\text{routecost}_{ij}(k,l) = t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}).$$

The routing cost from i to j, routecost_{ij}, is then obtained by searching the hubs $k \in H^i$ and $l \in H^j$ minimizing the expression above, i.e.,

$$\text{routecost}_{ij} = \min_{k\in H^i, l\in H^j} \text{routecost}_{ij}(k,l).$$

Note that the time complexity of evaluating the routing costs if $\chi \neq \delta$ is $\mathcal{O}(n^2r^2)$, since the path from j to i does not necessarily involve the same hubs, k and l, than the path $i \to k \to l \to j$.

Once the p hubs have been located, r hubs have been assigned to each node, and all the traffics have been routed, we have a feasible solution s for the UrApHMP. It

is denoted as $s = (H, A)$, and its cost by $f(s)$, where $H = \{h_1, \ldots, h_p\} \subseteq V$ is the set of hubs in the solution and A is the matrix whose rows contain the r hubs assigned to each node.

Reference Set Initialization

As mentioned above, to create the reference set, *RefSet*, the notion of *best* solutions is not limited to their quality, as measured by the objective function, since it also considers their diversity. In particular, our method first chooses $\frac{b}{2}$ solutions attending to their quality. We simply order the solutions in P by their costs, and introduce them, one by one, in *RefSet*, if there is no other solution already in *RefSet* with the same cost. We stop this selection process after examining the 50% of the solutions in P, even if those selected are less than $\frac{b}{2}$ solutions. The rest of the solutions of the *RefSet* are then selected from P by a diversity criterion as follows.

Given $s \notin RefSet$ and $t \in RefSet$, let $C = \{h : h \in H_s \cap H_t\}$ be the set of common hubs in solutions s and t. We define $d_H(s,t) = p - |C|$ as the number of hubs in s not present in t (and vice versa). Note that the lower the value of $d_H(s,t)$ is, the closer s and t are. To select the solution $s \in P$ to be included in *RefSet*, we define $\text{dist}(s, RefSet) = \min_{t \in RefSet} d_H(s,t)$. This distance is computed for all solutions in P, and the solution s^* with maximum value of $\text{dist}(s, RefSet)$ is introduced in *RefSet*.

The Subset Generation Method

The SGM we propose works by generating subsets defined by any two different solutions in *RefSet*. To avoid the repetition of previously generated subsets in previous iterations, each subset is generated only if at least one of its solutions was introduced in *RefSet* in the preceding iteration. Suppose that m subsets were not considered for this reason, then the number of resulting subsets at each iteration for which the combination method will be applied is $\frac{b^2-b}{2} - m$.

The Solution Combination Method

Other methodologies, such as genetic algorithms, base their designs on "generic" combination procedures, also called context independent or black-box operators. On the contrary, scatter search is based on specific combination methods that exploit the characteristics of the problem being solved. The solutions obtained from the combinations in each subset are stored in a set called $Pool$.

Let $U = \{h : h \in H_s \cup H_t\}$ and $I = \{h : h \in H_s \cap H_t\}$. We propose two combination methods for a pair (s,t) of solutions in a subset. They are described in what follows:

- **Method 1**: This method is applied when $|U| > p$. It creates a solution with the hubs in U. It can be considered as a "convex combination" of s and t. In particular, it selects the p elements in U with best evaluation to be hubs, where candidates are evaluated with the same g function introduced in the diversification generation method.
- **Method 2**: This method is applied when $|I| < p$. It creates a solution by including all the elements in I as hubs and selects the rest $p - |I|$ hubs from $V \setminus I$. As in Method 1, candidate hubs are evaluated with g and the best ones are selected in a greedy fashion. The output can be considered as a "non-convex combination" of solutions s and t.

The Reference Set Update Method

The *RefSet* update operation consists of maintaining a record of the b best solutions found so far by the procedure. The issues related to this updating function are straightforward: All the solutions in *RefSet* that are worse than those in the current *Pool* will be replaced by these ones, with the aim of keeping in *RefSet* the b best and most different solutions found so far. Let $s \in Pool$ and $w \in RefSet$ such that the objective function value of s is less than that of w. In this case, s will replace w if s is different from all other solutions in *RefSet*. Note that the update method focuses on quality. In other words, although *RefSet* was created considering both quality and diversity, when updating it, we only consider the objective function value of the candidate solutions obtained from the combination method. In particular, once all the combinations have been performed and the new solutions have been moved to $Pool$, the new *RefSet* simply contains the best b solutions in the set formed with the previous $RefSet \cup Pool$. This strategy, based on quality, favors the convergence of the method.

The Improvement Method

As mentioned in section "The Uncapacitated r-Allocation p-Hub Median Problem," three optimization subproblems arise when solving the UrApHMP. Since we solve the routing subproblem optimally, we propose two improvement procedures based on local search strategies for the other two subproblems: LS_H for the hub selection and LS_A for the terminal allocations. Both are based on the local search procedures proposed in [48].

LS_H implements a classical exchange procedure in which a hub h_i is removed from H and a non-hub $h'_i \in N \setminus H$ replaces h_i, thus obtaining $H' = \{h_1, h_2, \ldots, h'_i, \ldots, h_p\}$. When hub h_i is replaced with h'_i, we have to reevaluate the hub assignment for the vertices assigned to h_i. Moreover, the routes for the traffics are not necessarily optimal for the new set of hubs H', and, hence, they have to be recomputed. The local search procedure LS_H performs moves as long as the

cost improves. We have implemented here the so-called *first strategy*, in which the first improving move in the neighborhood is performed.

The local search procedure LS_A is similar to LS_H, but it considers changes in the assignment of terminals to hubs. In particular, for a node i with $H^i = \{h_{i1}, \ldots, h_{ia}, \ldots, h_{ir}\}$, this procedure exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace h_{ia} with $\bar{h}_{ia} \in H \backslash H^i$, thus obtaining $\bar{H}^i = \{h_{i1}, \ldots, \bar{h}_{ia}, \ldots, h_{ir}\}$. As in LS_H, the method performs moves while improving and returns the local optimum reached as its output.

Computational Experiments

This section describes the computational experiments performed to first find the parameter values of the scatter search method, and then study its performance, especially with respect to previous approaches. The procedure has been implemented in C using GCC 4.8.2, and the results reported in this section have been obtained with an Intel Core i7–3770 at 3.40 GHz and 16 GB of RAM under Ubuntu 14.04 GNU/Linux – 64-bit operating system. The metrics we use to measure the performance of the algorithms are:

- Dev: Average percentage deviation with respect to the best solution found (or from the optimal solution, if available).
- # Best: Number of best solutions found.
- CPU: Average computing time in seconds.

We have tested our algorithms on the three sets of instances previously reported [8, 45, 48]. The **CAB** (Civil Aviation Board) data set, based on airline passenger flows among some important cities in the United States, consists of 23 instances with 25 nodes and $p = \{1, \ldots, 5\}$ and $r = \{1, \ldots, p\}$. The **AP** (Australian Post) data set is based on real data from the Australian postal service. We have extended this set by generating, from the original file, 311 instances with $40 \leq n \leq 200$ and $1 \leq p \leq 20$. The third data set, **USA423**, consists of a data file concerning 423 cities in the United States, where real distances and passenger flows for an accumulated 3 months' period are considered. From the original data, 30 instances have been generated with $p \in \{3, 4, 5, 6, 7\}$. The total set of 264 instances has been divided into three subsets: small ($25 \leq n \leq 50$), medium ($55 \leq n \leq 100$), and large ($105 \leq n \leq 200$). The entire set of instances is available at www.optsicom.es.

The experiments are divided into two main blocks. The first block, described in section "Parameter Calibration", is devoted to study the behavior of the components of the solution procedure, as well as to determine the best values for the search parameters. The second block of experiments, reported in section "Comparison with a GRASP Algorithm", compares our procedure with the best published methods.

Parameter Calibration

The first set of experiments to calibrate our method is performed on a subset of 47 instances: five instances from the CAB set with $n = 25$ and 42 instances with $40 \leq n \leq 200$ from the AP set. We refer to these 47 instances as the *training set* and to the remaining instances as the *testing set*.

The Size of *P*

First, we study the value of the parameter used in the diversification generator method, *Psize*, which determines the number of solutions in *P*. We have tested four values: 50, 100, 150, and 200. For each instance, we generate $PSize/3$ solutions with each diversification generation method (DGM1, DGM2, and DGM3, respectively). In order to evaluate DGM methods in isolation (without the combination and improvement methods), this experiment only considers the constructive phase. The results on the training set are shown in Table 4.

As expected (see Table 4), the best solutions in terms of quality are obtained with $Psize = 200$. In this case, the algorithm obtains an average percentage deviation of 1.6% and 20 best-known solutions. The CPU time for $Psize = 200$ is still reasonable, with virtually no difference in small and medium instances. Although for the large instances, the CPU values are slightly larger, we consider that the improvement of the results deserves the CPU effort, so we set $Psize = 200$ in the rest of the experiments.

The Size of *RefSet*

In order to study the size of *RefSet*, b, we include all the elements of scatter search in the algorithm except the local search procedures (to avoid the effects of the improvement methods). Table 5 shows the results for different values of this parameter ($b = 5, 6, \ldots, 10$). As it can be seen, the higher the value of b, the better are the results. Obviously, this implies higher computing times. So, we choose $b = 6$, since this value results in a good trade-off between solution quality and CPU time.

Local Search Methods

We now study the effect of the improvement procedures described in section "The Improvement Method". Recall that two local searches have been proposed, one for

Table 4 Calibration of *Psize*

		Dev (%)				CPU				# best			
Size	# inst	50	100	150	200	50	100	150	200	50	100	150	200
Small	8	3.8	2.5	0.7	1.6	0.01	0.02	0.03	0.03	0	3	5	4
Medium	27	3.8	3.9	2.6	1.4	0.08	0.16	0.24	0.33	1	4	8	14
Large	12	4.2	1.9	2.0	2.2	0.47	0.95	1.45	1.96	3	4	3	2
Summary	47	3.9	3.2	2.1	1.6	0.17	0.34	0.51	0.69	4	11	16	20

Table 5 Calibration of b

		Dev (%)						CPU						# best					
Size	# inst	5	6	7	8	9	10	5	6	7	8	9	10	5	6	7	8	9	10
Small	8	0.1	0.1	0.1	0.0	0.0	0.1	0.04	0.03	0.04	0.05	0.05	0.06	7	7	7	8	8	7
Medium	27	3.1	2.4	2.5	1.5	1.0	0.8	0.31	0.35	0.38	0.42	0.45	0.51	4	6	6	12	12	12
Large	12	1.6	1.7	1.6	0.4	0.3	0.7	1.81	1.97	2.11	2.32	2.57	2.65	3	5	6	9	10	7
Summary	47	2.2	1.8	1.8	1.0	0.7	0.6	0.65	0.71	0.77	0.84	0.92	0.98	14	18	19	29	30	26

Table 6 Comparison of strategies SS^A and SS^B

		Dev (%)		CPU		# best	
Size	# inst	SS^A	SS^B	SS^A	SS^B	SS^A	SS^B
Small	8	0.0	0.6	0.23	0.06	8	4
Medium	27	0.0	0.1	6.32	1.01	27	17
Large	12	0.0	0.1	57.15	8.84	12	7
Summary	47	0.0	0.2	18.26	2.85	47	28

the hub selection (LS_H) and another for the terminal allocations (LS_A). Although the standard SS design specifies to apply the improvement method to all the solutions resulting from the combination method, considering that the local searches for the p-hub problem are quite time consuming, we limit their application to the best solutions across all the global iterations. In particular, we only apply LS_H and LS_A to the solutions in the final *RefSet* (i.e., just before the end of the algorithm). To evaluate the efficiency of this selective strategy, we have studied the following two variants:

SS^A: LS_H and LS_A are applied to all the solutions in the final *RefSet*.
SS^B: LS_H and LS_A are applied only to the best solution in the final *RefSet*.

Table 6 shows the results obtained on the 47 instances of the training set with both methods. The results indicate that applying the improvement methods to all the solutions in *RefSet* substantially improves their value. Variant SS^A exhibits a larger number of best solutions found compared to variant SS^B. Nevertheless, this exhaustive application of the local searches results in much higher CPU times (from 2.85 to 18.26 s in average). Despite the fact that variant SS^B is not able to get some of the best-known solutions, its average deviation is very small (from 0.1% to 0.6%). We keep both variants in the following comparison with a previous method.

To complement the analysis above, we now explore the contribution on the quality of the final solution of the three main methods in SS: construction, combination, and improvement. Figure 4 shows, for six representative instances, the value of the best solution obtained with the diversification generation method (DGM), the value of the best solution obtained with the combination method (REF), and the value after the application of the local search to the solutions in the final

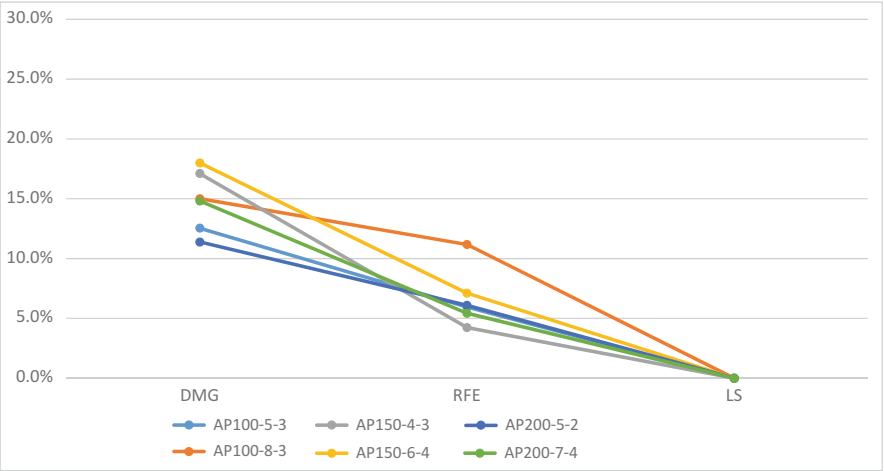


Fig. 4 Search profile for six different instances

Table 7 Comparison between the SS and a GRASP method

		Dev (%)			CPU			# best		
Size	# inst	SS^A	SS^B	GRASP	SS^A	SS^B	GRASP	SS^A	SS^B	GRASP
Medium	116	0.00	0.10	0.28	3.8	0.6	17.1	98	73	18
Large	48	0.00	0.09	0.31	34.3	4.5	31.6	43	21	5
Extra-large	30	0.21	4.42	4.54	679.5	91.4	607.4	19	4	11
Summary	194	0.03	0.77	0.95	115.8	15.6	112.0	160	98	34

RefSet. This clearly illustrates how the three methods contribute to reach the final high-quality solutions.

Comparison with a GRASP Algorithm

After the calibration process described before, we now compare the performance of our SS algorithms with the GRASP proposed in [48], which as far as we know is the best heuristic algorithm for the UrApHMP. All the methods under comparison are run in the same computer on the entire set of 194 instances. The results of both scatter search variants, SS^A and SS^B, and the GRASP are summarized in Table 7. This table clearly shows that our SS procedures outperform the previously proposed GRASP method in terms of quality and number of best solution found. In particular, both SS procedures obtain a larger number of best solutions and a lower average deviation than GRASP. SS^B is very competitive since it is able to obtain very good solutions in short computing times.

Conclusions

In this chapter, we summarize our experiences on the subject of implementing scatter search. We have experimented with this methodology for a number of years, and as a result of implementing different search strategies, and performing extensive experimental testing, we have learned some valuable lessons condensed above. As it is well known, metaheuristic methodologies are based on principles and not necessarily on theory that can be spelled out with theorems and proofs. Hence, to illustrate and prove the efficiency of search mechanisms, strategies, and key elements in scatter search, we have shown how to solve an important and difficult combinatorial optimization problem: the uncapacitated r-allocation p-hub median problem. In particular, we describe in detail how to construct, improve, and combine solutions for this $\mathcal{NP}$-hard problem in an efficient way, i.e., with fast methods that are able to produce high-quality solutions. These research opportunities carry with them an emphasis on producing systematic and strategically designed rules, rather than following the policy of relegating decisions to random choices, as often is fashionable in evolutionary methods.

Cross-References

▸ Evolution Strategies
▸ Evolutionary Algorithms
▸ Genetic Algorithms
▸ Network Optimization

Acknowledgments This work was supported by the Spanish Ministerio de Economia y Competitividad (projects TIN-2012-35632-C02 and MTM-2012-36163-C06-02 and grant BES-2013-064245) and by the Generalitat Valenciana (project Prometeo 2013/049).

References

1. Martí R (ed) (2006) Scatter search methods for optimization. Volume 169 of feature cluster of the European Journal of Operational Research. Elsevier, Amsterdam
2. Alkhazaleh HA, Ayob M, Ahmad Z (2013) Scatter search for solving team orienteering problem. Res J Appl Sci 8(3):181–190
3. Bernardino AM, Bernardino EM, Sánchez-Pérez JM, Gómez-Pulido JA, Vega-Rodríguez MA (2012) Solving sonet problems using a hybrid scatter search algorithm. In: Madani K, Dourado CA, Rosa A, Filipe J (eds) Computational intelligence. Volume 399 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 81–97
4. Botón-Fernández M, Vega-Rodríguez MA, Prieto-Castrillo F (2013) An efficient and self-adaptive model based on scatter search: solving the grid resources selection problem. In: Moreno-Díaz R, Pichler F, Quesada-Arencibia A (eds) Computer aided systems theory –

EUROCAST 2013. Volume 8111 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 396–403
5. Bova N, Ibáñez Ó, Cordón Ó (2013) Image segmentation using extended topological active nets optimized by scatter search. IEEE Comput Intell Mag 8(1):16–32
6. de Athayde Costa e Silva M, Klein CE, Mariani VC, Dos Santos Coelho L (2013) Multiobjective scatter search approach with new combination scheme applied to solve environmental/economic dispatch problem. Energy 53:14–21
7. Engin O, Ylmaz MK, Baysal ME, Sarucanl A (2013) Solving fuzzy job shop scheduling problems with availability constraints using a scatter search method. J Multiple-Valued Log Soft Comput 21(3–4):317–334
8. Ernst AT, Krishnamoorthy M (1996) Efficient algorithms for the uncapacitated single allocation p-hub median problem. Locat Sci 4(3):139–154
9. Everett H (1963) Generalized Lagrangean multiplier method for solving problems of optimal allocation of resources. Oper Res 11:399–417
10. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6(2):109–133
11. Festa P, Resende MGC (2011) Grasp: basic components and enhancements. Telecommun Syst 46(3):253–271
12. Glover F (1965) A multiphase-dual algorithm for the zero-one integer programming problem. Oper Res 13:879–919
13. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(7):156–166
14. Glover F (1994) Tabu search for nonlinear and parametric optimization with links to genetic algorithms. Discret Appl Math 49(1–3):231–255
15. Glover F (1998) A template for scatter search and path relinking. In Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Artificial Evolution. Volume 1363 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp 13–54.
16. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29(3):652–684
17. Guo XW, Liu SX, Wang DZ (2012) Scatter search for solving multi-objective disassembly sequence optimization problems. J Northeast Univ 33(1):56–59
18. Habibi MR, Rashidinejad M, Zeinaddini-Meymand M, Fadainejad R (2014) An efficient scatter search algorithm to solve transmission expansion planning problem using a new load shedding index. Int Trans Electr Energy Syst 24(2):153–165
19. Hariharan R, Golden Renjith Nimal RJ (2014) Solving flow shop scheduling problems using a hybrid genetic scatter search algorithm. Middle East J Sci Res 20(3):328–333
20. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
21. Hu L, Jiang Y, Zhu J, Chen Y (2013) Hybrid of the scatter search, improved adaptive genetic, and expectation maximization algorithms for phase-type distribution fitting. Appl Math Comput 219(10):5495–5515
22. Huacuja HJF, Valdez GC, Rangel RAP, Barbosa JG, Reyes LC, Valadez JMC, Soberanes HJP, Villanueva DT (2012) Scatter search with multiple improvement methods for the linear ordering problem. Malays J Comput Sci 25(2):76–89
23. Hvattum LM, Duarte A, Glover F, Martí R (2013) Designing effective improvement methods for scatter search: An experimental study on global optimization. Soft Comput 17(1):49–62
24. Ibáñez O, Cordón O, Damas S, Santamaría J (2012) An advanced scatter search design for skull-face overlay in craniofacial superimposition. Expert Syst Appl 39(1):1459–1473
25. Jabal-Ameli MS, Moshref-Javadi M (2014) Concurrent cell formation and layout design using scatter search. Int J Adv Manuf Technol 71(1–4):1–22
26. Jabal-Ameli MS, Moshref-Javadi M, Tabrizi BB, Mohammadi M (2013) Cell formation and layout design with alternative routing: a multi-objective scatter search approach. Int J Ind Syst Eng 14(3):269–295

27. Jaradat G, Ayob M, Ahmad Z (2014) On the performance of scatter search for post-enrolment course timetabling problems. J Comb Optim 27(3):417–439
28. Kashefi AH, Meshkin A, Zargoosh M, Zahiri J, Taheri M, Ashtiani S (2013) Scatter-search with support vector machine for prediction of relative solvent accessibility. EXCLI J 12:52–63
29. Kothari R, Ghosh D (2014) A scatter search algorithm for the single row facility layout problem. J Heuristics 20(2):125–142
30. Krishnan M, Karthikeyan T, Chinnusamy TR, Venkatesh Raja K (2012) A novel hybrid metaheuristic scatter search-simulated annealing algorithm for solving flexible manufacturing system layout. Eur J Sci Res 73(1):52–61
31. Laguna M, Gortázar F, Gallego M, Duarte A, Martí R (2014) A black-box scatter search for optimization problems with integer variables. J Glob Optim 58(3):497–516
32. Laguna M, Martí R (2002) Scatter search: methodology and implementations in C. Kluwer Academic, Norwell
33. Le Q, Yang G, Hung W, Guo W (2012) Reliable NoC mapping based on scatter search. In: Liu B, Ma M, Chang J (eds) Information computing and applications. Volume 7473 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 640–647
34. Le Q, Yang G, Hung WNN, Zhang X, Fan F (2014) A multiobjective scatter search algorithm for fault-tolerant noc mapping optimisation. Int J Electron 101(8):1056–1073
35. Li J, Prins C, Chu F (2012) A scatter search for a multi-type transshipment point location problem with multicommodity flow. J Intell Manuf 23(4):1103–1117
36. Li VC, Liang YC, Sun YY, Chen YS (2012) A scatter search method for the multidimensional knapsack problem with generalized upper bound constraints. J Chin Inst Ind Eng 29(8):559–571
37. Li ZZ, Song XY, Sun JZ, Huang ZT (2012) A scatter search methodology for the aircraft conflict resolution problem. In: Huang D, Jiang C, Bevilacqua V, Figueroa JC (eds) Intelligent computing technology. Volume 7389 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 18–24
38. Lin SW, Chen SC (2012) Parameter determination and feature selection for c4.5 algorithm using scatter search approach. Soft Comput 16(1):63–75
39. Liu Q, Wang WX, Zhu KR, Zhang CY, Rao YQ (2014) Advanced scatter search approach and its application in a sequencing problem of mixed-model assembly lines in a case company. Eng Optim 46(11):1485–1500
40. Love RF, Morris JG, Wesolowski GO (1988) Facilities location: models and methods. Elsevier, New York
41. Lv Y, Wang G, Tang L (2014) Scenario-based modeling approach and scatter search algorithm for the stochastic slab allocation problem in steel industry. ISIJ Int 54(6):1324–1333
42. Martí R, Corberán Á, Peiró J (2015) Scatter search for an uncapacitated p-hub median problem. Comput Oper Res 58(0):53–66
43. Meymand MZ, Rashidinejad M, Khorasani H, Rahmani M, Mahmoudabadi A (2012) An implementation of modified scatter search algorithm to transmission expansion planning. Turk J Electr Eng Comput Sci 20(Suppl.1):1206–1219
44. Naderi B, Ruiz R (2014) A scatter search algorithm for the distributed permutation flowshop scheduling problem. Eur J Oper Res 239(2):323–334
45. O'Kelly ME (1987) A quadratic integer program for the location of interacting hub facilities. Eur J Oper Res 32(3):393–404
46. Pantrigo JJ, Duarte A (2013) Low-level hybridization of scatter search and particle filter for dynamic TSP solving. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for dynamic optimization. Volume 433 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 291–308
47. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199(1):285–304
48. Peiró J, Corberán A, Martí R (2014) GRASP for the uncapacitated r-allocation p-hub median problem. Comput Oper Res 43(1):50–60

49. Pendharkar PC (2013) Scatter search based interactive multi-criteria optimization of fuzzy objectives for coal production planning. Eng Appl Artif Intell 26(5–6):1503–1511
50. Resende MGC, Ribeiro CC, Glover F, Martí R (2010) Scatter search and path-relinking: fundamentals, advances, and applications. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science, vol 146. Springer, New York, pp 87–107
51. Sadiq AT, Sagheer AM, Ibrahim MS (2012) Improved scatter search for 4-colour mapping problem. Int J Reasoning-based Intell Syst 4(4):221–226
52. Salazar-Aguilar MA, Ríos-Mercado RZ, González-Velarde JL, Molina J (2012) Multiobjective scatter search for a commercial territory design problem. Ann Oper Res 199(1):343–360
53. Shen Z, Zou H, Sun H (2012) Task scheduling for imaging reconnaissance satellites using multiobjective scatter search algorithm. In: Li Z, Li X, Liu Y, Cai Z (eds) Computational intelligence and intelligent systems. Communications in computer and information science. Springer, Berlin/Heidelberg, pp 240–249
54. Tan Y, Cheng TCE, Ji M (2014) A multi-objective scatter search for the ladle scheduling problem. Int J Prod Res 52(24):7513–7528
55. Tang J, Zhang G, Zhang B, Cen H (2012) Resource-constrained project scheduling using electromagnetism-based scatter search. J Comput Inf Syst 8(12):5219–5227
56. Tavakkoli-Moghaddam R, Ranjbar-Bourani M, Amin GR, Siadat A (2012) A cell formation problem considering machine utilization and alternative process routes by scatter search. J Intell Manuf 23(4):1127–1139
57. Valsecchi A, Damas S, Santamaría J, Marrakchi-Kacem L (2014) Intensity-based image registration using scatter search. Artif Intell Med 60(3):151–163
58. Wang J, Hedar AR, Wang S, Ma J (2012) Rough set and scatter search metaheuristic based feature selection for credit scoring. Expert Syst Appl 39(6):6123–6128
59. Xu Y, Qu R (2012) A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. Appl Intell 36(1):229–241
60. Yaman H (2011) Allocation strategies in hub networks. Eur J Oper Res 211(3):442–451
61. Zhang T, Chaovalitwongse WA, Zhang Y (2012) Scatter search for the stochastic travel-time vehicle routing problem with simultaneous pick-ups and deliveries. Comput Oper Res 39(10):2277–2290

Tabu Search

25

Manuel Laguna

Contents

Introduction . . . 742
Short-Term Memory . . . 742
Example 1 . . . 744
Example 2 . . . 747
Long-Term Memory . . . 748
Example 3 . . . 749
Strategic Oscillation . . . 750
Example 4 . . . 751
Example 5 . . . 753
Path Relinking . . . 755
Example 6 . . . 756
Conclusions . . . 757
Cross-References . . . 757
References . . . 758

Abstract

Tabu search (TS) is a solution methodology within the area of metaheuristics. While the methodology applies to optimization problems in general, most TS applications have been and continue to be in discrete optimization. A key and distinguishing feature of tabu search is the use of special strategies based on adaptive memory. The underlying philosophy is that an effective search for optimal solutions should involve a flexible process that responds to the objective function landscape in a manner that allows it to learn appropriate directions to exploit specific areas of the solution space and useful departures to explore new terrain. The adaptive memory structures of tabu search enable the implementation

M. Laguna (✉)
Leeds School of Business, University of Colorado Boulder, Boulder, CO, USA
e-mail: laguna@colorado.edu

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_24

of procedures that are capable of searching effectively and produce solutions of suitable quality within reasonable computational effort.

Keywords
Heuristic search · Metaheuristics · Optimization

Introduction

Tabu search (TS) has its origins in Fred Glover's seminal 1977 paper *Heuristics for Integer Programming Using Surrogate Constraints*. This article introduced three key concepts: (1) scatter search, (2) oscillating assignment, and (3) strongly determined and consistent variables. Scatter search was later developed into a full-blown metaheuristic that is now well-established in the literature. Once tabu search was fully developed, oscillating assignment became the TS mechanism known as strategic oscillation. The concept of strongly determined and consistent variables became part of the tabu search long-term memory strategies, as described below. Without using the term tabu, the knapsack heuristic in [1] illustrates the use of short-term memory. The tabu search name was coined in [2], where the method was introduced as a "metaheuristic superimposed on another heuristic." It was also the first time that "metaheuristic" was used in the literature.

A tabu search starts from an initial solution and explores the solution space by making transformations to move from one solution to the next. That is, TS belongs to the family of metaheuristic methodologies that are based on single-solution searches. This family includes methodologies such as simulated annealing and variable neighborhood search. In contrast, genetic algorithms and scatter search belong to population-based metaheuristics, which create, maintain, and evolve a population of solutions as a means to explore the solution space. TS is based on the premise that problem-solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration.

The choices to determine how to transform a solution are guided by information collected during the search through adaptive memory structures that minimize the reliance in the sampling techniques that are typically employed by memoryless designs. Adaptive memory contrasts with rigid memory designs characteristic of branch and bound strategies. Responsive exploration integrates the basic principles of intelligent search, i.e., exploiting good solution features while exploring new promising regions. Tabu search is concerned with finding new and more effective ways of taking advantage of the mechanisms associated with both adaptive memory and responsive exploration.

Short-Term Memory

Tabu search can be applied directly to verbal or symbolic statements of many kinds of decision problems, without the need to transform them into mathematical formulations. Nevertheless, it is useful to introduce mathematical notation to express

a broad class of these problems, as a basis for describing certain features of tabu search. We characterize this class of problems as that of optimizing (minimizing or maximizing) a function $f(x)$ subject to $x \in X$, where the set X summarizes constraints on the vector of decision variables x, including those that compel all or some components of x to receive discrete values. The problem of interest may not have an easy mathematical representation, but the verbal stipulations could be coded as rules.

Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each solution $x \in X$ has an associated neighborhood $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a *move*. Hence, x' is said to be a neighbor of x.

In a minimization problem, best improving refers to the strategy of identifying $x^* \in N(x)$ such that $f(x^*) < f(x')$ for all $x' \in N(x)$. The move value is the change in the objective function value from the current solution to the solution after the move, i.e., $v = f(x^*) - f(x)$. Therefore, $v < 0$ for improving moves and $v \geq 0$ for nonimproving moves. In steepest descent, the procedure only makes improving moves, and it stops when no improving move is available in the neighborhood of the current solution. That is, the best neighbor of a solution becomes the new current solution as long as the move reduces the objective function value. In contrast, the best-improving strategy in TS chooses the best neighbor of x^* of x even when $v \geq 0$. In spite of its attractiveness, in certain settings the best-improving strategy is sometimes impractical because it may be computationally too expensive, as when $N(x)$ is large or each element is costly to retrieve or evaluate.

The relevance of choosing good solutions from current neighborhoods is magnified when the guidance mechanisms of tabu search are introduced to go beyond the locally optimal termination point of a descent method. Thus, an important first-level consideration for tabu search is to determine an appropriate *candidate list strategy* for narrowing the examination of elements of $N(x)$ in order to achieve an effective trade-off between the quality of x^* and the effort expended to find it. A popular candidate list strategy is known as the first-improving strategy. This strategy is such that neighbors are scanned in a predetermined order and the search moves to the first $x' \in N(x)$ for which $f(x') < f(x)$. If no such a move exists, then the search moves to the best-nonimproving move in the neighborhood.

Short-term memory functions give the search the opportunity to continue beyond local optima, by allowing the execution of nonimproving moves coupled with the modification of the neighborhood structure of subsequent solutions. The modified neighborhood, denoted by $N^*(x)$, is the result of maintaining a selective history of the states encountered during the search.

The tabu classification serves to identify elements of $N(x)$ that are excluded from $N^*(x)$. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of x is not a static set, but rather a set that can change according to the history of the search. A TS process based on short-term strategies may allow a solution x to be visited more than once, but it is likely that the corresponding reduced neighborhood $N^*(x)$ will be different each time.

From a practical standpoint, the method will characteristically identify an optimal or near-optimal solution long before a substantial portion of X is examined.

The approach of storing complete solutions (*explicit* memory) generally consumes a massive amount of space and time when applied to each solution generated. Therefore, instead of recording full solutions, TS memory structures are based on recording attributes (*attributive* memory). In addition, short-term memory is based on the most recent history of the search trajectory (*recency-based* memory). In recency-based attributive memory, selected attributes that occur in solutions recently visited are labeled *tabu-active* and moves that contain tabu-active elements, or particular combinations of these attributes, are those that are classified tabu. This prevents certain solutions from the recent past from belonging to $N^*(x)$ and hence from being revisited. Other solutions that share such tabu-active attributes are also similarly prevented from being visited.

The number of iterations (i.e., executed moves) that an attribute remains tabu-active is known as the tabu tenure. Tabu tenure can be static, i.e., fixed throughout the search, or dynamic, i.e., vary either systematically or probabilistically. Aspiration criteria are introduced in tabu search to determine when tabu activation rules can be overridden, thus removing a tabu classification otherwise applied to a move. The most common aspiration criterion consists of removing a tabu classification from a trial move when the move yields a solution better than the best obtained so far.

Example 1

The short-term memory structure of tabu search will be illustrated with the following instance of the classical 0-1 knapsack problem:

Maximize $67x_1 + 500x_2 + 98x_3 + 200x_4 + 120x_5 + 312x_6 + 100x_7 + 200x_8 + 180x_9 + 100x_{10}$

Subject to $5x_1 + 45x_2 + 9x_3 + 19x_4 + 12x_5 + 32x_6 + 11x_7 + 23x_8 + 21x_9 + 14x_{10} \leqslant 100$

$$x \in \{0, 1\}$$

The initial solution is found using a greedy heuristic that consists of selecting items (i.e., setting variable values to 1), one by one, in decreasing order of their bang-for-buck ratio. The process ends when adding one more item exceeds the capacity of the knapsack. The bang for buck is the ratio of profit to weight, where profits are the objective function coefficients and weights are the constraint coefficients. For simplicity, the variables in the example are indexed by their bang-for-buck ratio. That is, x_1 is the variable with the highest ratio and x_{10} is the variable with the lowest ratio. The construction of the initial solution is summarized in Table 1.

The move mechanism is defined as changing the value of a single variable from either zero to one or one to zero; this is also known as a "flip" move. The best-

Table 1 Construction of initial solution for Example 1

Item	Profit	Weight	Ratio	Chosen?	Profit	Weight
1	67	5	13.40	1	67	5
2	500	45	11.11	1	500	45
3	98	9	10.89	1	98	9
4	200	19	10.53	1	200	19
5	120	12	10.00	1	120	12
6	312	32	9.75		0	0
7	100	11	9.09		0	0
8	200	23	8.70		0	0
9	180	21	8.57		0	0
10	100	14	7.14		0	0
				Total	985	90

Table 2 Neighborhood of the current solution in Example 1

No.	Move	Neighbor	Profit	Weight
1	$x_1 = 0$	(2, 3, 4, 5)	918	85
2	$x_2 = 0$	(1, 3, 4, 5)	485	45
3	$x_3 = 0$	(1, 2, 4, 5)	887	81
4	$x_4 = 0$	(1, 2, 3, 5)	785	71
5	$x_5 = 0$	(1, 2, 3, 4)	865	78
6	$x_6 = 1$	(1, 2, 3, 4, 5, 6)	1297	122
7	$x_7 = 1$	(1, 2, 3, 4, 5, 7)	1085	101
8	$x_8 = 1$	(1, 2, 3, 4, 5, 8)	1185	113
9	$x_9 = 1$	(1, 2, 3, 4, 5, 9)	1165	111
10	$x_{10} = 1$	(1, 2, 3, 4, 5, 10)	1085	104

improving strategy is used, meaning that all moves are considered and the one that results in the neighbor with the best objective function value is chosen. The neighborhood of the current solution is then given in Table 2.

Since in this example the search is not allowed to visit infeasible solutions, there is no improving move in the neighborhood of the current solution. Under this restriction, the best move is the first in the table and consists of changing the value of x_1 from one to zero. The Neighbor column shows the indexes of the variables that are set to one as a result of the corresponding move. The short-term memory structure may be designed as follows:

Attribute: Index of the variable that was chosen to change values
Tabu classification: A move that includes a tabu-active variable is classified tabu
Tabu tenure: Three iterations

Using this structure, attribute 1 is declared tabu-active and therefore a move that attempts to change the current value of x_1 is classified tabu for three iterations.

Table 3 Ten iterations of the short-term memory procedure

Iteration	Tabu-active	Move	Solution	Profit	Weight
0			(1, 2, 3, 4, 5)	985	90
1		1	(2, 3, 4, 5)	918	85
2	1	7	(2, 3, 4, 5, 7)	1018	96
3	7 1	3	(2, 4, 5, 7)	920	87
4	3 7 1	5	(2, 4, 7)	800	75
5	5 3 7	8	(2, 4, 7, 8)	1000	98
6	8 5 3	7	(2, 4, 8)	900	87
7	7 8 5	3	(2, 3, 4, 8)	998	96
8	3 7 8	4	(2, 3, 8)	798	77
9	4 3 7	9	(2, 3, 8, 9)	978	98
10	9 4 3	8	(2, 3, 9)	778	75

Table 3 shows ten iterations of the procedure using the short-term memory structure defined above.

The tabu-active column shows the indexes of the variables that are tabu-active. Each index appears for three iterations. It starts in the first position, and it moves one position in each iteration until it is removed from the list. The Move column indicates the variable that has been chosen to "flip" its value. The Solution column shows the solution after the move has been executed with the resulting profit and weight shown in the last two columns, respectively. The best solution is found in the second iteration and has a profit of 1018.

This example is based on very simple "flip" moves. The neighborhood defined by the move mechanism is a design choice of critical importance for the performance of the method. Often, the neighborhood is defined by several move types that may vary in complexity. The attributes to be used for the tabu classification of moves depend on those choices, and they also influence the performance of the search. For instance, in this example, a slightly more complex move would consist of flipping the values of two variables. Instead of 10 neighbors, this move mechanism would generate 45 neighbors, corresponding to all combinations of variables taken two at a time. A potential short-term memory structure for this neighborhood is as follows:

Attributes: Indexes of the variables that were chosen to change values.

Tabu classification: A move that includes at least one tabu-active variable is classified tabu.

Tabu tenure: Three iterations for a variable flipping from 1 to 0 and two iterations for a variable flipping from 0 to 1.

The tabu tenure recognizes that in a typical knapsack problem there will be more variables outside the knapsack (i.e., with their values set to zero) than those inside the knapsack (i.e., with their values set to one). Therefore, if an item is taken from the knapsack (i.e., its corresponding variable value is switched from one to zero), the tabu tenure keeps this item outside longer than when an item enters the knapsack. This illustrates that the flexibility of the TS methodology allows the analyst to be creative and take full advantage of context information.

Example 2

Given a matrix of weights, the linear ordering problem (LOP) consists of finding a permutation p of the columns (and rows) in order to maximize the sum of the weights in the upper triangle. Consider the LOP instance in Table 4.

The objective function value associated with the solution $p = (1, 2, 3, 4, 5, 6, 7)$ is $f(p) = 78$, corresponding to the sum of the 21 values in the upper triangle. The elements in the permutation correspond to columns (and rows) of the matrix. Suppose that a tabu search explores the solution space by moving an element $p(i)$ from its current position i to a position j, between elements $p(j-1)$ and $p(j)$. In permutation problems, these are commonly referred to as *insert* moves. For instance, inserting $p(6) = 6$ in position 2 results in $p' = (1, 6, 2, 3, 4, 5, 7)$ with $f(p') = 86$.

Since TS determines where to move via neighborhood search, it is important to be able to calculate move values with the fewest possible number of operations. One naïve way of calculating a move value consists of executing the proposed move and calculating, from scratch, the objective function of the resulting neighbor solution. Then, the value of the move is obtained as the difference between the objective function value of the current solution and the objective function value that results after executing the move. In many settings, this process is inefficient and unnecessary. Clever move value calculations can significantly improve the efficiency of the neighborhood exploration. In the case of the LOP, the move value involves only a portion of the entries in the column (and corresponding row) associated with the element that is changing positions. For instance, for the insertion of $p(6) = 6$ in position 2 to transforms p into p', only the entries in the bold rectangles in Table 5 are needed to calculate the move value v.

Table 4 Instance of the linear ordering problem

p	1	2	3	4	5	6	7
1	0	12	5	3	1	8	3
2	6	0	3	6	4	4	2
3	8	5	0	5	7	0	3
4	−2	7	2	0	−3	6	0
5	8	0	3	−1	0	4	1
6	9	1	6	2	13	0	4
7	2	9	4	−5	8	1	0

Table 5 Data for the move value calculation (bold-outlined rectangles)

p	1	2	3	4	5	6	7
1	0	12	5	3	1	8	3
2	6	0	3	6	4	4	2
3	8	5	0	5	7	0	3
4	−2	7	2	0	−3	6	0
5	8	0	3	−1	0	4	1
6	9	1	6	2	13	0	4
7	2	9	4	−5	8	1	0

The move value in this case is $v = 1 - 4 + 6 - 0 + 2 - 6 + 13 - 4 = 8$. The objective function value is $f(p') = f(p) + v = 78 + 8 = 86$. A short-term memory structure for this problem may be defined as follows:

Attribute: Index of the element that was inserted in a new position
Tabu classification: An insertion of a tabu-active element is classified tabu
Tabu tenure: Five iterations

Note that due to the nature of the moves and the tabu classification, elements during their tabu-active period may change positions. To illustrate this point, consider the insertion of $p(6) = 6$ in position 2 that results in $p' = (1, 6, 2, 3, 4, 5, 7)$. Element 6 becomes tabu-active, and therefore it is not allowed to participate in insertions for the next five iterations. However, other non-tabu-active elements may be inserted in either position 2 or position 1, causing element 6 to shift to position 3.

Long-Term Memory

In many applications, the short-term TS memory components are sufficient to produce very high-quality solutions. However, in general, TS becomes significantly stronger by including longer-term memory and its associated strategies. Long-term memory plays an important role in creating the right balance between intensification and diversification of a tabu search. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Diversification on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

Long-term memory is often implemented using a frequency-based approach. Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Like recency, frequency often is weighted or decomposed into subclasses by considering solution quality.

Frequencies consist of ratios, whose numerators represent either *transition* counts or *residence* counts. A transition count is the number of iterations where an attribute changes (enters or leaves) the solutions visited. A residence count is the number of iterations where an attribute belongs to solutions visited. The denominators generally represent one of three types of quantities: (1) the total number of occurrences of all events represented by the numerators (such as the total number of iterations), (2) the sum (or average) of the numerators, and (3) the maximum numerator value.

The use of longer-term memory does not require long solution runs before its benefits become visible. Often its improvements begin to be manifest in a relatively modest length of time and can allow solution efforts to be terminated somewhat earlier than otherwise possible, due to finding very high-quality solutions within an

economical time span. The chance of finding still better solutions as time grows – in the case where an optimal solution has not already been found – is enhanced by using longer-term TS memory in addition to short-term memory.

Residence frequencies and transition frequencies sometimes convey related information but in general carry different implications. For example, residence measures, by contrast to transition measures, are not concerned with whether an attribute changes in moving from somewhat inferior solutions to better solutions. A high residence frequency may indicate that an attribute is highly attractive if the domain consists of high-quality solutions or may indicate the opposite, if the domain consists of low-quality solutions.

Frequency-based memory is used to define penalty and incentive values to modify the evaluation of moves and therefore determine which moves are selected. In a minimization problem, the modified move value function has the following mathematical form:

$$v' = v \times (1 + pq)$$

Here, v is the original move value, p is the penalty factor, q is the frequency ratio, and v' is the modified move value. When $p = 0$, the original move value is not modified. A value of $p > 0$ penalizes the move and v' makes it less attractive. A value of $p < 0$ provides an incentive, and v' makes the move more attractive.

Long-term memory is linked to the notion of *strongly determined* and *consistent* variables. A strongly determined variable is one that cannot change its value in a given high-quality solution without seriously degrading quality or feasibility, while a consistent variable is one that frequently takes on a specific value (or a highly restricted range of values) in good solutions. The development of useful measures of "strength" and "consistency" is critical to exploiting these notions, particularly by accounting for trade-offs determined by context. However, straightforward uses of frequency-based memory for keeping track of consistency, sometimes weighted by elements of quality and influence, have produced methods with very good performance outcomes.

For instance, a penalty/incentive function based on frequency ratios may be used in a restarting procedure to generate initial solutions based on penalty/incentive functions. The function may be designed to asses a penalty on consistent variables in order to induce diversification.

Example 3

Suppose that a residency memory is added to the short-term memory TS for the knapsack problem described above. The memory accumulates the number of times each item is included in the knapsack. That is, the memory is a count of the number of times that a variable takes on the value of one. To create a frequency ratio, the numerator is defined as the number of iterations. The search is going to be restarted from a new initial solution. Restarting means that the short-term memory will be

Table 6 Reordering of the variables with the modified bang-for-buck ratios

Variable	q	r	r'	Order by r'
1	0.28	13.40	9.65	6
2	1.00	11.11	0.00	1
3	0.38	10.89	6.75	3
4	0.39	10.53	6.42	4
5	0.47	10.00	5.30	10
6	0.00	9.75	9.75	7
7	0.39	9.09	5.55	5
8	0.50	8.70	4.35	9
9	0.41	8.57	5.06	8
10	0.16	7.14	6.00	2

erased and the long-term memory will be used during the construction of the new solution. The construction is guided by the following modified bang-for-buck ratio:

$$r' = r \times (1 - q)$$

The original bang-for-buck ratio is r and q is the frequency ratio. In this penalty function, the implicit value of p is -1. The function penalizes items with high-frequency ratios (i.e., the consistent variables). In this example, long-term memory is being used to induce diversification by attempting to construct a solution that has not been visited so far. Table 6 shows the calculation of the modified bang-for-buck ratios for a given set of q values.

The last column in the table above shows the order in which the items will be considered to be included in the knapsack according to the modified ratio. This results in a solution with variables 6, 1, 3, 4, 10, and 7 set to 1. The total profit is 877 and the weight is 90. This solution, which has not been visited so far, becomes the restarting point for the search.

Strategic Oscillation

Strategic oscillation is closely linked to the origins of tabu search and provides a means to achieve an effective interplay between intensification and diversification over the intermediate to long term. Strategic oscillation operates by orienting moves in relation to the feasibility boundary, which represents a point where a search would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the feasibility boundary to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point.

The process of repeatedly approaching and crossing the feasibility boundary from different directions creates an oscillatory behavior, which gives the method its name. Control over this behavior is established by generating modified evaluations and rules of movement, depending on the region navigated and the direction of search.

The implementation of strategic oscillation entails the selection of a proximity measure and the amplitude of the oscillation. The proximity measure assigns a numerical value to moving toward or away from the feasibility boundary. This measure serves as guidance to create the oscillation pattern. The amplitude determines the point at which the search is directed to turn around once it has crossed the feasibility boundary in either direction.

Proximity measures are defined in reference to the problem constraints. For instance, for a combinatorial optimization problem consisting of selecting k out of a given number elements, a proximity measure t may be the difference between the number of elements that have been selected and k. Feasible solutions are those for which $t = 0$. Solutions with $t > 0$ have too many elements and solutions with $t < 0$ have too few. Moves can be designed to keep the search at $t = 0$ or to create an oscillation pattern around $t = 0$.

The oscillation amplitude is controlled by a number of moves m. If a move is made that causes the search to cross the feasibility boundary, then $m - 1$ is the number of additional moves that the search is allowed to make in the same direction (i.e., moving away from the feasibility boundary) before turning around.

In combinatorial optimization problems that require the selection of k elements, the rule to delete elements from the solution will typically be different in character from the one used for adding elements. In other words, one rule is not simply the inverse of the other. Rule differences are features of strategic oscillation that provide an enhanced heuristic vitality. The application of different rules may be accompanied by crossing a boundary to different depths on different sides. An option is to approach and retreat from the boundary while remaining on a single side, without crossing (i.e., electing a crossing of "zero depth").

In both one-sided and two-sided oscillation approaches, it is frequently important to spend additional search time in regions close to the feasibility boundary and, especially, to spend time at the boundary itself. In problems for which feasibility is determined by a set of constraints, vector-valued functions can be used to control the oscillation. In this case, controlling the search by bounding this function can be viewed as manipulating a parameterization of the constraint set.

Example 4

The feasibility boundary in the knapsack example is defined by the capacity of the knapsack, which in the example introduced above is a total weight of 100. To create a simple oscillation around the feasible boundary, the search is allowed to cross to the infeasible region and m is set to 1. The proximity measure t is

Table 7 Iterations of TS with strategic oscillation

Iteration	Tabu-active	Move	Solution	Profit	Weight
0			(1, 2, 3, 4, 5)	985	90
1		6	(1, 2, 3, 4, 5, 6)	1297	122
2	6	5	(1, 2, 3, 4, 6)	1177	110
3	5 6	4	(1, 2, 3, 6)	977	91
4	4 5 6	8	(1, 2, 3, 6, 8)	1177	114
5	8 4 5	6	(1, 2, 3, 8)	865	82
6	6 8 4	9	(1, 2, 3, 8, 9)	1045	103
7	9 6 8	3	(1, 2, 8, 9)	947	94
8	3 9 6	4	(1, 2, 4, 8, 9)	1147	113
9	4 3 9	8	(1, 2, 4, 9)	947	90
10	8 4 3	3	(1, 2, 3, 4, 9)	1045	99

defined as the difference between the total weight of the items in the knapsack and the knapsack capacity. Therefore, $t > 0$ indicates an infeasible solution and $t \leqslant 0$ corresponds to feasible solutions. The procedure operates with the following rules:

- When approaching the feasibility boundary from the feasible region, select the best non-tabu feasible move. If no feasible move is available, then select the non-tabu move that improves profit the most
- When approaching the feasibility boundary from the infeasible region, select the non-tabu move that removes the variable with the smallest bang-for-buck ratio.

The short-term memory structure is the same as the one defined in the Short-Term Memory section. Therefore, the move selections have to be done by considering the tabu classifications. Table 7 shows ten iterations of TS with strategic oscillation. The initial solution is given by the bang-for-buck heuristic.

The oscillation can be observed in the Weight column. Every time that the weight exceeds 100, the search turns around and moves are made to decrease the weight. The same occurs when the weight falls below 100. It is noteworthy to discuss the move made in the last iteration. The move is selected even though the move is classified tabu (note that 3 appears in the tabu-active list). At iteration 10, the initial solution is the best feasible solution found so far, and it has an objective function value of 985. Therefore, an aspiration criterion that overrides the tabu status of a move can be invoked (as discussed in the Short-Term Memory section). The criterion states that the tabu classification can be overridden if the move leads to a solution that is better than the best solution found so far. Figure 1 shows a graphical representation of the oscillation pattern.

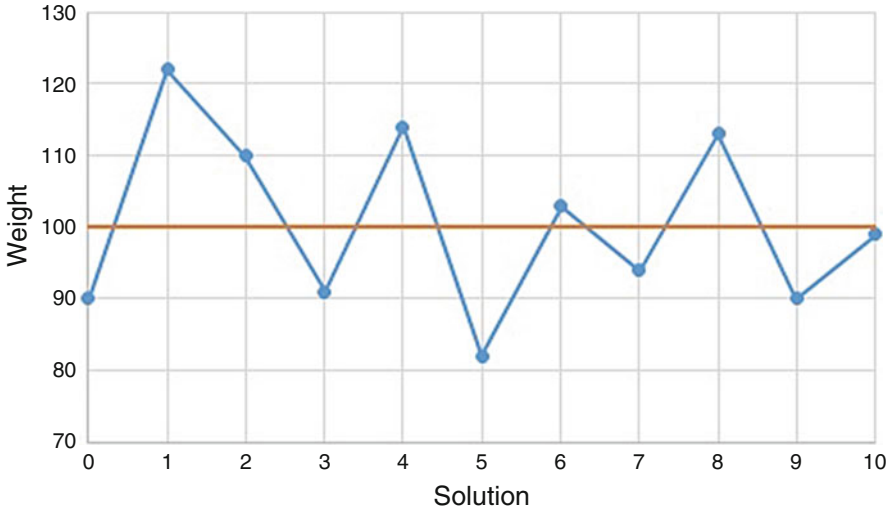


Fig. 1 Strategic oscillation around the knapsack capacity of 100

Example 5

The problem of maximizing diversity deals with selecting a subset of k elements from a given set of n elements in such a way that the diversity among the chosen elements is maximized. The best-known problem in this class deals with maximizing the sum of the diversity of the chosen elements. That is, if d_{ij} is the diversity measure between elements i and j, then the problem consists of maximizing $\sum_{i,j \in K} d_{ij}$, where K is the subset of chosen elements. The proximity measure t may be defined as the difference between the number of elements that have been selected and k, for an amplitude of m, t ranges between $-m$ and m, with $t = 0$ representing a feasible solution.

A swap move is defined as the exchange of an element currently in K with an element that is not currently in K. In this context, a swap neighborhood maintains feasibility as long as the starting solution is feasible (i.e., the solution includes k elements). In contrast, an add/delete neighborhood will not. A strategic oscillation may be defined as follows.

1. Construct a feasible solution K
2. Add elements $j \notin K$ that increases diversity the most. Elements are added one at a time until m is reached.
3. Delete elements $j \in K$ that decrease the diversity the least. Elements are deleted one at a time until m is reached.
4. Repeat 2 and 3 until a termination criterion is satisfied.

Short-term memory tabu structures may be embedded in this simple design in order to avoid cycling. For instance, elements that have been added may be declared tabu-active and are not allowed to be deleted during the next deletion cycle, and vice versa. Consider the diversity values in Table 8 for a problem with $k = 5$ and $n = 10$.

Let's assume that the starting solution is given by (1, 2, 3, 4, 5) with an objective function value of 45 and let $m = 2$. Table 9 shows 16 iterations of the strategic oscillation process, where a solution is defined by a set of 10 binary variables x_j:

The first column (Iter) shows the iteration number, where 0 is the initial solution. The next 10 columns show the values of the binary variables, where a value of 1 indicates that the element has been chosen. The Move column indicates the index of the element that enters (plus sign) or leaves (minus sign) the solution. The proximity

Table 8 Instance of the maximum diversity problem

	1	2	3	4	5	6	7	8	9	10
1		4	5	0	3	2	0	2	4	2
2			7	9	2	6	4	8	4	6
3				3	4	0	8	7	6	4
4					8	7	5	5	4	1
5						4	8	6	2	1
6							6	5	4	5
7								0	6	8
8									6	1
9										3
10										

Table 9 Sixteen iterations of the strategic oscillation process

Iter	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Move	t	MaxSum
0	1	1	1	1	1	0	0	0	0	0	+8	0	45
1	1	1	1	1	1	0	0	1	0	0	+9	1	73
2	1	1	1	1	1	0	0	1	1	0	−1	2	99
3	0	1	1	1	1	0	0	1	1	0	−5	1	81
4	0	1	1	1	0	0	0	1	1	0	−9	0	59
5	0	1	1	1	0	0	0	1	0	0	−3	−1	39
6	0	1	0	1	0	0	0	1	0	0	+6	−2	22
7	0	1	0	1	0	1	0	1	0	0	+5	−1	40
8	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**+7**	**0**	**60**
9	0	1	0	1	1	1	1	1	0	0	+3	1	83
10	0	1	1	1	1	1	1	1	0	0	−6	2	112
11	0	1	1	1	1	0	1	1	0	0	−8	1	84
12	0	1	1	1	1	0	1	0	0	0	−2	0	58
13	0	0	1	1	1	0	1	0	0	0	−4	−1	36
14	0	0	1	0	1	0	1	0	0	0	+9	−2	20
15	0	0	1	0	1	0	1	0	1	0	+10	−1	34
16	0	0	1	0	1	0	1	0	1	1		0	50

measure is given in the column labeled t. The objective function value is shown in the last column. The solution at iteration 8 is the best feasible solution, with a MaxSum value of 60 and $t = 0$. Note that an aspiration criterion is being used to override the tabu status. In particular, the tabu status of a move is overridden if it leads to a solution with an objective function value that is the best visited so far for the corresponding t value. For instance, element 3 is added at iteration 9 even though this element is tabu (since it was deleted in the last deletion cycle). However, the addition of 3 results in the solution at iteration 10 that has the best objective function value of all those solutions with $t = 2$ that have been visited so far. The same logic is applied to allow the deletion of element 6 at iteration 10.

Path Relinking

A useful integration of intensification and diversification strategies occurs in the approach called *path relinking*. This approach generates new solutions by exploring trajectories that connect reference solutions — by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Path relinking is a strategy that seeks to incorporate attributes of high-quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the path relinking approach subordinates all other considerations to the goal of choosing moves that introduce the attributes of the guiding solution, in order to create a "good attribute composition" in the current solution.

The procedure, at each step, chooses the best move, according to the change in the objective function value, from the restricted set of moves that incorporate a maximum number of the attributes of the guiding solutions. As in other applications of TS, aspiration criteria can override this restriction to allow other moves of particularly high quality to be considered.

Membership in the reference set is determined by setting a threshold which is connected to the objective function value of the best solution found during the search. For example, reference solutions might be those whose objective function value is within 10% of the best objective function value known so far.

Path relinking can be used to generate intensification strategies by choosing reference solutions that lie in a common region or that share common features. Similarly, diversification strategies based on path relinking characteristically select reference solutions that come from different regions or that exhibit contrasting features. The initiating solution can be used to give a beginning partial construction, by specifying particular attributes as a basis for remaining constructive steps. Constructive neighborhoods can be viewed as a feasibility-restoring mechanism since a null or partially constructed solution does not satisfy all conditions to qualify as feasible. Similarly, path relinking can make use of destructive neighborhoods, where

an initial solution is "overloaded" with attributes donated by the guiding solutions, and such attributes are progressively stripped away or modified until reaching a set with an appropriate composition. Destructive neighborhoods represent an instance of a feasibility-restoring function, as where an excess of elements may violate explicit problem constraints.

Path relinking consists of the following steps:

(a) Identify the neighborhood structure and associated solution attributes for path relinking (possibly different from those of other TS strategies applied to the problem).
(b) Select a collection of two or more reference solutions, and identify which members will serve as the initiating solution and the guiding solution(s). (Reference solutions can be infeasible, such as "incomplete" or "overloaded" solution components treated by constructive or destructive neighborhoods.)
(c) Move from the initiating solution toward the guiding solution(s), generating one or more intermediate solutions.

Example 6

Consider the two reference solutions in Table 10.
These solutions were found with a TS that used simple moves consisting of flipping one variable at a time. Both of these solutions are feasible. Suppose that solution 2 is chosen as the initiating solution and that solution 1 is the guiding solution. The relinking process starts by identifying the values that are common to both solutions. These correspond to variables 2, 3, and 5. Swap moves are used during the path relinking process in order to keep the intermediate solutions close to the feasibility boundary. A swap in this context corresponds to adding one item to the knapsack while taking another item out of the knapsack. This means that a variable that is set to 1 in the initiating solution and to 0 in the guiding solution must be paired with a variable that is set to 0 in the initiating solution and to 1 in the guiding solution. The pairs in our example are (1, 7), (1, 9), (4, 7), and (4, 9). Table 11 shows the intermediate solutions that result from these swaps:

The best feasible swap is (1, 7), and the search moves to the intermediate solution with profit of 1018 and weight of 96. After the move, there is only one swap left to reach the guiding solution, namely, (4, 9). In this case, the process produced only one intermediate solution; however, in general, path relinking visits more solutions during the transformation of the initiating solution into the guiding solution.

Table 10 Two reference solutions

Solution	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Profit	Weight
1	0	1	1	0	1	0	1	0	1	0	998	98
2	1	1	1	1	1	0	0	0	0	0	985	90

Table 11 Swaps in the path relinking process of Example 6

Swap	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Profit	Weight
(1, 7)	0	1	1	1	1	0	1	0	0	0	1018	96
(1, 9)	0	1	1	1	1	0	0	0	1	0	1098	106
(4, 7)	1	1	1	0	1	0	1	0	0	0	885	82
(4, 9)	1	1	1	0	1	0	0	0	1	0	965	92

Conclusions

This chapter presented the essential elements of tabu search, namely, short-term memory, long-term memory, strategic oscillation, and path relinking. Attention was given to these core components of the tabu search framework for the following reasons:

- Such a focus may help to uncover a better form for the strategies associated with these core components.
- Weaknesses and strengths of these core components, when studied in isolation from other ideas, may stand out more clearly, thus yielding insights into the features that a complete approach may require to produce better methods.
- For methods which are susceptible to highly modular implementations, as typically occurs for tabu search, simpler designs can readily be made a part of more complex designs.

While TS implementations using a combinations of these elements are likely to produce decent outcomes, readers are encouraged to pursue more complete approaches by examining strategies outlined in the publications referenced below, in particular those in the Tabu Search book [4].

A great deal remains to be learned about tabu search. Evidently, very little is known about how human beings use memory in problem-solving. It is not inconceivable that discoveries about effective uses of memory within search methods will provide clues about strategies that humans skillfully employ. The potential links between the areas of heuristic search and psychology have barely been examined. The numerous successes of tabu search implementations provide encouragement that such relationships might be profitable to investigate more fully.

Cross-References

▸ Adaptive and Multilevel Metaheuristics
▸ Scatter Search
▸ Theory of Local search

References

1. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Anal 8:156–166
2. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13:533–549
3. Glover F (1990) Tabu search: a tutorial. Interfaces 20(4):74–94
4. Glover F, Laguna M (1997) Tabu search. Springer, New York
5. Glover F, Laguna M (2002) Tabu search. In: Pardalos PM, Resende MGC (eds) Handbook of applied optimization. Oxford University Press, New York, pp 194–208
6. Glover F, Laguna M (2013) Tabu search. In: Du DZ, Pardalos PM (eds) Handbook of combinatorial optimization, 2nd edn. Springer, New York, pp 3261–3362
7. Glover F, Laguna M, Martí R (2007) Principles of tabu search. In: Gonzalez T (ed) Approximation algorithms and metaheuristics, Chapter 23. Chapman & Hall/CRC, New York
8. Schneider U (2010) A tabu search tutorial based on a real-world scheduling problem. Cent Eur J Oper Res 19(4):467–493

Variable Neighborhood Search

26

Pierre Hansen and Nenad Mladenović

Contents

Introduction 760
Basic Schemes 761
Some Extensions 769
VNS for Mixed-Integer Linear Programming 774
VNS for Continuous Global Optimization 775
Variable Neighborhood Programming 779
Discovery Science 782
Conclusions 784
References 785

Abstract

Variable neighborhood search (VNS) is a metaheuristic for solving combinatorial and global optimization problems. Its basic idea is systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. In this chapter we present the basic schemes of variable neighborhood search and some of its extensions. We next present four families of applications of VNS in which it has proved to be very successful: (i) finding feasible solutions to large mixed-integer linear programs, by hybridization of VNS and local branching, (ii) finding good feasible solutions to continuous nonlinear programs, (iii) finding programs in automatic fashion

P. Hansen (✉)
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada
e-mail: pierreh@crt.umontreal.ca

N. Mladenović
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

LAMIH, University of Valenciennes, Famars, France
e-mail: nenadmladenovic12@gmail.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_19

(artificial intelligence field) by building *variable neighborhood programming* methodology, and (iv) exploring graph theory in order to find conjectures, refutations, and proofs or ideas of proofs.

Keywords
Optimization · Metaheuristics · Artificial intelligence · Variable neighborhood search

Introduction

Optimization tools have greatly improved during the last two decades. This is due to several factors: (i) progress in mathematical programming theory and algorithmic design, (ii) rapid improvement in computer performances, and (iii) better communication of new ideas and integration of them in largely used complex softwares. Consequently, many problems long viewed as out of reach are currently solved, sometimes in very moderate computing times. This success, however, has led to address much larger instances and more difficult classes of problems. Many of these may again only be solved heuristically. Therefore thousands of papers describing, evaluating, and comparing new heuristics appear each year. Keeping abreast of such a large literature is a challenge. Metaheuristics or general frameworks for building heuristics are therefore needed in order to organize the study of heuristics. As evidenced by this handbook, there are many of them. Some desirable properties of metaheuristics [24,27,29] are listed in the concluding section of this chapter.

Variable neighborhood search (VNS) is a metaheuristic proposed by some of the present authors a dozen years ago [40]. Earlier work that motivated this approach is given in [8, 13, 18, 38]. It is based upon the idea of systematic change of neighborhood both in a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. Originally designed for approximate solution of combinatorial optimization problems, it was extended to address, nonlinear programs, and recently mixed-integer nonlinear programs. In addition VNS has been used as a tool for automated or computer-assisted graph theory. This led to the discovery of over 1500 conjectures in that field, the automated proof of more than half of them as well as the unassisted proof of about 400 of them by many mathematicians. Moreover, VNS methodology is recently applied in artificial intelligence as automatic programming technique.

Applications are rapidly increasing in number and pertain to many fields: location theory, cluster analysis, scheduling, vehicle routing, network design, lot sizing, artificial intelligence, engineering, pooling problems, biology, phylogeny, reliability, geometry, telecommunication design, etc. References are too numerous to be listed here, but many of them can be found in [25, 28, 29] and in the following special issues devoted to VNS: *IMA Journal of Management Mathematics* [37], *European Journal of Operational Research* [29], *Journal of heuristics* [47], *Computers and Operations Research* [45] and *Journal of Global Optimization* [11].

This chapter is organized as follows. In the next section we present the basic schemes of VNS, i.e., variable neighborhood descent (VND), reduced VNS (RVNS), basic VNS (BVNS), and general VNS (GVNS). In addition, to the list of basic schemes, we add skewed VNS (SVNS) and variable neighborhood decomposition search (VNDS). Three important extensions are presented in Section "Some Extensions": primal-dual VNS (PD-VNS), formulation space search (FSS), and recently developed variable formulation search (VFS). The remainder of the paper describes applications of VNS to several classes of large-scale and complex optimization problems for which it has proven to be particularly successful. Finding feasible solutions to large mixed-integer linear programs with VNS is discussed in Section "VNS for Mixed-Integer Linear Programming". Section "VNS for Continuous Global Optimization" addresses ways to apply VNS in continuous global optimization. Recent application of VNS in *automatic programming* field is discussed in section "Variable Neighborhood Programming". Applying VNS to graph theory per se (and not just to particular optimization problems defined on graphs) is discussed in Section "Discovery Science". Conclusions are drawn in Section "Conclusions". There the desirable properties of any metaheuristic are listed as well.

Basic Schemes

A deterministic optimization problem may be formulated as

$$\min\{f(x)|x \in X, X \subseteq \mathcal{S}\}, \tag{1}$$

where $\mathcal{S}, X, x$, and f, respectively, denote the *solution space* and *feasible set*, a *feasible solution*, and a real-valued *objective function*. If $\mathcal{S}$ is a finite but large set, a *combinatorial optimization* problem is defined. If $\mathcal{S} = \mathbb{R}^n$, we refer to *continuous optimization*. A solution $x^* \in X$ is *optimal* if

$$f(x^*) \leq f(x), \ \forall x \in X.$$

An *exact algorithm* for problem (1), if one exists, finds an optimal solution x^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $X = \emptyset$, or the solution is unbounded. Moreover, in practice, the time needed to do so should be finite (and not too long). For continuous optimization, it is reasonable to allow for some degree of tolerance, i.e., to stop when sufficient convergence is detected.

Let us denote with $\mathcal{N}_k$, $(k = 1, \ldots, k_{\max})$, a finite set of preselected neighborhood structures, and with $\mathcal{N}_k(x)$ the set of solutions in the kth neighborhood of x. Most local search heuristics use only one neighborhood structure, i.e., $k_{\max} = 1$. Neighborhoods $\mathcal{N}_k$ may be induced from one or more metric (or quasi-metric) functions introduced into a solution space S. An *optimal solution* x_{opt} (or global minimum) is a feasible solution where a minimum is reached. We call $x' \in X$ a *local minimum* of (1) with respect to $\mathcal{N}_k$ (w.r.t. $\mathcal{N}_k$ for short), if there is no solution

$x \in \mathcal{N}_k(x') \subseteq X$ such that $f(x) < f(x')$. Metaheuristics (based on local search procedures) try to continue the search by other means after finding the first local minimum. VNS is based on three simple facts:

Fact 1 *A local minimum w.r.t. one neighborhood structure is not necessarily so for another;*

Fact 2 *A global minimum is a local minimum w.r.t. all possible neighborhood structures;*

Fact 3 *Local minima w.r.t. one or several $\mathcal{N}_k$ are relatively close to each other, for many problems.*

This last observation, which is empirical, implies that a local optimum often provides some information about the global one. This may for instance be several variables with the same value in both. However, it is usually not known which ones are such. An organized study of the neighborhoods of this local optimum is therefore in order, until a better solution is found.

In order to solve (1) by using several neighborhoods, facts 1–3 can be used in three different ways: (i) deterministic, (ii) stochastic, and (iii) both deterministic and stochastic. We first give in Algorithm 1 the steps of the neighborhood change function that will be used later.

Algorithm 1: Neighborhood change

Function `NeighborhoodChange` (x, x', k)
1 **if** $f(x') < f(x)$ **then**
2 $\quad x \leftarrow x'$; $k \leftarrow 1$ // Make a move
else
3 $\quad k \leftarrow k + 1$ // Next neighborhood

Function `NeighborhoodChange()` compares the new value $f(x')$ with the incumbent value $f(x)$ obtained from the kth neighborhood (line 1). If an improvement is obtained, k is returned to its initial value and the new incumbent updated (line 2). Otherwise, the next neighborhood is considered (line 3).

(i) The *variable neighborhood descent* (VND) method is obtained if a change of neighborhoods is performed in a deterministic way. Its steps are presented in Algorithm 2, where neighborhoods are denoted as $N_\ell, \ell = 1, \ell_{\max}$.

Most local search heuristics use in their descents a single or sometimes two neighborhoods ($\ell_{\max} \leq 2$). Note that the final solution should be a local minimum w.r.t. all $\ell_{\max}$ neighborhoods, and thus chances to reach a global one are larger than by using a single structure. Besides this *sequential* order of neighborhood structures in VND above, one can develop a *nested* strategy. Assume, e.g., that $\ell_{\max} = 3$; then a possible nested strategy is: perform VND from Algorithm 2 for the first two neighborhoods from each point x' that belongs to the third ($x' \in N_3(x)$). Such an approach is successfully applied in [7, 25] and [5].

Algorithm 2: Steps of the basic VND

Function VND $(x, \ell_{\max})$
1 $\ell \leftarrow 1$
2 **repeat**
3 $\quad x' \leftarrow \arg\min_{y \in N_\ell(x)} f(x)$ // Find the best neighbor in $N_\ell(x)$
4 $\quad$ NeighborhoodChange (x, x', ℓ) // Change neighborhood
until $\ell = \ell_{\max}$

(ii) The *reduced VNS* (RVNS) method is obtained if random points are selected from $\mathcal{N}_k(x)$ and no descent is made. Rather, the values of these new points are compared with that of the incumbent, and updating takes place in case of improvement. We assume that a stopping condition has been chosen, among various possibilities, e.g., the maximum CPU time allowed $t_{\max}$, or the maximum number of iterations between two improvements. To simplify the description of the algorithms, we always use $t_{\max}$ below. Therefore, RVNS uses two parameters: $t_{\max}$ and $k_{\max}$. Its steps are presented in Algorithm 3.

Algorithm 3: Steps of the reduced VNS

Function RVNS$(x, k_{\max}, t_{\max})$
1 **repeat**
2 $\quad k \leftarrow 1$
3 $\quad$ **repeat**
4 $\quad\quad x' \leftarrow$ Shake(x, k)
5 $\quad\quad$ NeighborhoodChange (x, x', k)
$\quad$ **until** $k = k_{\max}$
6 $\quad t \leftarrow$ CpuTime()
until $t > t_{\max}$

With the function Shake represented in line 4, we generate a point x' at random from the kth neighborhood of x, i.e., $x' \in \mathcal{N}_k(x)$. Its steps are given in Algorithm 4, where it is assumed that points from $\mathcal{N}_k(x)$ are $\{x^1, \ldots, x^{|\mathcal{N}_k(x)|}\}$. RVNS is useful

Algorithm 4: Steps of the shaking function

Function Shake(x, x', k)
1 $w \leftarrow [1+\text{Rand}(0, 1) \times |\mathcal{N}_k(x)|]$
2 $x' \leftarrow x^w$

for very large instances for which local search is costly. It can be used as well for finding initial solutions for large problems before decomposition. It has been observed that the best value for the parameter $k_{\max}$ is often 2 or 3. In addition, a

maximum number of iterations between two improvements is usually used as the stopping condition. RVNS is akin to a Monte Carlo method, but is more systematic (see, e.g., [42] where results obtained by RVNS were 30 % better than those of the Monte Carlo method in solving a continuous min-max problem). When applied to the p-median problem, RVNS gave equally good solutions as the *fast interchange* heuristic of [56] while being 20–40 times faster [30].

(iii) The *basic VNS* (VNS) method [40] combines deterministic and stochastic changes of neighborhood. The deterministic part is represented by a local search heuristic. It consists in (a) choosing an initial solution x, (b) finding a direction of descent from x (within a neighborhood $N(x)$), and (c) moving to the minimum of $f(x)$ within $N(x)$ along that direction. If there is no direction of descent, the heuristic stops, and otherwise it is iterated. Usually the steepest descent direction, also referred to as *best improvement*, is used (`BestImprovement`). This set of rules is summarized in Algorithm 5, where we assume that an initial solution x is given. The output consists of a local minimum, also denoted by x, and its value.

Algorithm 5: Best improvement (steepest descent) heuristic

Function `BestImprovement`(x)
repeat
  $x' \leftarrow x$
  $x \leftarrow \arg\min_{y \in N(x)} f(y)$
until $(f(x) \geq f(x'))$

As *steepest descent* heuristic may be time-consuming, an alternative is to use the *first descent* heuristic. Vectors $x_i \in N(x)$ are then enumerated systematically and a move is made as soon as a direction for the descent is found. This is summarized in Algorithm 6.

Algorithm 6: First improvement (first descent) heuristic

Function `FirstImprovement`(x)
repeat
  $x' \leftarrow x$; $i \leftarrow 0$
  repeat
    $i \leftarrow i + 1$
    $x \leftarrow \arg\min\{f(x), f(x_i)\}$, $x_i \in N(x)$
  until $(f(x) < f(x_i)$ or $i = |N(x)|)$
until $(f(x) \geq f(x'))$

The stochastic phase is represented by random selection of one point from the k-th neighborhood. The steps of the BVNS are given on Algorithm 7.

Algorithm 7: Steps of the basic VNS

Function BVNS($x, k_{\max}, t_{\max}$)

```
1 t ← 0
2 while t < t_max do
3     k ← 1
4     repeat
5         x' ← Shake(x, k)                    // Shaking
6         x'' ← BestImprovement(x')           // Local search
7         NeighborhoodChange(x, x'', k)       //Change neighborhood
      until k = k_max
8     t ← CpuTime()
```

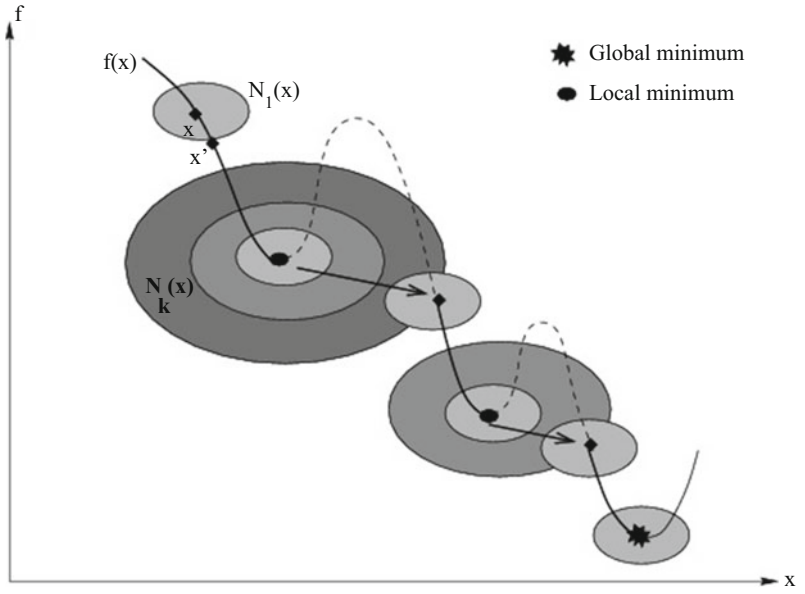


Fig. 1 Basic VNS

Often successive neighborhoods $\mathcal{N}_k$ are nested. Note that point x' is generated at random in Step 4 in order to avoid cycling, which might occur if a deterministic rule were applied. Basic VNS is also illustrated in Fig. 1.

Example. We illustrate the basic step on a minimum k-cardinality tree instance taken from [34] (see Fig. 2). The minimum k-cardinality tree problem on graph G (k-card for short) consists in finding a sub-tree of G with exactly k edges whose sum of weights is minimum.

The steps of BVNS for solving the 4-card problem are illustrated in Fig. 3. In Step 0 the objective function value, i.e., the sum of edge weights, is equal to 40;

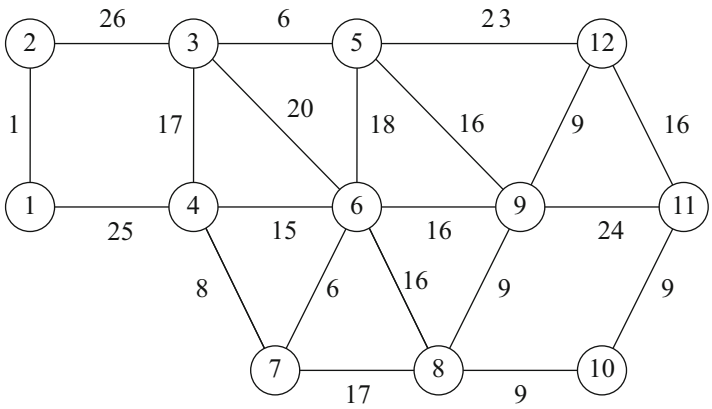


Fig. 2 4-cardinality tree problem

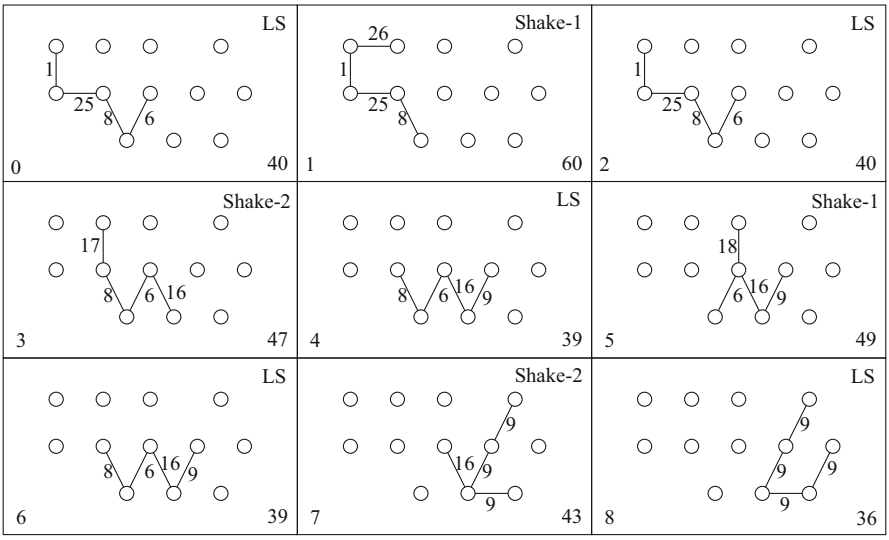


Fig. 3 Steps of the basic VNS for solving 4-card tree problem

it is indicated in the right bottom corner of the figure. That first solution is a local minimum with respect to the edge-exchange neighborhood structure (one edge in, one out). After shaking, the objective function is 60, and after another local search, we are back to the same solution. Then, in Step 3, we take out 2 edges and add another 2 at random, and after a local search, an improved solution is obtained with a value of 39. Continuing in that fashion, we obtain in Step 8 the optimal solution with an objective function value equal to 36.

(iv) *General VNS.* Note that the local search Step 5 may also be replaced by VND (Algorithm 2). Using this general VNS (VNS/VND) approach led to some of the most successful applications reported (see, e.g., [1, 7, 9, 10, 25, 31, 52, 53]). The steps of general VNS (GVNS) are given in Algorithm 8 below.

Algorithm 8: Steps of the general VNS

Function GVNS $(x, \ell_{\max}, k_{\max}, t_{\max})$
repeat
  $k \leftarrow 1$
  repeat
    $x' \leftarrow$ Shake(x, k)
    $x'' \leftarrow$ VND$(x', \ell_{\max})$
    NeighborhoodChange(x, x'', k)
  until $k = k_{\max}$
  $t \leftarrow$ CpuTime()
until $t > t_{\max}$

(v) The *skewed VNS* (SVNS) method [23] addresses the problem of exploring valleys far from the incumbent solution. Indeed, once the best solution in a large region has been found, it is necessary to go quite far to obtain an improved one. Solutions drawn at random in faraway neighborhoods may differ substantially from the incumbent, and VNS may then degenerate, to some extent, into the multistart heuristic (in which descents are made iteratively from solutions generated at random, and that is known not to be very efficient). So some compensation for distance from the incumbent must be made, and a scheme called skewed VNS is proposed for that purpose. Its steps are presented in Algorithms 9, 10, and 11. The KeepBest(x, x') function in Algorithm 10 simply keeps the better one of solutions x and x'.

Algorithm 9: Steps of neighborhood change for the skewed VNS

Function NeighborhoodChangeS(x, x'', k, α)
if $f(x'') - \alpha\rho(x, x'') < f(x)$ **then**
  $x \leftarrow x''$; $k \leftarrow 1$
else
  $k \leftarrow k + 1$

SVNS makes use of a function $\rho(x, x'')$ to measure distance between the incumbent solution x and the local optimum found x''. The distance function used to define $\mathcal{N}_k$, as in the above examples, could be used also for this purpose. The parameter α must be chosen in such a way to accept movement to valleys faraway from x when $f(x'')$ is larger than $f(x)$ but not too much larger (otherwise one will

Algorithm 10: Steps of the skewed VNS

Function SVNS $(x, k_{\max}, t_{\max}, \alpha)$
1 **repeat**
2 $k \leftarrow 1$; $x_{\text{best}} \leftarrow x$
3 **repeat**
4 $x' \leftarrow$ Shake(x, k)
5 $x'' \leftarrow$ FirstImprovement(x')
6 KeepBest (x_{best}, x)
7 NeighborhoodChangeS(x, x'', k, α)
until $k = k_{\max}$
8 $x \leftarrow x_{\text{best}}$
9 $t \leftarrow$ CpuTime()
until $t > t_{\max}$

Algorithm 11: Keep the better solution

Function KeepBest(x, x')
1 **if** $f(x') < f(x)$ **then**
2 $x \leftarrow x'$

always leave x). A good value for α is to be found experimentally in each case. Moreover, in order to avoid frequent moves from x to a close solution, one may take a smaller value for α when $\rho(x, x'')$ is small. More sophisticated choices for a function of $\alpha\rho(x, x'')$ could be made through some learning process.

(vi) *Variable neighborhood decomposition search* (VNDS). The VNDS method [30] extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem. Its steps are presented in Algorithm 12, where t_d is an additional parameter and represents the running time given for solving decomposed (smaller sized) problems by VNS.

For ease of presentation, but without loss of generality, we assume that the solution x represents the set of some elements. In Step 4 we denote with y a set of k solution attributes present in x' but not in x ($y = x' \setminus x$). In Step 5 we find the local optimum y' in the space of y; then we denote with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$). We notice that exploiting some *boundary effects* in a new solution can significantly improve the solution quality. That is why, in Step 6, we find the local optimum x''' in the whole space S using x'' as an initial solution. If this is time-consuming, then at least a few local search iterations should be performed.

VNDS can be viewed as embedding the classical successive approximation scheme (which has been used in combinatorial optimization at least since the 1960s; see, e.g., [21]) in the VNS framework.

Algorithm 12: Steps of VNDS

Function VNDS $(x, k_{\max}, t_{\max}, t_d)$
repeat
  $k \leftarrow 1$
  repeat
    $x' \leftarrow$ Shake (x, k); $y \leftarrow x' \setminus x$
    $y' \leftarrow \text{VNS}(y, k, t_d)$; $x'' = (x' \setminus y) \cup y'$
    $x''' \leftarrow$ FirstImprovement(x'')
    NeighborhoodChange(x, x''', k)
  until $k = k_{\max}$
until $t > t_{\max}$

Some Extensions

(i) *Primal-dual VNS*. For most modern heuristics, the difference in value between the optimal solution and the obtained one is completely unknown. Guaranteed performance of the primal heuristic may be determined if a lower bound on the objective function value is known. To that end, the standard approach is to relax the integrality condition on the primal variables, based on a mathematical programming formulation of the problem. However, when the dimension of the problem is large, even the relaxed problem may be impossible to solve exactly by standard commercial solvers. Therefore, it seems a good idea to solve dual relaxed problems heuristically as well. In this way we get guaranteed bounds on the primal heuristic's performance. The next problem arises if we want to get an exact solution within a branch-and-bound framework since having the approximate value of the relaxed dual does not allow us to branch in an easy way, e.g., exploiting complementary slackness conditions. Thus, the exact value of the dual is necessary.

In primal-dual VNS (PD-VNS) [22] one possible general way to get both the guaranteed bounds and the exact solution is proposed. Its steps are given in Algorithm 13. In the first stage a heuristic procedure based on VNS is used to

Algorithm 13: Steps of the basic PD-VNS

Function PD-VNS $(x, \ell_{\max}, k_{\max}, t_{\max})$
BVNS $(x, \ell_{\max}, k_{\max}, t_{\max})$ // Solve primal by VNS
DualFeasible(x, y) // Find (infeasible) dual such that $f_P = f_D$
DualVNS(y) // Use VNS do decrease infeasibility
DualExact(y) // Find exact (relaxed) dual
BandB(x, y) // Apply branch-and-bound method

obtain a near-optimal solution. In [22] it is shown that VNS with decomposition is a very powerful technique for large-scale simple plant location problems (SPLP) with up to 15,000 facilities and 15,000 users. In the second phase the objective is to find an exact solution of the relaxed dual problem. Solving the relaxed dual is accomplished in three stages: (i) find an initial dual solution (generally infeasible) using the primal heuristic solution and complementary slackness conditions, (ii) find a feasible solution by applying VNS to the unconstrained nonlinear form of the dual, and (iii) solve the dual exactly starting with initial feasible solution using a customized "sliding simplex" algorithm that applies "windows" on the dual variables substantially reducing the size of the problem. In all problems tested, including instances much larger than previously reported in the literature, the procedure was able to find the exact dual solution in reasonable computing time. In the third and final phase armed with tight upper and lower bounds, obtained respectively, from the heuristic primal solution in phase one and the exact dual solution in phase two, we apply a standard branch-and-bound algorithm to find an optimal solution of the original problem. The lower bounds are updated with the dual sliding simplex method and the upper bounds whenever new integer solutions are obtained at the nodes of the branching tree. In this way it was possible to solve exactly problem instances of sizes up to 7000×7000 for uniform fixed costs and $15{,}000 \times 15{,}000$ otherwise.

(ii) *Variable neighborhood formulation space search*. Traditional ways to tackle an optimization problem consider a given formulation and search in some way through its feasible set X. The fact that a same problem may often be formulated in different ways allows to extend search paradigms to include jumps from one formulation to another. Each formulation should lend itself to some traditional search method, its "local search" that works totally within this formulation, and yields a final solution when started from some initial solution. Any solution found in one formulation should easily be translatable to its equivalent solution in any other formulation. We may then move from one formulation to another using the solution resulting from the former's local search as initial solution for the latter's local search. Such a strategy will of course only be useful when local searches in different formulations behave differently.

This idea was recently investigated in [43] using an approach that systematically changes formulations for solving circle packing problems (CPP). It is shown there that a stationary point of a nonlinear programming formulation of CPP in Cartesian coordinates is not necessarily also a stationary point in a polar coordinate system. A method called *reformulation descent* (RD) that alternates between these two formulations until the final solution is stationary with respect to both is suggested. Results obtained were comparable with the best known values, but they were achieved some 150 times faster than by an alternative single formulation approach. In the same paper the idea suggested above of *formulation space search* (FSS) is also introduced, using more than two formulations. Some research in that direction has been reported in [32, 43, 44, 50]. One methodology that uses variable neighborhood

idea in searching through the formulation space is given in Algorithms 14 and 15. Here ϕ (ϕ') denotes a formulation from given space $\mathcal{F}$, x (x') denotes a solution in the feasible set defined with that formulation, and $\ell \leq \ell_{\max}$ is the formulation neighborhood index. Note that Algorithm 15 uses a reduced VNS strategy in $\mathcal{F}$.

Algorithm 14: Formulation change function

Function `FormulationChange`$(x, x', \phi, \phi', \ell)$
if $f(\phi', x') < f(\phi, x)$ **then**
  $\phi \leftarrow \phi'$; $x \leftarrow x'$; $\ell \leftarrow \ell_{\min}$
else
  $\ell \leftarrow \ell + \ell_{\text{step}}$

Algorithm 15: Reduced variable neighborhood FSS

Function `VNFSS`$(x, \phi, \ell_{\max})$
repeat
  $\ell \leftarrow 1$ // Initialize formulation in $\mathcal{F}$
  while $\ell \leq \ell_{\max}$ **do**
    `ShakeFormulation`(x,x',ϕ,ϕ',ℓ) // $(\phi',x') \in (N_\ell(\phi),\mathcal{N}(x))$ at random
    `FormulationChange`(x,x',ϕ,ϕ',ℓ) // Change formulation
until *some stopping condition is met*

(iii) *Variable formulation search*. Many optimization problems in the literature, like min-max type of problems, present a flat landscape. This means that, given a formulation of the problem, there are many neighboring solutions with the same value of the objective function. When this happens, it is difficult to determine which neighborhood solution is more promising to continue the search. To address this drawback, the use of alternative formulations of the problem within VNS is proposed in [41, 46, 49]. In [49] it is named variable formulation search (VFS). It combines the change of neighborhood within the VNS framework, with the use of alternative formulations. In particular, the alternative formulations will be used to compare different solutions with the same value of the objective function, when considering the original formulation.

Let us assume that, besides the original formulation and the corresponding objective function $f_0(x)$, there are p other formulations denoted as $f_1(x), \ldots, f_p(x), x \in X$. Note that two formulations are equivalent if the optimal solution of one is the optimal solution of the other and vice versa. Without loss of clarity, we will denote different formulations as different objectives $f_i(x), i = 1, \ldots, p$. The idea of VFS is to add the procedure `Accept`(x, x', p), given in

Algorithm 16 in all three basic steps of BVNS: `Shaking`, `LocalSearch`, and `NeighbourhoodChange`. Clearly, if a better solution is not obtained by any formulation among the p preselected, the move is rejected. The next iteration in the loop of Algorithm 16 will take place only if the objective function values according to all previous formulations are equal.

Algorithm 16: Accept procedure with p secondary formulations

```
1: logical function Accept (x, x', p)
2: for i = 0 to p do
3:    condition1 = f_i(x') < f_i(x)
4:    condition2 = f_i(x') > f_i(x)
5:    if (condition1) then
6:       Accept← True; return
7:    else if (condition2) then
8:       Accept ← False return
9:    end if
10: end for
```

If `Accept` (x, x', p) is included into `LocalSearch` subroutine of BVNS, then it will not stop the first time a non-improved solution is found. In order to stop `LocalSearch` and thus claim that x' is local minimum, x' should not be improved by any among the p different formulations. Thus, for any particular problem, one needs to design different formulations of the problem considered and decide the order they will be used in the `Accept` subroutine. Answers to those two questions are problem specific and sometimes not easy. The `Accept` (x, x', p) subroutine can obviously be added to `NeighbourhoodChange` and `Shaking` steps of BVNS from Algorithm 7 as well.

In [46], three evaluation functions, or acceptance criteria, within `Neighborhood Change` step are used in solving **bandwith minimization problem**. This min-max problem consists in finding permutations of rows and columns of a given square matrix such that the maximal distance of nonzero element from the main diagonal in the corresponding row is minimum. Solution x may be presented as a labeling of a graph and the move from x to x' as $x \leftarrow x'$. Three criteria used are (1) the simplest one is based on the objective function value $f_0(x)$ (bandwidth lenth), (2) the total number of critical vertices $f_1(x)$ $(f_1(x') < f_1(x)|)$, and (3) $f_3(x, x') = \rho(x, x') - \alpha$, if $(f_0(x') = f_0(x)$ and $f_1(x') = f_1(x))$, but x and x' are relatively far one from another $\rho(x, x') > \alpha$, where α is an additional parameter. The idea for a move to even worse solution if it is very far is used within skewed VNS. However, move to the solution with the same value only if its Hamming distance from the incumbent is greater than α that is performed in [46].

In [41] different mathematical programming formulation of the original problem is used as a secondary objective within `Neighborhood Change` function of VNS. There, two combinatorial optimization problems on the graph are considered: **metric dimension problem** and **minimal doubly resolving set problem**.

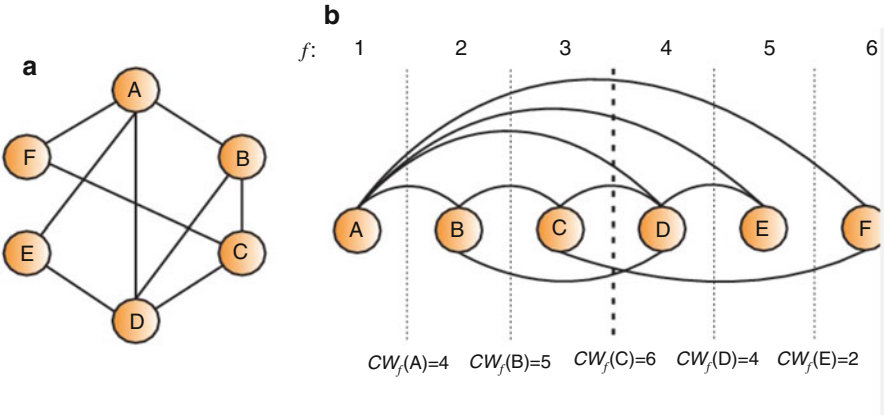


Fig. 4 Cutwidth minimization example as in [49]

Table 1 Impact of the use of alternative formulations in the search process, within 30 s

	BVNS	VFS_1	VFS_2	VFS_3
Avg.	137.31	93.56	91.56	90.75
Dev. (%)	192.44	60.40	49.23	48.22

More general VFS approach is done in [49], where the **cutwidth graph minimization problem** (CWP) is considered. CWP also belongs to min-max problem family. For a given graph, one needs to make sequence of nodes such that the maximum cutwidth is minimum. The definition of the cutwidth of the graph is clear from Fig. 4, where at (a) graph G with six vertices and nine edges is given.

At Fig. 4b, ordering x of the vertices of the graph in (a) with the corresponding cutwidth CW values of each vertex is presented. It is clear that the CW represents the number of cut edges between two consecutive nodes in the solution x. The cutwidth value $f_0(x) = CW(x)$ of the ordering $x = (A, B, C, D, E, F)$ is equal to $f_0(x) = \max\{4, 5, 6, 4, 2\} = 6$ (see Fig. 4). One needs to find order x that minimizes the maximum cutwidth value of each vertex.

Two additional formulations are used in [49] and implemented within VND local search. In Table 1 the results obtained with four different VFS variants, when executing them for 30 s over each instance of the *test* data set, are presented. The column BVNS represents a heuristic based on the BVNS which make use only of the original formulation of the CMP. VFS_1 denotes BVNS heuristic that uses only one secondary criterion. VFS_2 is equivalent to the previous one with the difference that now only f_2 is considered (instead of f_1). Finally, the fourth column of the table, denoted as VFS_3, combines the original formulation of the CMP with the two alternative ones, in the way presented in Algorithm 16. All the algorithms were configured with $k_{\max} = 0.1n$ and they start from the same random solution. It appears that the significant improvements in the solution quality are obtained when at least one secondary formulation is used in case of ties (compare, e.g., 192.44 % and 60.40 % deviations from the best known solutions reported by BVNS and VFS_1,

Table 2 Comparison with the state-of-the-art algorithms over the grid and HB data sets

	81 'grid' test instances				86 HB instances			
	GPR [2]	SA [12]	SS [48]	VFS [49]	GPR [2]	SA [12]	SS [48]	VFS [49]
Avg.	38.44	16.14	13.00	12.23	364.83	346.21	315.22	314.39
Dev. (%)	201.81	25.42	7.76	3.25	95.13	53.30	3.40	1.77
#Opt.	2	37	44	59	2	8	47	61
CPU t (s)	235.16	216.14	210.07	90.34	557.49	435.40	430.57	128.12

respectively). Additional improvement is obtained if all three formulations are used (in VFS_3).

Comparison of VFS_3 and state-of-the-art heuristics are given in Table 2. It appears that the best quality results are obtained by VFS in less computing time.

VNS for Mixed-Integer Linear Programming

The mixed-integer linear programming (MILP) problem consists of maximizing or minimizing a linear function, subject to equality or inequality constraints and integrality restrictions on some of the variables. The mixed integer programming problem (P) can be expressed as

$$(MILP) \quad \left[\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in M = \{1, 2, \ldots, m\} \\ & x_j \in \{0, 1\} & \forall j \in \mathcal{B} \neq \emptyset \\ & x_j \geq 0, \texttt{integer} & \forall j \in \mathcal{G} \\ & x_j \geq 0 & \forall j \in \mathcal{C} \end{array}\right.$$

where the set of indices $N = \{1, 2, \ldots, n\}$ is partitioned into three subsets $\mathcal{B}, \mathcal{G}$, and $\mathcal{C}$, corresponding to binary, general integer, and continuous variables, respectively.

Numerous combinatorial optimization problems, including a wide range of practical problems in business, engineering, and science, can be modeled as MILP problems. Its several special cases, such as knapsack, set packing, cutting and packing, network design, protein alignment, traveling salesman, and other routing problems, are known to be NP-hard [19]. There are several commercial solvers such as CPLEX [33] for solving MILPs. Methods included in such software packages are usually of branch-and-bound (B&B) or of branch-and-cut (B&C) types. Basically, those methods enumerate all possible integer values in some order and perform some restrictions for the cases where such enumeration cannot improve the currently best solution.

The connection between local search type of heuristics and exact solvers may be established by introducing the so-called local branching constraint [17]. By adding just one constraint into the (MILP), the kth neighborhood of MILP is defined.

This allows the use of all local search-based Metaheuristics, such as tabu search, simulating annealing, VNS, etc. More precisely, given two solutions x and y of the problem (MILP), we define the distance between x and y as

$$\delta(x, y) = \sum_{j \in \mathcal{B}} \mid x_j - y_j \mid.$$

Let X be the solution space of the problem (MILP) considered. The neighborhood structures $\{\mathcal{N}_k \mid k = 1, \ldots, k_{\max}\}$ can be defined, knowing the distance $\delta(x, y)$ between any two solutions $x, y \in X$. The set of all solutions in the kth neighborhood of $y \in X$ is denoted as $\mathcal{N}_k(y)$ where

$$\mathcal{N}_k(y) = \{x \in X \mid \delta(x, y) \leq k\}.$$

For the pure 0-1 MILP given above, ($\mathcal{G} = \emptyset$, $\delta(.,.)$ represents the Hamming distance, and $\mathcal{N}_k(y)$ may be expressed by the following so-called local branching constraint:

$$\delta(x, y) = \sum_{j \in S}(1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k, \tag{2}$$

where $S = \{j \in \mathcal{B} \mid y_j = 1\}$.

In [31] a general VNS procedure for solving 0-1 MILPs is developed (see Algorithm 17).

An exact MILP solver (CPLEX) is used as a black box for finding the best solution in the neighborhood, based on the given formulation (MILP) plus the added local branching constraints. Shaking is performed using the Hamming distance defined above. The detailed explanation of VNB method given in Algorithm 17 below and the meaning of all variables (*x_cur, x_opt, UB, first, f_opt, cont, rhs,* etc.) and constants (`total_time_limit, node_time_limit, k_step`) used may be found in [31].

VNS for Continuous Global Optimization

The continuous constrained nonlinear global optimization problem (GOP) in general form is given as follows:

$$\text{(GOP)} \quad \left[\begin{array}{lll} \min & f(x) & \\ \text{s.t.} & g_i(x) \leq 0 & \forall i \in \{1, 2, \ldots, m\} \\ & h_i(x) = 0 & \forall i \in \{1, 2, \ldots, r\} \\ & a_j \leq x_j \leq b_j & \forall j \in \{1, 2, \ldots, n\} \end{array}\right.$$

where $x \in R^n$, $f : R^n \to R$, $g_i : R^n \to R$, $i = 1, 2, \ldots, m$, and $h_i : R^n \to R$, $i = 1, 2, \ldots, r$ are possibly nonlinear continuous functions, and $a, b \in R^n$ are the variable bounds.

Algorithm 17: Steps of the VNS branching

```
Function VnsBra(total_time_limit, node_time_limit, k_step, x_opt)
 1  TL := total_time_limit; UB := ∞; first := true
 2  stat := MIPSOLVE(TL, UB, first, x_opt, f_opt)
 3  x_cur:=x_opt; f_cur:=f_opt
 4  while (elapsedtime < total_time_limit) do
 5      cont := true; rhs := 1; first := false
 6      while (cont or elapsedtime < total_time_limit) do
 7          TL = min(node_time_limit, total_time_limit-elapsedtime)
 8          add local br. constr. δ(x, x_cur) ≤ rhs; UB := f_cur
 9          stat := MIPSOLVE(TL, UB, first, x_next, f_next)
10          switch stat do
11              case "opt_sol_found":
12                  reverse last local br. const. into δ(x, x_cur) ≥ rhs + 1
13                  x_cur := x_next; f_cur := f_next; rhs := 1;
14              case "feasible_sol_found":
15                  reverse last local br. constr. into δ(x, x_cur) ≥ 1
16                  x_cur := x_next; f_cur := f_next; rhs := 1;
17              case "proven_infeasible":
18                  remove last local br. constr.; rhs := rhs+1;
19              case "no_feasible_sol_found":
20                  cont := false
21      if f_cur < f_opt then
22          x_opt := x_cur; f_opt := f_cur; k_cur := k_step;
        else
23          k_cur := k_cur+k_step;
24      remove all added constraints; cont := true
25      while cont and (elapsedtime < total_time_limit) do
26          add constraints k_cur ≤ δ(x, x_opt) < k_cur + k_step
27          TL := total_time_limit-elapsedtime; UB := ∞; first := true
28          stat := MIPSOLVE(TL, UB, first, x_cur, f_cur)
29          remove last two added constraints; cont =false
30          if stat = "proven_infeasible" or "no_feasible" then
31              cont :=true; k_cur := k_cur+k_step
```

GOP naturally arises in many applications, e.g., in advanced engineering design, data analysis, financial planning, risk management, scientific modeling, etc. Most cases of practical interest are characterized by multiple local optima, and, therefore, in order to find the globally optimal solution, a global scope search effort is needed.

If the feasible set X is convex and objective function f is convex, then GOP is relatively easy to solve, i.e., the Karush-Kuhn-Tucker conditions may be applied. However, if X is not a convex set or f is not a convex function, we could have many local minima, and thus, the problem may not be solved by using classical techniques.

For solving GOP, VNS has been used in two different ways: (a) with neighborhoods induced by using an ℓ_p norm and (b) without using an ℓ_p norm.

(a) VNS with ℓ_p norm neighborhoods [3, 4, 15, 36, 39, 42]. A natural approach in applying VNS for solving GOP is to induce neighborhood structures $\mathcal{N}_k(x)$ from the ℓ_p metric:

$$\rho(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty) \tag{3}$$

or

$$\rho(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p \to \infty). \tag{4}$$

The neighborhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k-th neighborhood of x, and using the metric ρ, it is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x, y) \leq \rho_k\}, \tag{5}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} \leq \rho(x, y) \leq \rho_k\}, \tag{6}$$

where ρ_k, known as the radius of $\mathcal{N}_k(x)$, is monotonically increasing with k.

For solving box constraint GOP, both [36] and [15] use neighborhoods as defined in (6). The basic differences between the two are as follows: (1) in the procedure suggested in [36] the ℓ_∞ norm is used, while in [15] the choice of metric is either left to the analyst or changed automatically according to some predefined order; (2) as a local search procedure within VNS, the commercial solver SNOPT [20] is used in [36], while in [15] the analyst may choose one out of six different convex minimizers. A VNS-based heuristic for solving the generally constrained GOP is suggested in [39]. There, the problem is first transformed into a sequence of box-constrained problems within well-known exterior point method:

$$\min_{a \leq x \leq b} F_{\mu,q}(x) = f(x) + \frac{1}{\mu} \sum_{i=1}^{m} (\max\{0, q_i(x)\})^q + \sum_{i=1}^{r} |h_i(x)|^q, \tag{7}$$

where μ and $q \geq 1$ are a positive penalty parameter and penalty exponent, respectively. Algorithm 18 outlines the steps for solving the box constraint subproblem as proposed in [39]:

The `Glob-VNS` procedure from Algorithm 18 contains the following parameters in addition to $k_{\max}$ and $t_{\max}$:

1. *Values of radii* ρ_k, $k = 1, \ldots, k_{\max}$. Those values may be defined by the user or calculated automatically in the minimizing process;

Algorithm 18: Steps of continuous VNS using ℓ_p norm

Function `Glob-VNS` $(x^*, k_{\max}, t_{\max})$
1 Select the set of neighborhood structures $\mathcal{N}_k$ $k = 1, \ldots, k_{\max}$
2 Select the array of random distributions types and an initial point $x* \in X$
3 $x \leftarrow x^*$, $f^* \leftarrow f(x)$, $t \leftarrow 0$
4 **while** $t < t_{\max}$ **do**
5 $\quad k \leftarrow 1$
6 $\quad$ **repeat**
7 $\quad\quad$ **for** *all distribution types* **do**
8 $\quad\quad\quad y \leftarrow$ `Shake`(x^*, k) // Get $y \in \mathcal{N}_k(x^*)$ at random
9 $\quad\quad\quad y' \leftarrow$ `BestImprovment`(y) // Apply LS to obtain a local minimum y'
10 $\quad\quad\quad$ **if** $f(y') < f^*$ **then**
11 $\quad\quad\quad\quad x^* \leftarrow y'$, $f^* \leftarrow f(y')$, `go to line 5`
12 $\quad\quad k \leftarrow k + 1$
$\quad$ **until** $k = k_{\max}$
13 $\quad t \leftarrow$ `CpuTime()`

2. *Geometry* of neighborhood structures $\mathcal{N}_k$, defined by the choice of metric. Usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms;
3. *Distribution* used for obtaining the random point y from $\mathcal{N}_k$ in the *shaking* step. Uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to much better performance on some problems.

Different choices of geometric neighborhood shapes and random point distributions lead to different VNS-based heuristics.

(b) VNS without using ℓ_p norm. Two different neighborhoods, $N_1(x)$ and $N_2(x)$, are used in VNS-based heuristic suggested in [55]. In $N_1(x)$, r (a parameter) random directions from the current point x are generated and one-dimensional searches along each performed. The best point (out of r) is selected as a new starting solution for the next iteration, if it is better than the current one. If not, as in VND, the search is continued within the next neighborhood $N_2(x)$. The new point in $N_2(x)$ is obtained as follows. The current solution is moved parallel to each x_j $(j = 1, \ldots, n)$ by value Δ_j, taken at random from interval $(-\alpha, \alpha)$, i.e., $x_j^{(\text{new})} = x_j + \Delta_j$ or $x_j^{(\text{new})} = x_j - \Delta_j$. Points obtained by the plus or minus sign for each variable define neighborhood $N_2(x)$. If change of $x_j^{(\text{new})}$ by 1 % to the right gives a better solution than in $x^{(\text{new})}$, the + sign is chosen; otherwise the – sign is chosen.

Neighborhoods N_1 and N_2 are used for designing two algorithms. The first, called VND, iterates these neighborhoods until there is no improvement in the

solution value. In the second variant, a local search is performed in N_2 and $k_{\max}$ set to 2 for the shaking step. In other words, a point from neighborhood $k = 2$ is obtained by generating a random direction followed by line search along it (as prescribed for N_1) and then by changing each of the variables (as prescribed for N_2).

Variable Neighborhood Programming

Building an intelligent machine is an old dream that, thanks to computers, began to take shape. Automatic programming is an efficient technique that has contributed an important development in the field of artificial intelligence. Genetic programming (GP) [35], inspired by genetic algorithm (GA), is among the few evolutionary algorithms used to evolve population of programs. The main difference between GP and GA is the presentation of a solution. An individual in GA can be a string, while GPs individuals are programs. The usual way to present program within GP is by using a tree. For example, assume that the current solution of the problem is the following function:

$$f(x_1, \ldots, x_5) = \frac{x_1}{x_2 + x_3} + x_4 - x_5.$$

Then the code (tree) that calculates it using GP may be presented as in Fig. 5a.

Souhir et al. [16] recently adapted VNS rules in solving automatic programming problem. They first suggested an extended solution presentation by adding coefficients to variables. Each terminal node was attached to its own parameter value. These parameters gave a weight for each terminal node, with values from the interval [0, 1]. This form of presentation allowed VNP to examine parameter values and the

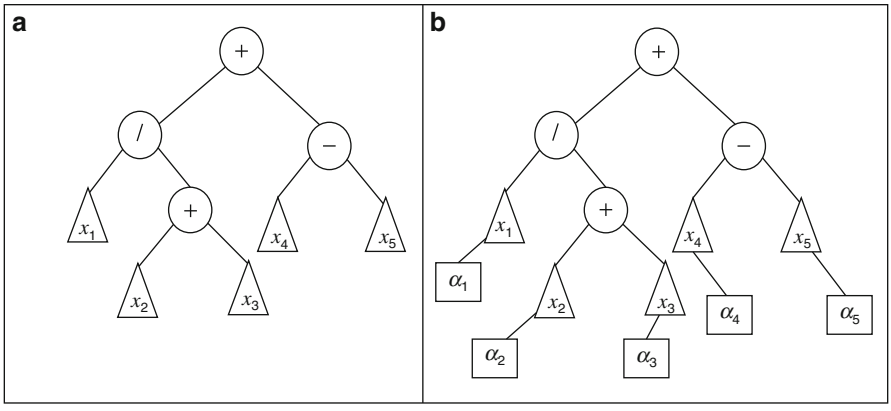


Fig. 5 Current solution representation in automatic programming problem: $\frac{x_1}{x_2+x_3} + x_4 - x_5$. (**a**) GP solution representation. (**b**) VNP solution representation

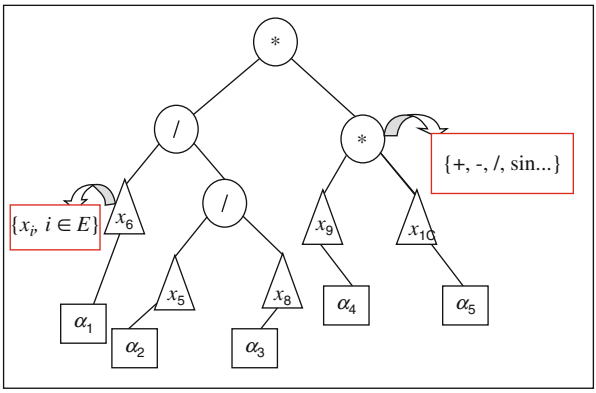


Fig. 6 Neighborhood structure N_1: changing a node value

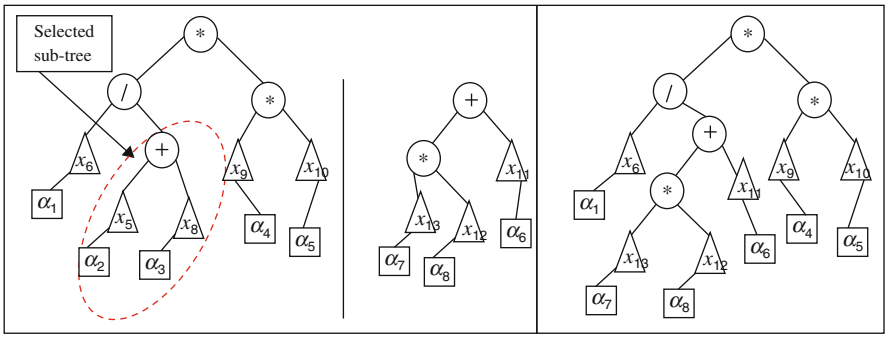


Fig. 7 Neighborhood structure N_2: swap operator

remaining tree structures in the same iteration. Let $G = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ denote a parameter set. In Fig. 5b an example of VNP's solution representation is illustrated.

Neighborhood structures. At such solution presentation as a tree T, nine different neighborhood structures are proposed in [16]. To save the space, we will just mention some of them:

- $N_1(T)$ – **Changing a node value operator.** This neighborhood structure conserves the skeleton of the tree and changes only the values of a functional or a terminal node. Each node can obtain many values from its corresponding set. Let x_i be the current solution; its neighbor x_{i+1} differs from x_i by just a single node. A move within this neighborhood structure is shown in Fig. 6.
- $N_2(T)$ – **Swap operator**. By this operator a first node from the current tree is taken at random, and a new sub-tree is generated as presented in Fig. 7a, b. Then the selected node is attached in the place of the sub-tree. In this move, the constraint related to the maximum tree size should be respected. More details can be seen in Fig. 7.

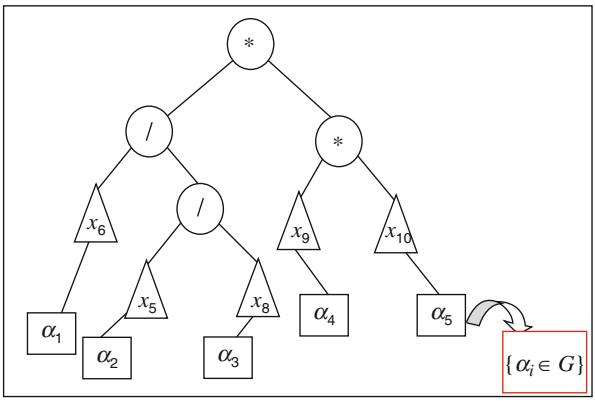


Fig. 8 Neighborhood structure N_3: change parameters

- $N_3(T)$ – **Changing parameter values.** In the previous neighborhood structures, the tree form and its node values were considered. In $N_3(T)$ neighborhood, attention is paid on parameters. So, the position and value of nodes are kept in order to search the neighbors in the parametric space. Figure 8 illustrates details. The change from one value to another is at random.

These neighborhood structures may be used in both local search step ($N_\ell, \ell \in [1, \ell_{\max}]$) and in the shaking step ($\mathcal{N}_k, k \in [1, k_{\max}]$) of the VNS.

VNP shaking. The shaking step allows the diversification in the search space. Our proposed VNP algorithm does not use exactly the same neighborhood structures N_ℓ as for the local search. That is the reason why neighborhoods used in shaking are denoted differently: $\mathcal{N}_k(T), k = 1, k_{\max}$. $\mathcal{N}_k(T)$ may be constructed by repeating k times one or more moves from the set $\{N_\ell(T), |\ell = 1, \ldots, \ell_{\max}\}$, explained earlier. Nevertheless, in the shaking phase, we use a neighborhood structure that mainly affects the skeleton of a presented solution with its different forms for the perturbation. For more clear explanation, we take the swap operator $N_3(T)$ as an example. Let m denote the maximum size of a solution. We can get a solution from the kth neighborhood of T using the swap operator: k may represent the size of the new generated sub-tree. If n denotes the size of the original tree after deleting the old sub-tree, then $n + k_{\max} \leq m$. The objective of this phase is to provide a good starting point for the local search.

VNP objective function. The evaluation consists of defining a fitness or objective function assessing the proposed problem. This function is defined according to the problem considered. After running each solution (program) on training data set, fitness may be measured by counting how many training cases the current solution result is correct or near to the exact solution.

An example: time series forecasting (TSF) problem. Two widely used benchmark data sets of TSF problem are explored in [16] to examine VNP capabilities: Mackey-Glass series and Box-Jenkins set. The parameters for the VNP implementation are chosen after some preliminary testing are given in Table 3.

Table 3 VNP parameters adjustment for the forecasting problem

Parameters	Values
The functional set	$F = \{+, *, , pow\}$;
The terminal sets	$\{(x_i, c), i \in [1, \ldots, m], m =$ number of inputs, $c \in R\}$;
Neighborhood structures	$\{N_1, N_2, N_3\}$;
Minimum tree length	20 nodes;
Maximum tree length	200 nodes;
Maximum number of iterations	50,000

Table 4 Comparison of testing error on Box-Jenkins dataset

Method	Prediction error RMSE
ODE [54]	0.5132
HHMDDE [14]	0.3745
FBBFNT [6]	0.0047
VNP [16]	**0.0038**

The root-mean-square error (RMSE) is used as fitness function, as it is usual in the literature:

$$f(T) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_t^j - y_{\text{out}}^j)^2}$$

where n represents the total number of samples and y_{out}^j and y_t^j are outputs of the sample number j obtained by the VNP model and the desired one, respectively. Here we will just illustrate comparison on Box-Jenkins instance.

The gas furnace data of Box and Jenkins were collected from a combustion process of a methane-air mixture [15]. This time series has found a widespread application as a benchmark example in many practical sciences for testing prediction algorithms. The data set contains 296 pairs of input-output values. The input $u(t)$ corresponds to the gas flow, and the output $y(t)$ presents the CO_2 concentration in outlet gas. The inputs are $u(t-4)$ and $y(t-1)$, and the output is $y(t)$. In this work, 200 samples are used in the training phase, and the remaining samples are used for the testing phase. The performance of the evolved model is evaluated by comparing it with the abovementioned approaches. The RMSE achieved by VNP output model is 0.0038, which appeared to be better than the RMSE obtained by other approaches. Table 4 shows that VNP approach proves effectiveness and generalization ability.

Discovery Science

In all the above applications, VNS is used as an optimization tool. It can also lead to results in "discovery science," i.e., help in the development of theories. This has been done for graph theory in a long series of papers with the common title "Variable

neighborhood search for extremal graphs" and reporting on the development and applications of the system AutoGraphiX (AGX) [10]. This system addresses the following problems:

- Find a graph satisfying given constraints;
- Find optimal or near optimal graphs for an invariant subject to constraints;
- Refute a conjecture;
- Suggest a conjecture (or repair or sharpen one);
- Provide a proof (in simple cases) or suggest an idea of proof.

Then a basic idea is to consider all of these problems as parametric combinatorial optimization problems on the infinite set of all graphs (or in practice some smaller subset) with a generic heuristic. This is done by applying VNS to find extremal graphs, with a given number n of vertices (and possibly also a given number of edges). Then a VND with many neighborhoods is used. Those neighborhoods are defined by modifications of the graphs such as the removal or addition of an edge, rotation of an edge, and so forth. Recently all moves involving four vertices or less were considered jointly in learning optimization framework. Once a set of extremal or near-extremal graphs, parameterized by their order, is found, their properties are explored with various data mining techniques, leading to conjectures, refutations, and simple proofs or ideas of proof.

All papers in this area are divided into the following groups in [28], where the extensive list of references for each class of problems can be found:

(i) *Principles of the approach* and its implementation;
(ii) *Applications to spectral graph theory*, e.g., finding bounds on the index or spectral radius of the adjacency matrix for various families of graphs and finding graphs maximizing or minimizing the index subject to some conditions;
(iii) *Studies of classical graph parameters*, e.g., independence, chromatic number, clique number, and average distance;
(iv) *Studies of little known or new parameters of graphs*, e.g., irregularity, proximity, and remoteness;
(v) *New families of graphs discovered by AGX*, e.g., bags, which are obtained from complete graphs by replacing an edge by a path, and bugs, which are obtained by cutting the paths of a bag;
(vi) *Applications to mathematical chemistry*, e.g., study of chemical graph energy and of the Randić index;
(vii) *Results of a systematic study of pairwise comparison of 20 graph invariants*, involving the four usual operators: plus, minus, ratio, and product, which led to almost 1500 new conjectures, more than half of which were automatically proved by AGX and over 300 by various mathematicians;
(viii) *Refutation or strengthening of conjectures from the literature*;
(ix) *Surveys and discussions about various discovery systems in graph theory*, assessment of the state-of-the-art and the forms of interesting conjectures together with proposals for the design of more powerful systems.

Conclusions

The general schemes of variable neighborhood search have been presented, discussed, and illustrated by examples. In order to evaluate the VNS research program, one needs a list of the desirable properties of Metaheuristics. The following eight of these are presented in Hansen and Mladenović [27]:

(i) *Simplicity:* the metaheuristic should be based on a simple and clear principle, which should be widely applicable;
(ii) *Precision:* the steps of the metaheuristic should be formulated in precise mathematical terms, independent of possible physical or biological analogies which may have been the initial source of inspiration;
(iii) *Coherence:* all steps of the heuristics for particular problems should follow naturally from the principle of the metaheuristic;
(iv) *Efficiency:* heuristics for particular problems should provide optimal or near-optimal solutions for all or at least most realistic instances. Preferably, they should find optimal solutions for most problems of benchmarks for which such solutions are known, when available;
(v) *Effectiveness:* heuristics for particular problems should take a moderate computing time to provide optimal or near-optimal solutions;
(vi) *Robustness:* the performance of heuristics should be consistent over a variety of instances, i.e., not merely fine-tuned to some training set and less good elsewhere;
(vii) *User-friendliness:* heuristics should be clearly expressed, easy to understand, and, most important, easy to use. This implies they should have as few parameters as possible, ideally none;
(viii) *Innovation:* preferably, the principle of the metaheuristic and/or the efficiency and effectiveness of the heuristics derived from it should lead to new types of application.
(ix) *Generality:* the metaheuristic should lead to good results for a wide variety of problems;
(x) *Interactivity:* the metaheuristic should allow the user to incorporate his knowledge to improve the resolution process;
(xi) *Multiplicity:* the metaheuristic should be able to present several near-optimal solutions from which the user can choose one.

As shown above, VNS possesses, to a great extent, all of the above properties. This has led to heuristics which are among the very best ones for many problems. Interest in VNS is clearly growing at speed. This is evidenced by the increasing number of papers published each year on this topic (15 years ago, only a few; 10 years ago, about a dozen; about 50 in 2007). According to Google Scholar, the first two papers on VNS were cited almost 4,000 times!

Moreover, the 18th and 28th EURO Mini Conferences were entirely devoted to VNS. It led to special issues of *IMA Journal of Management Mathematics* in 2007

[37], *European Journal of Operational Research* [29], *Journal of heuristics* [47] in 2008, *Computers and Operations Research* in 2014 [45], and *Journal of Global Optimization* in 2016 [11]. The 3rd International VNS meeting took place in Tunisia, and three new special issues entirely devoted to VNS are in preparation (*Computers and Operations Research*, *Optimization Letters*, and *Yugoslav Journal of Operations Research*). In retrospect, it appears that the good shape of the VNS research program is due to the following decisions, strongly influenced by Karl Popper's philosophy of science [51]: (i) in devising heuristics favor insight over efficiency (which comes later) and (ii) learn from the heuristics mistakes.

References

1. Aloise DJ, Aloise D, Rocha CTM, Ribeiro CC, Ribeiro JC, Moura LSS (2006) Scheduling workover rigs for onshore oil production. Discret Appl Math 154(5):695–702
2. Andrade DV, Resende MGC (2007) GRASP with path-relinking for network migration scheduling. In: Proceedings of international network optimization conference (INOC), Spa
3. Audet C, Báchard V, Le Digabel S (2008) Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search J Glob Optim 41(2):299–318
4. Audet C, Brimberg J, Hansen P, Mladenović N (2004) Pooling problem: alternate formulation and solution methods, Manag Sci 50:761–776
5. Belacel N, Hansen P, Mladenović N (2002) Fuzzy J-means: a new heuristic for fuzzy clustering. Pattern Recognit 35(10):2193–2200
6. Bouaziz S, Dhahri H, Alimi AM, Abraham A (2013) A hybrid learning algorithm for evolving flexible beta basis function neural tree model. Neurocomputing 117: 107–117. doi:10.1016/j.neucom.2013.01.024
7. Brimberg J, Hansen P, Mladenović N, Taillard É (2000) Improvements and comparison of heuristics for solving the multisource Weber problem. Oper Res 48(3):444–460
8. Brimberg J, Mladenović N (1996) A variable neighborhood algorithm for solving the continuous location-allocation problem. Stud Locat Anal 10:1–12
9. Canuto S, Resende M, Ribeiro C (2001) Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks 31(3):201–206
10. Caporossi G, Hansen P (2000) Variable neighborhood search for extremal graphs 1. The AutoGraphiX system. Discret Math 212:29–44
11. Carrizosa E, Hansen P, Moreno-Perez JA (2015). Variable neighborhood search. J Glob Optim (Spec Issue) 63(3):427–629
12. Cohoon J, Sahni S (1987) heuristics for backplane ordering. J VLSI Comput Syst 2:37–61
13. Davidon WC (1959) Variable metric algorithm for minimization. Argonne National Laboratory report ANL-5990
14. Dhahri H, Alimi AM, Abraham A (2012) Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network. Neurocomputing 97:131–140
15. Dražić M, Kovacevic-Vujcić V, Cangalović M, Mladenović N (2006) GLOB – a new VNS-based software for global optimization In: Liberti L, Maculan N (eds) Global optimization: from theory to implementation. Springer, New York, pp 135–144
16. Elleucha S, Jarbouia B, Mladenovic N (2015) Variable neighborhood programming a new automatic programming method in artificial intelligence, Gerad Technical report G-2016-21, HEC Montreal, Canada
17. Fischetti M, Lodi A (2003) Local branching. Math Program 98(1–3):23–47
18. Fletcher R, Powell MJD (1963) Rapidly convergent descent method for minimization. Comput J 6:163–168

19. Garey MR, Johnson DS (1978) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New-York
20. Gill P, Murray W, Saunders MA (2002) SNOPT: an SQP algorithms for largescale constrained optimization. SIAM J Optim 12(4):979–1006
21. Griffith RE, Stewart RA (1961) A nonlinear programming technique for the optimization of continuous processing systems. Manag Sci 7:379–392
22. Hansen P, Brimberg J, Urošević D, Mladenović N (2007) Primal-dual variable neighborhood search for the simple plant location problem. INFORMS J Comput 19(4):552–564
23. Hansen P, Jaumard B, Mladenović N, Parreira A (2000) Variable neighborhood search for weighted maximum satisfiability problem. Les Cahiers du GERAD G–2000–62, HEC Montréal
24. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130:449–467
25. Hansen P, Mladenović N (2001) J-means: a new local search heuristic for minimum sum-of-squares clustering. Pattern Recognit 34:405–413
26. Hansen P, Mladenović N (2001) Developments of variable neighborhood search. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer, Dordrecht/London, pp 415–440
27. Hansen P, Mladenović N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 145–184
28. Hansen P, Mladenović N, Brimberg J, Moreno-Perrez JA (2010) Variable neighborhood search. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, 2nd edn. Kluwer, New York, pp 61–86
29. Hansen P, Mladenović N, Moreno Pérez JA (2008) Variable neighborhood search. Eur J Oper Res 191(3):593–595
30. Hansen P, Mladenović N, Pérez-Brito D (2001) Variable neighborhood decomposition search. J Heuristics 7(4):335–350
31. Hansen P, Mladenović N, Urošević D (2006) Variable neighborhood search and local branching. Comput Oper Res 33(10):3034–3045
32. Hertz A, Plumettaz M, Zufferey N (2008) Variable space search for graph coloring. Discret Appl Math 156(13):2551–2560
33. ILOG (2006) CPLEX 10.1. User's manual
34. Jornsten K, Lokketangen A (1997) Tabu search for weighted k-cardinality trees. Asia-Pac J Oper Res 14(2):9–26
35. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
36. Liberti L, Dražić M (2005) Variable neighbourhood search for the global optimization of constrained NLPs. In: Proceedings of GO workshop, Almeria
37. Melián B, Mladenović N (2007) Editorial IMA J Manag Math 18(2):99–100
38. Mladenovic N (1995) Variable neighborhood algorithm – a new metaheuristic for combinatorial optimization. In: Optimization days conference, Montreal, p 112
39. Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization Eur J Oper Res 191(3):753–770
40. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24: 1097–1100
41. Mladenovic N, Kratica J, Kovacevic-Vujcic V, Cangalovic M (2012) Variable neighborhood search for metric dimension and minimal doubly resolving set problems. Eur J Oper Res 220(2):328–337
42. Mladenović N, Petrović J, Kovačević-Vujčić V, Čangalović M (2003) Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. Eur J Oper Res 151:389–399
43. Mladenović N, Plastria F, Urošević D (2005) Reformulation descent applied to circle packing problems. Comput Oper Res 32:2419–2434
44. Mladenović N, Plastria F, Uroševic D (2007) Formulation space search for circle packing problems. Engineering Stochastic local search algorithms. Designing, implementing and

analyzing effective heuristics. Lecture notes in computer science, vol 4638, pp 212–216. https://link.springer.com/book/10.1007/978-3-540-74446-7
45. Mladenovic N, Salhi S, Hnafi S, Brimberg J (eds) (2014) Recent advances in variable neighborhood search. Comput Oper Res 52(B):147–148
46. Mladenovic N, Urosevic D, Pérez-Brito D, García-González CG (2010) Variable neighbourhood search for bandwidth reduction. Eur J Oper Res 200(1):14–27
47. Moreno-Vega JM, Melián B (2008) Introduction to the special issue on variable neighborhood search. J Heuristics 14(5):403–404
48. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199:285–304
49. Pardo EG, Mladenovic N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252 (2013)
50. Plastria F, Mladenović N, Urošević D (2005) Variable neighborhood formulation space search for circle packing. In: 18th mini Euro conference VNS, Tenerife
51. Popper K (1959) The logic of scientific discovery Hutchinson, London
52. Ribeiro CC, de Souza MC (2002) Variable neighborhood search for the degree-constrained minimum spanning tree problem. Discret Appl Math 118(1–2):43–54
53. Ribeiro CC, Uchoa E, Werneck R (2002) A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J Comput 14(3):228–246
54. Subudhi B, Jena D (2011) A differential evolution based neural network approach to nonlinear system identification, Appl Soft Comput 11:861–871. doi:10.1016/j.asoc.2010.01.006
55. Toksari AD, Güner E (2007) Solving the unconstrained optimization problem by a variable neighborhood search. J Math Anal Appl 328(2):1178–1187
56. Whitaker R (1983) A fast algorithm for the greedy interchange of large-scale clustering and median location problems INFOR 21:95–108

Part IV
Analysis and Implementation

A History of Metaheuristics

27

Kenneth Sörensen, Marc Sevaux, and Fred Glover

Contents

Introduction . . . 792
Period 0: The Pre-theoretical Period . . . 795
Period 1: The Early Period . . . 797
Period 2: The Method-Centric Period . . . 798
Period 3: The Framework-Centric Period . . . 801
The Metaphor-Centric Period . . . 803
Period 4: The Scientific Period? . . . 804
Conclusions . . . 805
References . . . 806

Abstract

This chapter describes the history of metaheuristics in five distinct periods, starting long before the first use of the term and ending a long time in the future.

The field of metaheuristics has undergone several paradigm shifts that have changed the way researchers look upon the development of heuristic methods.

K. Sörensen (✉)
University of Antwerp, Antwerp, Belgium
e-mail: kenneth.sorensen@uantwerpen.be

M. Sevaux
Université de Bretagne-Sud, Lab-STICC, CNRS, Lorient, France
e-mail: marc.sevaux@univ-ubs.fr

F. Glover
OptTek Systems, Inc, Boulder, CO, USA
e-mail: glover@opttek.com

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_4

Most notably, there has been a shift from the *method-centric* period, in which metaheuristics were thought of as algorithms, to the *framework-centric* period, in which researchers think of metaheuristics as more general high-level frameworks, i.e., consistent collections of concepts and ideas that offer guidelines on how to go about solving an optimization problem heuristically.

Tremendous progress has been made in the development of heuristics over the years. Optimization problems that seemed intractable only a few decades ago can now be efficiently solved. Nevertheless, there is still much room for evolution in the research field, an evolution that will allow it to move into the *scientific period*. In this period, we will see more structured knowledge generation that will benefit both researchers and practitioners.

Unfortunately, a significant fraction of the research community has deluded itself into thinking that scientific progress can be made by resorting to ever more outlandish metaphors as the basis for so-called "novel" methods. Even though considerable damage to the research field will have been inflicted by the time these ideas have been stamped out, there is no doubt that science will ultimately prevail.

Keywords
History

Introduction

Even though people have used heuristics throughout history, and the human brain is equipped with a formidable heuristic engine to solve an enormous array of challenging optimization problems, the *scientific study* of heuristics (and, by extension, metaheuristics) is a relatively young endeavor. It is not an exaggeration to claim that the field of (meta) heuristics, especially compared to other fields of study like physics, chemistry, and mathematics, has yet to reach a mature state. Nevertheless, enormous progress has been made since the first metaheuristics concepts were established. In this chapter, we will attempt to describe the historical developments this field of study has gone through since its earliest days.

No history is ever neutral, and a history of metaheuristics – or any other topic – can be written in many different ways. A straightforward (one could say "easy") history of metaheuristics would consist of an annotated and chronological list of metaheuristic methods. Useful as such a list may be, it suffers from a lack of insight into the development of the field as a whole. To illustrate this viewpoint, consider the list in Fig. 1 that appeared on Wikipedia until April 8, 2013, to illustrate the "most important contributions" in the field of metaheuristics. It is our opinion that such a list is not particularly enlightening (and neither was the article that contained it) when it comes to explaining the evolution of the field of metaheuristics.

Taking a bird's eye view of the field of metaheuristics, one has to conclude that there has been a large amount of progressive insight over the years. Moreover,

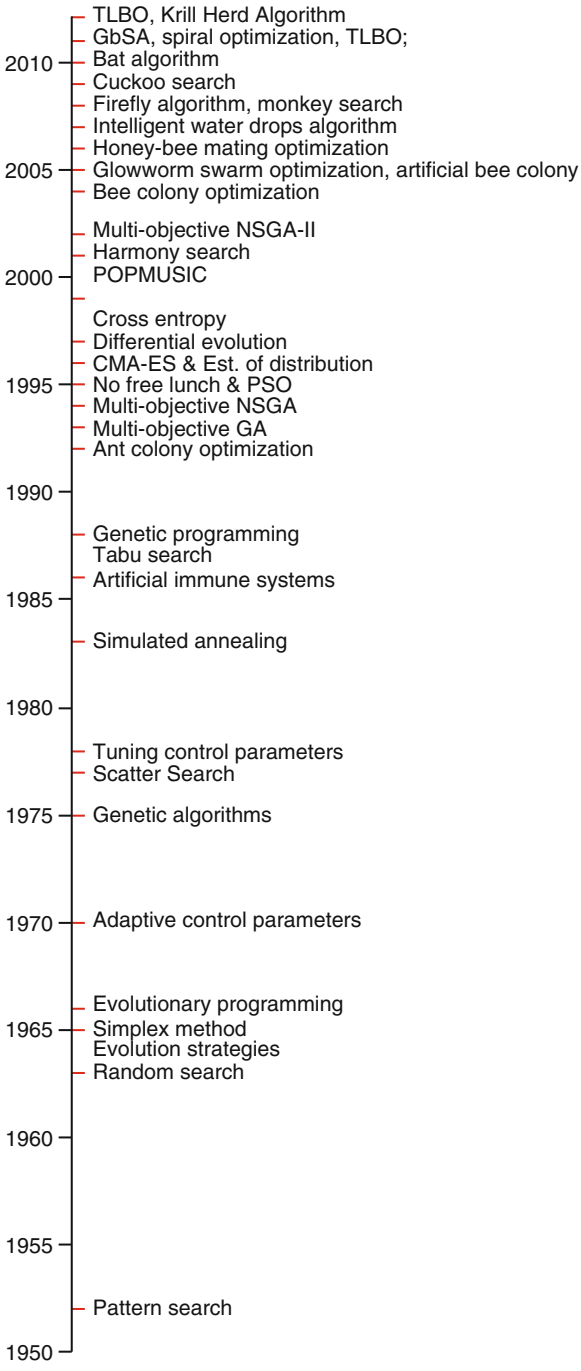


Fig. 1 "Most important contributions" list as it appeared on Wikipedia until April 8, 2013

this progressive insight has not reached its end point: the way researchers and practitioners look at metaheuristics is still continually shifting. Even the answer to the question what a metaheuristic is has changed quite a lot since the word was first coined in the second half of the 1980s. In our view, it is this shifting viewpoint that deserves to be written down, as it allows us to truly understand the past and perhaps learn a few lessons that could be useful for the future development of research in metaheuristics. We did not limit our discussion in this chapter to metaheuristics that have been formally written down and published. When studying the history of metaheuristics with an open mind, one has to conclude that people have been using heuristics and metaheuristics long before the term even existed.

We have therefore adopted a different approach to write "our" history of metaheuristics. Our approach starts well before the term "metaheuristic" was coined and is based on the premise that people have looked at metaheuristics through different sets of glasses over the years. The way in which people – not only researchers – have interpreted the different metaheuristic concepts has shaped the way in which the field has been developing. To understand the design choices that people have been making when developing metaheuristic optimization algorithms, it is paramount that these choices are understood in relationship to the trends and viewpoints of the time during which the development took place.

Our history divides time in five distinct periods. The crispness of the boundaries between each pair of consecutive periods, however, is a gross simplification of reality. The real (if one can use that word) time periods during which the paradigm shifts took place are usually spaced out over several years, but it is difficult, if not impossible, to trace the exact moments in time at which the paradigm shifts began and ended. More importantly, not every researcher necessarily makes the transition at the same time.

- The *pre-theoretical* period (until c. 1940), during which heuristics and even metaheuristics are used but not formally studied.
- The *early* period (c. 1940–c. 1980), during which the first formal studies on heuristics appear.
- The *method-centric* period (c. 1980–c. 2000), during which the field of metaheuristics truly takes off and many different methods are proposed.
- The *framework-centric* period (c. 2000–now), during which the insight grows that metaheuristics are more usefully described as frameworks and not as methods.
- The *scientific* period (the future), during which the design of metaheuristics becomes a science instead of an art.

Until recently, a clear definition of the word metaheuristic has been lacking, and it could be argued that it is still disputed. In this chapter, we adopt the definition of Sörensen and Glover [39].

> A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.

The term "metaheuristic" has been used (and is used) for two entirely different things. One is a high-level framework, a set of concepts and strategies that blend together and offer a perspective on the development of optimization algorithms. In this sense, variable neighborhood search [28] is nothing more (or less) than the idea to use different local search operators to work on a single solution, together with a perturbation operator once all neighborhoods have reached a local optimum. There is a compelling motivation, as well a large amount of empirical evidence, as to why multi-neighborhood search is indeed a very good idea. This motivation essentially comes down to the fact that a local optimum for one local search operator (or one neighborhood structure) is usually not a local optimum for another local search operator. The idea to switch to a different local search operator once a local optimum has been found is therefore both sensible and in practice extremely powerful.

The second meaning of the term "metaheuristic" denotes a specific implementation of an algorithm based on such a framework (or on a combination of concepts from different frameworks) designed to find a solution to a specific optimization problem. The variable neighborhood search (-based) algorithm for the location-routing problem by Jarboui et al. [23] is an example of a metaheuristic in this sense.

In this chapter, we will use the term "metaheuristic framework" to refer to the first sense and "metaheuristic algorithm" to refer to the second sense of the word "metaheuristic."

As mentioned, a history of any topic is not a neutral. We therefore do not attempt to hide the fact that certain ways in which the field has been progressing seem to us less useful and sometimes even harmful to the development of the field in general. For example, many of the entries that appear on the list in Fig. 1 are, in our view, not "important contributions" at all but rather marginal additions to a list of generally useless "novel" metaphor-based methods that are best forgotten as quickly as possible.

Period 0: The Pre-theoretical Period

Optimization problems are all around us. When we decide upon the road to take to work, when we put the groceries in the fridge, and when we decide which investments to make so as to maximize our expected profit, we are essentially solving an optimization problem (a shortest path problem, a packing problem, and a knapsack problem, respectively). For human beings (and many animal species), solving an optimization problem does not require any formal training, something which is immediately clear from the examples given here. The difference between exact solutions and approximate solutions and the difference between easy and hard optimization problems or between fast (polynomial) and slow (exponential) algorithms are all moot to the average problem-solver.

Indeed, the human mind seems to be formidably equipped from early childhood on to solve an incredible range of problems, many of which could be easily modeled as optimization problems. Most likely, the ability to solve optimization problems adequately and quickly is one of the most important determinants of the probability

of survival in all sentient species and has therefore been favored by evolution throughout time. Clearly, the human (and animal) mind solves optimization problems *heuristically* and not exactly, i.e., the solutions produced by the brain are by no means guaranteed to be optimal. Given what we now know about exact solution procedures, this makes perfect sense. When determining the trajectory of a spear to hit a mammoth, it is much more important that this trajectory be calculated quickly rather than optimally. Given our knowledge on exact solution methods, we can now say that the calculation of the exact solution (let us say the solution that has the highest probability to hit the mammoth exactly between the eyes given its current trajectory, the terrain in front of it, its anticipated trajectory changes, the current wind direction, etc.) would almost certainly be found only *after* our target has disappeared on the horizon. Moreover, it would almost certainly require too much computing power from the brain, quickly depleting the body's scarce energy resources.

Given the diversity of problems the human mind must solve, including problems with which it has no prior experience, there is very little doubt that the human mind has the capacity (whether evolved or learned) to use *meta*-heuristic strategies. Just like the metaheuristics for optimization that form the subject of this book, such strategies are not heuristics in themselves but are used to derive heuristics from. For example, when confronted with a new problem to which a solution is not immediately obvious (e.g., determining the trajectory of a spear to hit a mammoth), the human mind will automatically attempt to find similar problems it has solved in the past (e.g., determining the trajectory of a stone to hit a bear) and attempt to derive the rules it has learned by solving this problem. This strategy is called *learning by analogy* [3]. Another example is called means-end analysis [36] and can be summarized as follows: given a current state and a goal state, choose an action that will lead to a new state that is closer to the goal state than the current state. This rule is iteratively applied until the goal state has been reached or no other state can be found closer to the goal state than the current state. Obviously, this strategy is a more general counterpart of all formal optimization heuristics that can be categorized as local search, in which a solution is iteratively improved using small, incremental operations we have come to call *moves*. The technique of *path relinking* [15], in which an incumbent solution is transformed, one move at a time, into a guiding solution, is another example of a formalized means-end analysis strategy.

Whereas heuristics (and even metaheuristics) are completely natural to us humans, exact methods seem to be a very recent invention, coinciding with the introduction of the field of Operations Research around WWII. On the other hand, even though heuristics have been applied since the first life on earth evolved, the *scientific study* of heuristics also had to wait until the twentieth century. It could be hypothesized that heuristics are so natural to us that we had to wait until a formal theory of optimization, especially of linear programming, had to be developed before anyone considered it a topic worthy of study.

Period 1: The Early Period

In 1945, immediately after WWII, the Hungarian mathematician George Pólya, then working at Stanford University, published a small volume called "How to Solve It" [30]. In his book, he argued that problems can be solved by a limited set of generally applicable strategies, most of which serve to make the problem simpler to solve. The book's focus was not on optimization problems but on the more general class of "mathematical" problems, i.e., problems that can be modeled and solved by mathematical techniques. Nevertheless, most of the solution strategies proposed in his book are equally applicable to develop optimization algorithms.

The "analogy" principle, e.g., tells the problem-solver to look for another problem that closely resembles the problem at hand and to which a solution method is known. By studying the similarities and differences between both problems, ideas can be garnered to solve the original problem. The principle of "induction" consists in solving a problem by deriving a generalization from some examples. The "auxiliary problem" idea asks whether a subproblem exists that can help to solve the overall problem. Even though it is a bit of stretch to call these principles "*meta*-heuristics," it is clear that the start of the field of OR also marks the age during which people start thinking about more general principles that are useful in the design of heuristic algorithms (or solution methods for other types of problems). A case can be made for the fact that many of Pólya's principle are still heavily used today by heuristic designers. Looking for similar problems in the literature or elsewhere, and modifying the best-known methods for them to suit the problem at hand (analogy), is an extremely common strategy to arrive at a good heuristic fast. Solving some simple examples by hand, and using the lessons learned from your own (or someone else's) perceived strategy to derive an intelligent solution strategy from (induction), is also a useful technique. Finally, decomposing a problem into smaller subproblems and developing specialized techniques for each of them (auxiliary problem) has proven to be a powerful heuristic design strategy on a large number of occasions.

What is important is that none of Pólya's strategies actually solve any problem, nor can they be called "algorithms" in themselves. Instead, they are high-level, meta-strategies that are useful to influence the way a heuristic designer thinks about a problem. In that sense at the very least, they are very like the more advanced and specialized metaheuristic frameworks that we have today.

Several very high-level algorithmic ideas also came about around this period. The fact that good solutions can be reached by a constructive procedure, for example, is one of them. A constructive algorithm is one that starts from an empty solution and iteratively adds one element at a time until a complete solution has been formed. Simple rules for selecting this element from the set of all potential elements have led to different types of algorithms. The *greedy* selection rule selects the best (value for each) element at each iteration. Kruskal's or Prim's algorithm for the minimum spanning tree problem, Dijkstra's algorithm for the shortest path problem, etc., are all examples of greedy heuristics [7]. *Regret* algorithms present a similar class of

optimization procedures that select, at each iteration, the element for which *not* selecting its best value results in the highest penalty cost. Vogel's approximation method [35] for the transportation problem is a well-known example. Again, calling the greedy idea or the regret idea "metaheuristics" is a bit of a stretch, but they *are* high-level strategies, and they are *not* algorithms themselves.

Also during this period, Simon and Newell [37] see heuristics specifically as fit to solve what they call "ill-structured" problems. Contrary to well-structured problems, such problems cannot be formulated explicitly or solved by known and feasible computational techniques. Their predictions in 1958 have turned out to be slightly optimistic, but it cannot be denied that heuristics have turned out to be more flexible problem-solving strategies than exact methods.

Even though the heuristics developed in the early period were very simple, the realization that high-level strategies existed that could be used as the basis for the development of heuristics for *any* optimization problem led to insights that paved the way for more complex meta-strategies. Together with the widespread availability of computers, these developments took the field of heuristics into the next period in this history, the method-centric period.

Period 2: The Method-Centric Period

Even though the frameworks and ideas developed during what we have called the early period lacked the comprehensiveness of the later developed metaheuristic frameworks like tabu search [14], it is not too far-fetched to call them early metaheuristics. Like later metaheuristic frameworks, these methods offered – in the form of some generally applicable strategies – inspiration for the development of optimization algorithms. Of course, these principles still needed to be instantiated for each different optimization problem, but at least the process of coming up with an optimization strategy did not have to start from scratch.

Much of the work done in the early period can be characterized under the umbrella term of *artificial intelligence* because it involves mimicking human problem-solving behavior and learning lessons from this behavior on a more abstract level. Starting in the 1960s, however, an entirely different line of research into problem-solving methods came to life. These methods used an analogy with life's main problem-solving method: evolution.

Evolution by natural selection has been called "the best idea ever" [4]. No single idea explains as much as Darwin's realization that species evolve over time to adapt to their environment. The way in which this happens, by natural selection of inheritable characteristics, is both so clever and so simple it begs the question why the world needed to wait until the second half of the nineteen century before someone thought of it. Nevertheless, it took another century and the advent of the computer before researchers would become interested in simulating the process of natural evolution.

Although researchers in the late 1950s and early 1960s had developed what we would now label as evolutionary algorithms, their main aim was not to solve

optimization problems but to study the phenomenon of natural evolution. The insight that the principles of natural evolution could be used to solve optimization problems in general came in the early 1960s, when researchers like Box, Friedman, and several others had independently developed algorithms inspired by evolution for function optimization and machine learning. One of the first methods to receive some share of recognition was the so-called evolution strategy (as later reported in [31]). Evolution strategy was still quite far from what we would call an evolutionary algorithm: it did not use a population or crossover. One solution (called the parent) was mutated and the best of the two solutions became the parent for the next round of mutation.

Evolutionary programming, introduced a few years later [11], represented solutions as finite-state machines but also lacked the concepts of both a population and crossover. The true start of the field of evolutionary algorithms came with the seminal work of John Holland [20], who was the first to recognize the importance of both concepts. With his schemata theory, which essentially states that high-quality schemata ("parts") of solutions will increase in frequency in successive iterations of the algorithm, Holland was also among the pioneers of theory-building in metaheuristics. The schemata theory was criticized later for its limited use and lack of general applicability, but it demonstrated that the field of metaheuristics needed not forever be devoid of theoretical underpinning.

It was perhaps the book by David Goldberg [18] (a student of John Holland) that truly sparked the evolutionary revolution. Evolutionary methods became extremely popular, journals and conferences specifically devoted to this topic sprouted, and an exponentially increasing number of papers appeared in the literature. A large number of variants were proposed, each with its own specific characteristics. Extraordinary claims were made, not necessarily grounded in empirical evidence. The quest for a generic heuristic optimization method that could solve any problem efficiently, without requiring problem-specific information, seemed finally to be on the right track.

In the 1980s, the first papers start to appear that introduced general problem-solving frameworks not based on natural evolution. One of the first used another metaphor: annealing, the controlled heating and cooling process used in metallurgy and glass production to remove stresses from the material [24]. Simulated annealing used random solution changes and "accepted" these if they improved the solution or, if they did not, with a probability inversely proportional to the solution quality decrease and proportional to an external parameter called the "temperature."

For a while, it might have seemed that the development of metaheuristics was all about finding a suitable process to imitate. The 1980s, however, also saw the development of several methods that reached back to the early period and used ideas derived from human problem-solving. One of the most powerful ideas was that solutions could be gradually improved by iteratively making small changes, called *moves*, to them. To this end, an algorithm would investigate all or some of the solutions that could be reached from the *current* solution by executing a single move. Together, these solutions form the *neighborhood* of the current solution.

Threshold accepting [9], a simple variant of simulated annealing demonstrated that a metaphor was certainly not necessary to develop a powerful general-purpose

optimization framework. The great deluge method and record-to-record travel [8] differed from threshold accepting only by the way in which they accepted new solutions. Still, each of these was seen as a different method.

Perhaps the most influential of the AI-based methods was tabu search [14]. The basic premise of this framework is that a local search algorithm could be guided toward a good solution by using some of the information gathered during the search in the past. To this end, the tabu search framework defined a number of memory structures that captured aspects of the search. The most emblematic is without a doubt the *tabu list*, a list that records attributes of solutions and prohibits, for a certain number of iterations, any solutions that exhibit an attribute on the tabu list.

The same paper that introduced tabu search also coined the word "*meta*-heuristic" [14]. However, not everybody agreed with this term and a push was made to use the (more modest) term "modern heuristics" instead. Clearly, not everybody agreed that the limited set of metaheuristics proposed by the 1980s had a higher-level aspect to them. Many still viewed them essentially as (admittedly, more complicated than their simple counterparts) algorithms, i.e., unambiguous step-by-step sets of operations to be performed. Indeed, it is very common in the late 1980s for a "new metaheuristic" to be described based on a flowchart or another typical algorithmic representation. The widespread realization that metaheuristics could and should be viewed as general frameworks rather than as algorithms would come during the next period, the framework-centric period.

Interestingly, neural networks [21] were among the limited list of metaheuristics proposed by the late 1980s. These methods imitate the functioning of a brain (including neurons and synapses) and were originally proposed in the context of pattern recognition (for which they are still mostly used).

By 1995, research in metaheuristics had grown to a level that could sustain its own conference series, and thus the MIC (Metaheuristics International Conference) series was established. In the same year, the first issue of the Journal of Heuristics (http://link.springer.com/journal/10732/1/1/page/1), the only journal dedicated to publishing research in metaheuristics, was published.

Several other frameworks that had been proposed around the early 1990s also gained increasing interest during the mid-1990s. The innovation proposed in the GRASP (greedy randomized adaptive search procedure) framework was to modify a greedy heuristic by selecting at each iteration not necessarily the best element but one of the best elements randomly [10]. Similarly, ant colony optimization [5] proposed not only to mix deterministic and stochastic information but also proposed a way for solutions to exchange information.

Evolutionary algorithms invariably introduce a large amount of randomness in the search process. Some authors argued that it might be beneficial to reduce the reliance on randomness, but rather create algorithms that perform a more systematic search of the solution space. Scatter search and path relinking, both introduced in Glover [15], are the most notable examples in which the principles of evolutionary algorithms were used with the randomness removed from them [17].

By the second half of the 1990s, however, it gradually became clear that metaheuristics based on metaphors would not necessarily lead to good approaches.

The promised black box optimizers that would always "just work" and that had attracted so much attention seemed elusive. Even the theoretical studies lost some of their shine. The convergence results obtained for simulated annealing [19], because they only worked when an infinite running time was available, were not as compelling for practical situations as initially thought. Similarly, the automatic detection of good building blocks by genetic algorithms only really worked if such building blocks actually existed and if they were not continually being destroyed by the crossover and mutation operators operating on the solutions. Even though the early metaheuristic frameworks offered some compelling ideas, they did not remove the need for an experienced heuristic designer. The advent of metaheuristics had not changed the simple fact that a metaheuristic that extensively exploited the characteristics of the optimization problem at hand would almost always be superior to one that took a black box approach, regardless of the metaheuristic framework used.

In general, researchers during the method-centric period proposed *algorithms*, i.e., formalized structures that were meant to be followed like a cookbook recipe. More often than not, the "new metaheuristic" was given a name, even when the difference between the new method and an existing method was small.

Period 3: The Framework-Centric Period

The insight that metaheuristics could be more usefully described as high-level algorithmic *frameworks*, rather than as algorithms, was a natural thing to happen. The main indicator that this mindset change was taking place – a change that has given rise to a period that we have dubbed the *framework-centric* period – is the increasing popularity of so-called "hybrid" metaheuristics during the early 2000s. Indeed, this period could by rights have been called the "hybrid metaheuristic period." Whereas earlier researchers used to restrain themselves to a single metaheuristic framework, more and more researchers around the turn of the century combined ideas from different frameworks into a single heuristic algorithm. Some combinations became more popular than others, like the use of a constructive heuristic to generate an initial solution for a local search algorithm or the use of GRASP to generate solutions that are then combined using path relinking.

One type of hybrid metaheuristic even received a distinct name: the use of local search (or any "local learning" approach) to improve solutions that are obtained by an evolutionary algorithm was called a *memetic* algorithm [29]. In 2004, the term "hybrid metaheuristic" had become common, and a new conference series with the same name was started.

The hybridization of metaheuristics, however, did not restrict itself to a combination of a metaheuristic with another metaheuristic. Opening up the individual algorithmic frameworks allowed researchers to combine a metaheuristic with any auxiliary method available. Constraint programming, linear programming, and mixed-integer programming were all used in combinations with ideas from metaheuristics. The combination of metaheuristics and exact methods was coined

"matheuristics" [27] (though these methods too had many antecedents in the metaheuristics literature). In 2006, the first edition of the *Matheuristics* conference took place.

Soon after its introduction, the term "hybrid" metaheuristic would become obsolete, as researchers made a general transition from seeing metaheuristics as algorithms of which some components could be borrowed by other metaheuristics to general sets of concepts ("frameworks"). The "metaheuristic framework" concept entailed that metaheuristics were nothing more (or less) than a more or less coherent set of ideas, which could, of course, be freely combined with other ideas. Today, many researchers develop metaheuristics using their experience and knowledge about which methods will work well for certain problems and which most likely will not.

Some general patterns started to appear in the literature on which methods work well for which problems, and the community gravitated toward approaches that always delivered. For almost any variant of the vehicle routing problem, e.g., a large majority of approaches use some form of local search as their main engine, generally in a multi-neighborhood framework like variable neighborhood search [28]. The use of several different local search operators or the use of several different constructive procedures in general is now a well-regarded strategy and often used as the first choice by heuristic designers. Clearly, variable neighborhood search presented a framework within which the use of multiple neighborhoods could be captured, but many other ways of combining several local search operators in a single heuristic are possible. The use of several constructive heuristics (usually combined with several destructive operators) in a single heuristic became known under the name large neighborhood search [1, 33, 43].

Crucial in this period, which is still ongoing, is that researchers do not have to propose a "new algorithm" anymore to get their papers published. By combining the most efficient operators of existing metaheuristic frameworks, and carefully tuning the resulting heuristic, algorithms can be created that solve any real-life optimization problem efficiently. Researchers can now focus on studying a single, mundane aspect of a metaheuristic framework in detail like, e.g., its stopping rules [32].

Recently, a focus can be observed on frameworks that present a much simpler approach that – in many cases – is not much less effective. Iterated local search [25], which proposes to use a single local search operator in alternation with a single perturbation operator, has a long history and dates back to at least the early 1980s [2], yet is still as popular as ever. Its constructive counterpart, often called iterated greedy [12, 34], has recently gained a large amount of traction, despite the fact that it can be seen as a restricted form of strategic oscillation (SO), a technique introduced in Glover [13] often employed in the context of tabu search. SO offers a multitude of concepts and ideas to allow the search to move within and between regions demarcated by various boundaries within the search space, such as those defining feasibility or local optimality or thresholds for various functions (such as objective functions or sums of variables). Reference to such boundaries enriches the search process by introducing different types of moves and

evaluations depending on which side of the boundary the search lies on and on whether the search is moving toward or away from the boundary. Moves leading toward or away from a boundary are joined by moves launched at the boundary and at selected distances from the boundary which involve more complex searches (e.g., by employing exchange moves in place of constructive or destructive moves). Successful applications of strategic oscillation are reported in Glover and Laguna [16], Hvattum et al. [22], Lozano et al. [26], and Corberán et al. [6]. An apparent paradox is that the simplification of a metaheuristic to a few simple rules, which may restrict both its scope and its power, seems to increase its popularity. A possible explanation is that it renders the framework more accessible to non-expert users, who – contrary to the scientific community – value robustness, ease of development, and flexibility over performance.

Traditionally, the metaheuristic community has put a heavy focus on *performance*. Research is only considered good if (and only if) it produces a heuristic algorithm that "performs" well with respect to some benchmark, such as another heuristic or a lower bound. This has been called the "up-the-wall game" (though it might also be called the "one-upmanship game"). All other contributions (e.g., heuristics that are many times simpler than the best-performing heuristic in the literature, studies on heuristics that should perform well but for some reason do not, etc.) are much more difficult to publish. However, several researchers have pointed out the adverse effects of this paradigm (which effectively reduces science to a game), and some recent contributions that go beyond the up-the-wall game demonstrate the framework-centric period is gradually transforming into the *scientific* period. In this period the study of metaheuristics will shift its focus from performance to understanding. Unfortunately, however, not all of the metaheuristic community makes the transition to the framework-centric period, and we are forced to report on a period which essentially runs in parallel with this period.

The Metaphor-Centric Period

Starting in the 1980s, a subfield has arisen of research (we hesitate to put quote marks around the term for reasons that will be explained below) that focuses on the development of new metaheuristic methods based on metaphors of natural or man-made processes. In our history, this period has not been assigned a number because it does not fit chronologically between the other periods, but rather is a side step that happened (and is still happening) in parallel to the framework-centric period.

Although metaphors had been useful in the development of early metaheuristics as a source of inspiration for the development of novel frameworks, it has always been evident to many that a metaphor is only a metaphor and always breaks down at a certain point. It is therefore useful for inspiration, but not necessarily everything about it usefully translates to a metaheuristic framework. Importantly, a metaphor is not enough to *justify* metaheuristic design choices or to create a foundation for completely new metaheuristics.

In recent years, however, a different attitude seems to have taken hold of a subfield of the metaheuristic community. The aim of the "metaphor-based" subfield seems to center entirely around the development of "novel" metaphors that can be used to motivate new metaheuristics. The list of natural and man-made processes that have inspired such metaheuristic frameworks is huge. Ants, bees, termites, bacteria, invasive weed, bats, flies, fireflies, fireworks, mine blasts, frogs, wolves, cats, cuckoos, consultants, fish, glowworms, krill, monkeys, anarchic societies, imperialist societies, league championships, clouds, dolphins, Egyptian vultures, green herons, flower pollination, roach infestations, water waves, optics, black holes, the Lorentz transformation, lightning, electromagnetism, gravity, music making, "intelligent" water drops, river formation, and many, many more, have been used as the basis of a "novel" metaheuristic technique.

Moreover, there does not seem to be any restriction on the type of process that can be translated into a metaheuristic framework. One would expect that, at the very least, the process that is to become the basis for a metaheuristic should *optimize* something (e.g., an annealing process *minimizes* the energy level, natural evolution *minimizes* the discrepancy between the characteristics of a species and the requirements of this species' environment, ants *minimize* the distance between their nest and their source of food). Nevertheless, many metaheuristic frameworks can now be found based on processes that by no stretch of imagination can be said to optimize anything, like fireworks, mine blasts, or cloud formation.

Both the causes and the consequences of this "metaphor fallacy" have been extensively dealt with in a number of other publications [38,42] (short summary: it is not science), and this is not the place to repeat all the arguments why metaphor-based metaheuristics are a bad idea. Nevertheless, metaphor-based "novel" metaheuristics take up a (dark) page in the history of metaheuristics, a page that should be turned quickly.

Period 4: The Scientific Period?

For a long time, the field of metaheuristics has had difficulties to be taken seriously. In 1977, one of the authors of this chapter wrote "[exact] algorithms are conceived in analytic purity in the high citadels of academic research, heuristics are midwifed by expediency in the dark corners of the practitioner's lair [...] and are accorded lower status" [13]. Traditionally, the theoretical underpinning of heuristics and metaheuristics has not been on par with that of other areas in OR, more specifically exact methods. The development of heuristic optimization algorithms, whether using a metaheuristic framework or not, is guided by experience, not theory. Early attempts to firmly ground the development of metaheuristics in theory have not delivered upon their promises. Understanding the behavior of metaheuristics on a fundamental level has proven to be a difficult task, notwithstanding several noteworthy efforts (e.g., [40,41]).

Nevertheless, it is hard to argue with success. The obvious usefulness of metaheuristics in practical optimization problems has drawn researchers to improve

the frameworks and methods developed. To solve a large majority of real-life optimization problems, heuristics are and will remain the only option, whether developed using a metaheuristic framework or not. Nevertheless, there are many things that can be improved about the way the metaheuristic community operates. To list just a few:

- The establishing of adequate testing protocols, to ensure that algorithms perform as well as they are claimed to do.
- The introduction of meta-analysis (i.e., a review of a clearly formulated question that uses systematic methods to identify, select, and evaluate relevant research, as well as to collect and analyze data from relevant studies) to the field of metaheuristics (Hvattum, 2015, Personal communication).
- The requirement to disclose source code, so that researchers can check and build on each other's work without in a more efficient way, without reinventing the wheel.
- The development of powerful general-purpose heuristic solvers to decrease development time, like CPLEX or Gurobi, but then heuristically. LocalSolver seems to be on the right track.
- Supporting these general-purpose solvers, the development of a powerful and generally accepted modeling language, geared more toward the development of heuristics and less toward the MIP paradigm.
- ...

Most importantly, the change from a performance-driven community to a community in which scientific understanding is more important will take place during the scientific period. Without doubt, this will lead to the development of even better heuristics, even more efficient, but it will also lead to heuristics that are usable outside of the developer's lab environment.

Conclusions

Describing the history of the field of metaheuristics in a few pages is not an easy undertaking, and completeness is a goal that simply cannot be achieved. In this chapter, we have attempted to clarify the evolution in this field by not focusing exclusively on important events or publications but by attempting to identify the important paradigm shifts that the field has dealt with. What is certain is that the use of metaheuristics is older, much older, than the term itself. As mentioned, our brain itself houses some powerful metaheuristics that have helped humans survive from the dawn of mankind. The scientific study of metaheuristics, however, had to wait until the second half of the previous century.

Scientific communities invariably develop a conceptual framework within which a few axioms are held to be true. This can also be said of the metaheuristic community. It is those shared truths that we have attempted to uncover in this chapter. Even though the field of metaheuristics is still young, it has already

undergone several paradigm shifts that have changed the way researchers look upon the development of heuristic optimization methods.

The transition from the method-centric to the framework-centric period has been beneficial for the entire community, and there is no doubt that the transition toward the scientific period can take the field further into the right direction. Metaheuristics are a fascinating area of study with highly significant practical ramifications, and the field will certainly keep on evolving in the foreseeable future. There is no doubt that a more scientific, less dogmatic, and broader point of view can help us all in achieving our goals: the development of efficient methods to solve the most challenging and important real-life optimization problems.

References

1. Ahuja RK, Ergun Ö, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discret Appl Math 123(1):75–102
2. Baxter J (1981) Local optima avoidance in depot location. J Oper Res Soc 32:815–819
3. Carbonell JG (1983) Learning by analogy: formulating and generalizing plans from past experience. In: Machine learning. Springer Science & Business Media, Berlin/Heidelberg, pp 137–161. https://doi.org/10.1007/978-3-662-12405-5_5
4. Chu T (2014) Human purpose and transhuman potential: a cosmic vision for our future evolution. Origin Press, San Rafael. ISBN:978-1-57983-0250
5. Colorni A, Dorigo M, Maniezzo V (1992) Distributed optimization by ant colonies. In: Varela FJ, Bourgine P (eds) Proceedings of the first European conference on artificial life. MIT Press, Cambridge, pp 134–142
6. Corberán Á, Peiró J, Campos V, Glover F, Martí R (2016) Strategic oscillation for the capacitated hub location problem with modular links. J Heuristics 22(2):221–244
7. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge. ISBN:978-0-262-03384-8
8. Dueck G (1993) New optimization heuristics. J Comput Phys 104(1):86–92. https://doi.org/10.1006/jcph.1993.1010
9. Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 90(1):161–175. https://doi.org/10.1016/0021-9991(90)90201-b
10. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6(2):109–133. https://doi.org/10.1007/bf01096763
11. Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, New York
12. García-Martínez C, Rodriguez FJ, Lozano M (2014) Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem. Eur J Oper Res 232(3):454–463
13. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(1):156–166. https://doi.org/10.1111/j.1540-5915.1977.tb01074.x
14. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549. https://doi.org/10.1016/0305-0548(86)90048-1
15. Glover F (1998) A template for scatter search and path relinking. In: Artificial evolution. Lecture notes in computer Science. Springer Science & Business Media, pp 1–51. https://doi.org/10.1007/bfb0026589
16. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers/Springer, Boston

17. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control and Cybern 29(3):653–684
18. Goldberg D (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
19. Granville V, Krivanek M, Rasson J-P (1994) Simulated annealing: a proof of convergence. IEEE Trans Pattern Anal Mach Intell 16(6):652–656. https://doi.org/10.1109/34.295910
20. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
21. Hopfield J (1982) Neural networks and physical systems with emergent collective computational capabilities. Proc Natl Acad Sci 79(8):2254–2558
22. Hvattum LM, Løkketangen A, Glover F (2004) Adaptive memory search for boolean optimization problems. Discret Appl Math 142(1):99–109
23. Jarboui B, Derbel H, Hanafi S, Mladenović N (2013) Variable neighborhood search for location routing. Comput Oper Res 40(1):47–57. https://doi.org/10.1016/j.cor.2012.05.009
24. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. https://doi.org/10.1126/science.220.4598.671
25. Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. Int Ser Oper Res Manag Sci 57:321–354
26. Lozano M, Glover F, García-Martínez C, Rodríguez FJ, Martí R (2014) Tabu search with strategic oscillation for the quadratic minimum spanning tree. IIE Trans 46(4):414–428
27. Maniezzo V, Stützle T, Voß S (eds) (2010) Matheuristics. Springer. https://doi.org/10.1007/978-1-4419-1306-7
28. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100. https://doi.org/10.1016/s0305-0548(97)00031-2
29. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts – towards memetic algorithms. Technical Report 826, Caltech Concurrent Computation Program, Pasadena
30. Polya G (2014) How to solve it: a new aspect of mathematical method. Princeton university press, Princeton
31. Rechenberg I (1989) Evolution strategy: nature's way of optimization. In: Optimization: methods and applications, possibilities and limitations. Springer Science & Business Media, Berlin/New York, pp 106–126. https://doi.org/10.1007/978-3-642-83814-9_6
32. Ribeiro CC, Rosseti I, Souza RC (2011) Effective probabilistic stopping rules for randomized metaheuristics: GRASP implementations. In: Learning and intelligent optimization. Lecture notes in computer science, vol 6683. Springer Science & Business Media, pp 146–160. https://doi.org/10.1007/978-3-642-25566-3_11
33. Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472
34. Ruiz R, Stützle T (2008) An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 187(3):1143–1159
35. Shore HH (1970) The transportation problem and the VOGEL approximation method. Decis Sci 1(3-4):441–457. https://doi.org/10.1111/j.1540-5915.1970.tb00792.x
36. Simon HA (1996) The sciences of the artificial, 3rd edn. MIT Press, Cambridge
37. Simon HA, Newell A (1958) Heuristic problem solving: the next advance in operations research. Oper Res 6(1):1–10. https://doi.org/10.1287/opre.6.1.1
38. Sörensen K (2015) Metaheuristics–the metaphor exposed. Int Trans Oper Res 22(1):3–18. https://doi.org/10.1111/itor.12001
39. Sörensen K, Glover FW (2013) Metaheuristics. In: Gass SI, Fu MC (eds) Encyclopedia of operations research and management science, 663 3rd edn, pp 960–970, Springer, Boston, MA
40. Watson J-P, Barbulescu L, Whitley LD, Howe AE (2002) Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. INFORMS J Comput 14(2):98–123. https://doi.org/10.1287/ijoc.14.2.98.120

41. Watson J-P, Beck JC, Howe AE, Whitley LD (2003) Problem difficulty for tabu search in job-shop scheduling. Artif Intell 143(2):189–217. https://doi.org/10.1016/s0004-3702(02)00363-6
42. Weyland D (2010) A rigorous analysis of the harmony search algorithm. Int J Appl Metaheuristic Comput 1(2):50–60. https://doi.org/10.4018/jamc.2010040104
43. Yagiura M, Iwasaki S, Ibaraki T, Glover F (2004) A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. Discret Optim 1(1):87–98

Parallel Metaheuristic Search

28

Teodor Gabriel Crainic

Contents

Introduction . . . 810
Meta-heuristics and Parallelism . . . 811
  Sources of Parallelism . . . 811
  Characterizing Parallel Meta-heuristic Strategies . . . 813
Low-Level Parallelization Strategies . . . 814
Data Decomposition . . . 817
Independent Multi-search . . . 818
Cooperative Search . . . 819
  Synchronous Cooperation . . . 822
  Asynchronous Cooperation . . . 825
Advanced Cooperation Strategies: Creating New Knowledge . . . 829
  pC/KC with Solvers Working on the Complete Problem . . . 830
  pC/KC with Partial Solvers: The Integrative Cooperative Search . . . 833
Conclusions . . . 836
References . . . 839

Abstract

The chapter presents a general picture of parallel meta-heuristic search for optimization. It recalls the main concepts and strategies in designing parallel meta-heuristics, pointing to a number of contributions that instantiated them for neighborhood- and population-based meta-heuristics, and identifies trends and promising research directions. The focus is on cooperation-based strategies, which display remarkable performances, in particular strategies based on asyn-

T. G. Crainic (✉)
CIRRELT – Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, QC, Canada

School of Management, Université du Québec à Montréal, Montréal, QC, Canada
e-mail: TeodorGabriel.Crainic@cirrelt.net

© Springer International Publishing AG, part of Springer Nature 2018
R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_40

chronous exchanges and the creation of new information out of exchanged data to enhance the global guidance of the search.

Keywords
Parallel computation · Parallel meta-heuristics · Functional and data decomposition · Independent multi-search · Synchronous and asynchronous cooperative search · Integrative cooperative search

Introduction

Meta-heuristics often offer the only practical approach to addressing complex problems of realistic dimensions and are thus widely acknowledged as essential tools in numerous and diverse fields. Yet, even meta-heuristics may reach quite rapidly the limits of what may be addressed in acceptable computing times for many research and practice problem settings. Moreover, heuristics do not generally guarantee optimality, performance often depending on the particular problem setting and instance characteristics. Robustness is therefore a major objective in meta-heuristic design, in the sense of offering a consistently high level of performance over a wide variety of problem settings and instance characteristics.

Parallel meta-heuristics aim to address both issues. Their first goal is to solve larger problem instances than what is achievable by sequential methods and to do it in reasonable computing times. In appropriate settings, such as cooperative multi-search strategies, parallel meta-heuristics also prove to be much more robust than sequential versions in dealing with differences in problem types and characteristics. They also require less extensive, and expensive, parameter-calibration efforts.

The objective of this chapter is to paint a general picture of the parallel optimization meta-heuristic field. Following [40], it recalls the main concepts and strategies in designing parallel meta-heuristics, pointing to a number of contributions that instantiated them for neighborhood- and population-based meta-heuristics, and identifies a number of trends and promising research directions. It focuses on cooperation-based strategies, which display remarkable performances, reviewing in somewhat more depth the recently introduced integrative cooperative search [96]. Notice that parallel meta-heuristic strategies are examined and discussed from the conceptual, algorithmic design point of view, independent of implementation on particular computer architectures.

The parallel meta-heuristic field is very broad, while the space available for this chapter is limited. In addition to the references provided in the following sections, the reader may consult a number of surveys, taxonomies, and syntheses of parallel meta-heuristics, some addressing methods based on particular methodologies, while others address the field in more comprehensive terms. Methodology-dedicated syntheses may be found in [5, 84–86, 133] for parallel simulated annealing, [20, 21, 107, 119, 147] for genetic-based evolutionary methods, [32, 43, 45, 82, 170] for tabu search, [73] for scatter search, [18, 62, 92] for ant colony methods, and [116]

for variable neighborhood search (VNS). Surveys and syntheses that address more than one methodology may be found in [2, 4, 33, 36–38, 40, 50, 51, 90, 97, 125, 168].

The chapter is organized as follows. Section "Meta-heuristics and Parallelism" is dedicated to a general discussion of the potential for parallel computing in meta-heuristics, a brief description of performance indicators for parallel meta-heuristics, and the taxonomy used to structure the presentation. Section "Low-Level Parallelization Strategies" addresses strategies focusing on accelerating computing-intensive tasks without modifying the basic algorithmic design. Methods based on the decomposition of the search space are treated in section "Data Decomposition," while strategies based on the simultaneous exploration of the search space by several independent meta-heuristics constitute the topic of section "Independent Multi--search." Cooperation principles are discussed in section "Cooperative Search" and are detailed in sections "Synchronous Cooperation," "Asynchronous Cooperation," and "Advanced Cooperation Strategies: Creating New Knowledge." We conclude in section "Conclusions."

Meta-heuristics and Parallelism

We start with a brief overview of the potential for parallel computing in meta-heuristics and of performance indicators for parallel meta-heuristics, followed by the criteria used to describe and characterize the parallelization strategies for meta-heuristics described in the other sections of the chapter.

Sources of Parallelism

Addressing a given problem instance with a parallel solution method means that several processes work simultaneously on several processors with the goal of identifying the best solution for the instance. Parallelism thus follows from a decomposition of the total work load and the distribution of the resulting tasks to available processors. According to how "small" or "large" are the tasks in terms of algorithm work or search space, the parallelization is denoted *fine* or *coarse grained*, respectively.

The decomposition may concern the algorithm, the problem instance data, or the problem structure. *Functional parallelism* identifies the first case, where some computing-intensive components of the algorithm are separated into several tasks (processes), possibly working on the "same" data, which are allocated to different processors and run in parallel. The main source of functional parallelism for meta-heuristics is the concurrent execution of their innermost loop iterations, e.g., evaluating neighbors, computing the fitness of individuals, or having ants forage concurrently (section "Low-Level Parallelization Strategies"). This is often also the only source of readily available parallelism in meta-heuristics, most other steps being time dependent and requiring either the computation of the previous steps to be completed or the synchronization of computations to enforce this

time dependency. Consequently, functional parallelism is mainly interesting as a low-level component of hierarchical parallelization strategies or when addressing problem settings requiring a significant part of the computing effort to be spent in the inner-loop algorithmic component.

Parallelism for meta-heuristics may further be found in the domain of the problem addressed or the corresponding search space (for brevity reasons, the term "search space" is used in the rest of the chapter). There are indeed no data dependencies between the evaluation functions of different solutions, and, thus, these may be computed in parallel. Moreover, theoretically, the parallelism in the search space is as large as the space itself. Parallelism is then obtained by separating the search space into components allocated to the available processors. In most cases, these components are still too large for explicit enumeration, and an exact or heuristic search method has to be associated to each to implicitly explore it. *Space separation* is exploited in most of the strategies described in this chapter.

Space separation raises a number of issues with respect to defining an overall meta-heuristic search strategy, in particular, (1) whether to define the separation by partitioning the space, allowing components to partially overlap, or not, (2) how to control an overall search conducted separately on several components of the original space, (3) how to build a complete solution out of those obtained while exploring the components, and (4) how to allocate computing resources for an efficient exploration avoiding, for example, searching regions with poor-quality solutions.

Two main approaches are used to perform the search-space separation: *domain decomposition* (also called *data parallelism*) and *multi-search* (the name *multiple walks* is also found in the literature). The former explicitly separates the search space (section "Data Decomposition") and then addresses the initial problem instantiated on each of the resulting restricted regions, before combining the corresponding partial solutions into complete ones.

Multi-search strategies implicitly divide the search space through concurrent explorations by several methods, named *solvers* in the following. These meta-heuristic or exact solvers may address either the complete problem at hand or explore partial problems defined by decomposing the initial problem through mathematical programming or attribute-based heuristic approaches. In the former case, the decomposition method implicitly defines how a complete solution is built out of partial ones. In the latter case, some processors work on the partial problems corresponding to the particular sets of attributes defined in the decomposition, while others combine the resulting partial solutions into complete solutions to the original problem. Multi-search strategies, particularly those based on cooperation principles, are at the core of most successful developments in parallel meta-heuristics and the object of the largest number of recent publications in the field. Sections "Independent Multi-search," "Cooperative Search," "Synchronous Co-operation," "Asynchronous Cooperation," and "Advanced Cooperation Strategies: Creating New Knowledge" describe multi-search meta-heuristics.

We complete this subsection with a few notes on the performance evaluation of parallel meta-heuristic strategies and resulting algorithms.

The traditional goal when designing parallel solution methods is to reduce the time required to "solve," exactly or heuristically, given problem instances or to address larger instances without increasing the computational effort. For solution methods that run until the provable optimal solution is obtained, this translates into the well-known *speedup* performance measure, computed as the ratio between the wall-clock time required to solve the problem instance in parallel with p processors and the corresponding solution time of the best-known sequential algorithm. A somewhat less restrictive measure replaces the latter with the time of the parallel algorithm run on a single processor. See [9] for a detailed discussion of this issue, including additional performance measures.

Speedup measures are more difficult to interpret when the optimal solution is not guaranteed or the exact method is stopped before optimality is reached. Moreover, most parallelization strategies design parallel meta-heuristics that yield solutions that are different in value, composition, or both from those obtained by the sequential counterpart. Thus, equally important performance measures for parallel heuristics evaluate by how much they outperform their sequential counterparts in (relative) terms of solution quality and, ideally, the computational efficiency. Simply put, the parallel method should not require a higher overall computation effort than the sequential method or should justify the extra effort by higher-quality solutions.

Search robustness is another characteristic expected of parallel heuristics. Robustness with respect to a problem setting is meant here in the sense of providing "equally" good solutions to a large and varied set of problem instances, without excessive calibration, neither during the initial development nor when addressing new problem instances.

Multi-search methods, particularly those based on cooperation, generally offer enhanced performances compared to sequential methods and other parallelization strategies. They display behaviors different from those of the sequential methods involved and can be seen as proper meta-heuristics [2], usually finding better-quality solutions and enhancing the robustness of the meta-heuristic search. See [37,38] for a discussion of these issues.

Characterizing Parallel Meta-heuristic Strategies

Several strategies may be defined based on each one of the sources of parallelism discussed above. The classification of Crainic and Hail [36], generalizing that of Crainic, Toulouse, and Gendreau [43] ([168] and [51] present classifications that proceed of the same spirit), is used in this chapter to characterize these strategies.

The three dimensions of the classification focus on the control of the global problem-solving process, the information exchanged among processes, and the diversity of the search, respectively. The first dimension, *search control cardinality*, thus specifies whether the global search is controlled by a single process or by several processes that may collaborate or not. The two alternatives are identified as *1-control* (*1C*) and *p-control* (*pC*), respectively.

The second dimension addresses the issue of information exchanges and the utilization of the exchanged information to control or guide the search, hence the *search control and communications* name. Communications may proceed either *synchronously* or *asynchronously*. In the former case, processes stop and engage in some form of communication and information exchange at moments (number of iterations, time intervals, specified algorithmic stages, etc.) exogenously planned, either hard-coded or determined by a control (master) process. In the asynchronous communication case, each process is in charge of its own search, as well as of establishing communications with other processes, and the global search terminates once all individual searches stop. Four classes are defined within this dimension to reflect the quantity and quality of the information exchanged and shared, as well as the additional knowledge derived from these exchanges (if any): *rigid (RS)* and *knowledge synchronization (KS)* and, symmetrically, *collegial (C)* and *knowledge collegial (KC)*.

More than one solution method or variant (e.g., with different parameter settings) may be involved in a parallel meta-heuristic. The third dimension thus indicates the *search differentiation* or diversity: do solvers start from the same or different solutions and do they make use of the same or different search strategies? The four classes are *same initial point/population, same search strategy (SPSS)*; *same initial point/population, different search strategies (SPDS)*; *multiple initial points/populations, same search strategies (MPSS)*; and *multiple initial points/populations, different search strategies (MPDS)*. Obviously, one uses "point" for neighborhood-based methods, while "population" is used for genetic-based evolutionary methods, scatter search, and swarm methods.

Low-Level Parallelization Strategies

Functional-parallelism-based strategies, exploiting the potential for task decomposition within the inner-loop computations of meta-heuristics, are often labeled "low level" because they modify neither the algorithmic logic nor the search space. They aim solely to accelerate the search and generally do not modify the search behavior of the sequential meta-heuristic. Typically, the exploration is initialized from a single solution or population and proceeds according to the sequential meta-heuristic logic, while a number of computing-intensive steps are decomposed and simultaneously performed by several processors.

Most low-level parallel strategies belong to the 1C/RS/SPSS class and are usually implemented according to the classical *master-slave* parallel programming model. A "master" program executes the 1-control sequential meta-heuristic, separating and dispatching computing-intensive tasks to be executed in parallel by "slave" programs. Slaves perform evaluations and return the results to the master which, once all the results are in, resumes the normal logic of the sequential meta-heuristic. The complete control on the algorithm execution thus rests with the master, which decides the work allocation for all other processors and initiates most communications. No communications take place among slave programs.

The neighborhood-evaluation procedure of the local search component of neighborhood-based meta-heuristics (as well as of population-based ones implementing advanced "schooling" for offspring) is generally targeted in 1C/RS/SPSS designs. The master groups the neighbors into tasks and sends them to slaves. Each slave then executes the exploration/evaluation procedure on its respective part of the neighborhood and sends back the best, or first improving, neighbor found. The master waits for all slaves to terminate their computations, selects the best move, and proceeds with the search. See, e.g., [70] and [129] for applications of this strategy to tabu search meta-heuristics for the vehicle routing problem with time window constraints (VRPTW) and the scheduling of dependent tasks on heterogeneous processors, respectively.

The appropriate granularity of the decomposition, that is, the size of the tasks, depends upon the particular problem and computer architecture but is generally computationally sensitive to inter-processor communication times and workload balancing. Thus, for example, [54] discusses several decomposition policies for the permutation-based local search neighborhood applied to the scheduling of dependent tasks on homogeneous processors and shows that the uniform partition usually called upon in the literature is not appropriate in this context characterized by neighborhoods of different sizes. The authors also show that a fixed coarse-grained nonuniform decomposition, while offering superior results, requires calibration each time the problem size or the number of processors varies.

The best performing strategy was called by the authors *dynamic fine grained*. It defines each neighbor evaluation as a single task, the master dynamically dispatching these on a first-available, first-served basis to slave processors as they complete their tasks. The strategy partitions the neighborhood into a number of components equal to the number of available processors but of unequal size with a content dynamically determined at each iteration.

The dynamic fine-grained strategy provides maximum flexibility and good load balancing, particularly when the evaluation of neighbors is of uneven length. The uniform distribution appears more appropriate when the neighbor evaluations are sensibly the same, or when the overhead cost of the dynamic strategy for creating and exchanging tasks appears too high.

Similar observations may be made regarding population-based meta-heuristics. In theory, all genetic-algorithm operators may be addressed through a 1C/RS/SPSS design, and the degree of possible parallelism is equal to the population size. In practice, the computations associated to most operators are not sufficiently heavy to warrant parallelizing, while overhead costs may significantly reduce the degree of parallelism and increase the granularity of the tasks. Consequently, the fitness evaluation is often the only target of 1C/RS/SPSS parallelism for genetic-evolutionary methods, the resulting parallel GA being implemented using the master-slave model. Similarly to other 1-control low-level parallelizations, a 1C/RS/SPSS genetic-evolutionary algorithm performs the same search as the sequential program, only faster.

The 1C/RS/SPSS parallelism for ant colony and, more generally, swarm-based methods lies at the level of the individual ants. Ants share information indirectly

through the pheromone matrix, which is updated once all solutions have been constructed. There are no modifications of the pheromone matrix during a construction cycle, and, thus, each individual ant performs its solution-construction procedure without data dependencies on the progress of the other ants.

Most parallel ant colony methods implement some form of 1C/RS/SPSS strategy according to the master-slave model, including [18, 59, 132, 134, 157]. The master builds tasks consisting of one or several ants (each can be assimilated to a "small" colony) and distributes them to the available processors. Slaves perform their construction heuristic and return their solution(s) to the master, which updates the pheromone matrix, returns it to the slaves, and so on. To further speed up computation, the pheromone update can be partially computed at the level of each slave, each slave computing the update associated to its solutions. The fine-grained version with central matrix update has been the topic of most contributions so far, and, in general, it outperformed the sequential version of the algorithm. It is acknowledged, however, that it does not scale, and, similarly to other meta-heuristics, this strategy is outperformed by more advanced multi-search methods.

Scatter search and path relinking implement different evolution strategies, where a restricted number of elite solutions are combined, the result being enhanced through a local search or a full-fledged meta-heuristic, usually neighborhood based. Consequently, the 1C/RS/SPSS strategies discussed previously regarding the parallelization of local search exploration apply straightforwardly to the present context, as in [72–74] for the p-median and the feature selection problems.

A different 1C/RS/SPSS strategy for scatter search may be obtained by running concurrently the combination and improvement operators on several subsets of the reference set. Here, the master generates tasks by extracting a number of solution subsets, which are sent to slaves. Each slave then combines and improves its solutions, returning its results to the master for the global update of the reference set. Each subset sent to a slave may contain exactly the number of solutions required by the combination operator or a higher number. In the former case, the slave performs an "iteration" of the scatter search algorithm [72–74]. In the latter, several combination-improvement sequences could be executed, and solutions could be returned to the master as they are found or all together at the end of all sequences. This heavy load for slaves may conduct to very different computation times, and, thus, load-balancing capabilities should be added to the master.

To conclude, low-level, 1-control parallel strategies are particularly attractive when neighborhoods (populations) are large or neighbor (individual) evaluation procedures are costly, and a significant gain in computing time may be obtained (e.g., the parallel tabu searches of [23, 25, 150] for the quadratic assignment problem (QAP), [24] for the traveling salesman problem (TSP), [128–130] and [54] for the task-scheduling problem). More advanced multi-search strategies generally outperform low-level strategies in most cases. Yet, when a sufficiently large number of processors are available, it might prove worthy to combine a 1C/RS/SPSS approach and more sophisticated strategies into hierarchical solution schemes (e.g., [136] were low-level parallelism accelerated the move evaluations of the individual searches engaged into an independent multi-search procedure for the VRP).

Data Decomposition

Domain or *search-space decomposition* constitutes an intuitively simple and appealing parallelization strategy, dividing the search space into smaller partial sets, solving the resulting subproblems by applying the sequential meta-heuristic on each set, collecting the respective partial solutions, and reconstructing an entire solution out of the partial ones. This apparently simple idea may take several forms, however, according to the type of division performed and the permitted links among the resulting sets/subproblems.

The space may be *partitioned*, yielding disjoint partial sets, or a cover may be defined allowing a certain amount of overlap among partial sets. Thus, for example, the arc-design variables of a VRP may be partitioned into customer subsets (including the depot in each subset), while a cover would allow "close by" customers to belong to two subsets. The goal generally is to generate independent subproblems, which allows to discard from each subproblem the variables and constraints not directly involved in its definition. When this is not possible, e.g., the separated activities share some resources, one may fix the variables not in the subproblem definition (and thus project the corresponding constraints). One then still works on a smaller subproblem, but considering the complete vector of decision variables.

The second element one must consider is the degree of exploration overlap permitted among subproblems. One must thus decide whether the solution transformations (e.g., neighborhood moves or individual crossovers) performed within the partial set of a given subproblem are restricted to that partial set or may involve variables in neighboring subspaces creating an indirect overlapping of subsets. Strict partitioning strategies restrict the solvers to their subsets, resulting in part of the search space being unreachable and the parallel meta-heuristic being nonoptimal. Explicit or implicit overlapping partially addresses this issue. Only partially because, to fully ensure that all potential solutions are reachable, one needs to make overlapping cover the entire search space, which would negate the benefits of decomposition.

Consequently, strict partitioning and very limited overlapping are the preferred approaches found in the literature. A re-decomposition feature is generally included to increase the thoroughness of the search and allow all potential solutions to be examined. The decomposition is thus modified at regular intervals, and the search is restarted using the new decomposition. This feature provides also the opportunity to define a non-exhaustive decomposition, i.e., where the union of the subsets is smaller than the complete search space. A complete solution reconstruction feature is almost always part of the procedure.

This strategy is naturally implemented using master-slave 1C/RS schemes, with MPSS or MPDS search differentiation. The master process determines the decomposition and sends partial sets to slaves, synchronizes them and collects their solutions, reconstructs solutions, modifies the decomposition, and determines stopping conditions. Slaves concurrently and independently perform the search on their assigned partial sets. Design issues one must address in this context are the length of the exploration available to slaves and the reconstruction of global context

information (e.g., global tabu list) out of the partial ones. The extreme case of executing a full meta-heuristic on each partial set of the search space (this avoids the context issue), periodically modifying the partition and restarting the search, was actually generally used, particularly for problems for which a large number of iterations can be performed in a relatively short time, and restarting the method with a new decomposition does not require an unreasonable computational effort (e.g., [66] for the TSP, [95] for image filtering, and [80] for real-time ambulance fleet management).

In a pC/KS strategy, with MPSS or MPDS search differentiation, the decomposition is collegially decided and modified through information exchange phases (e.g., round-robin or many-to-many exchanges) activated at given synchronization points. Such an approach was proposed in [151] for the VRP, where the customer set was partitioned, vehicles were allocated to the resulting regions, and each subproblem was solved by an independent tabu search. All processors stopped after a number of iterations that varied according to the total number of iterations already performed, and the partition was modified by exchanging tours, undelivered cities, and empty vehicles between adjacent processors (corresponding to neighboring regions). At the time, this approach did allow to address successfully a number of problem instances, but the synchronization inherent in the design of the strategy hindered its performance. A parallel ant colony approach combining this decomposition idea to a master-slave implementation was presented in [60] (parallelizing the algorithm presented in [138]), where the master generates an initial solution, defines the partition, and updates the global pheromone matrix, while slaves execute a savings-based ant colony algorithm [137] for the resulting restricted VRP.

Data decomposition methods induce different search behavior and solution quality compared to those of the sequential meta-heuristic. Such methods appear increasingly needed as the dimensions of the contemplated problem instances continue to grow. Given the increased complexity of the problem settings, work is also required on how to best combine search-space decomposition and the other parallelization strategies, cooperation in particular. The integrative cooperative search of [96] is a step in this direction (see section "Advanced Cooperation Strategies: Creating New Knowledge").

Independent Multi-search

Independent multi-search was among the first parallelization strategies proposed in the literature. It is also the most simple and straightforward p-control parallelization strategy and generally offers very interesting performance.

Independent multi-search seeks to accelerate the exploration of the search space toward a better solution (compared to sequential search) by initiating simultaneous solvers from different initial points (with or without different search strategies). It thus parallelizes the multi-start strategy by performing several searches simultaneously on the entire search space, starting from the same or from different initial solutions, and selecting at the end the best among the best solutions obtained by all

searches. Independent multi-search methods thus belong to the pC/RS class of the taxonomy. No attempt is made to take advantage of the multiple solvers running in parallel other than to identify the best overall solution at the synchronization moment when all searches stop.

Independent multi-search turns out to be effective, simply because of the sheer quantity of computing power it allows one to apply to a given problem. Formal insights into the behavior of these strategies may be found in [11, 149, 152, 160]. Empirical efficiency was shown by many contributions that took advantage of its simplicity of implementation and relatively good performance expectation. pC/RS/MPSS parallelizations of neighborhood-based meta-heuristics were thus proposed for, e.g., tabu search for the QAP [11], VRP [136, 152, 156] and production planning [14]; GRASP for the QAP [105, 124, 126], the Steiner problem [110, 111], and the 2-path telecommunication network design [139–141]; simulated annealing for graph partitioning [7, 8, 104] and the TSP [114]; and variable neighborhood search for the p-median problem [71]. Independent multi-search pC/RS/MPSS applications to nongenetic-evolutionary methods have been proposed for scatter search [72, 74], as well as for ant colony optimization for set covering [132], the TSP [149], and the VRP [61]. Similar performance was observed for genetic methods with full-sized populations [28, 29], which avoided the premature convergence observed for pC/RS independent multi-search GA with small-sized populations obtained by separating the initial population among the independent GA searches (e.g., [88, 144]).

Independent multi-search offers an easy access to parallel meta-heuristic computation, offering a tool when looking for a "good" solution without investment in methodological development or actual coding. Independent multi-search methods are generally outperformed by cooperative strategies, however, the latter integrating mechanisms enabling the independent solvers to share, during the search, the information their exploration generates. As explained in the following sections, this sharing and the eventual creation of new information out of the shared one yield in most cases a collective output of superior solutions compared to independent and sequential search.

Cooperative Search

Cooperative multi-search has emerged as one of the most successful meta-heuristic methodologies to address hard optimization problems (see, e.g., [2, 32, 33, 36, 39, 40, 155, 159]). Cooperative search is based on harnessing the capabilities of several solution methods through *cooperation mechanisms* providing the means to share information while addressing the same problem instance (and create new information out of the exchanged data in advanced settings; see section "Advanced Cooperation Strategies: Creating New Knowledge").

Cooperative search strategies are thus defined by the solver components engaged in cooperation, the nature of the information shared, and their interaction mechanism. The solvers define trajectories in the search space from possibly different

initial points or populations, by using possibly different search strategies (including the same method with different parameter settings or populations). The information-sharing cooperation mechanism specifies how these independent solvers interact, how the exchanged information is used globally (if at all), and how each process acts on the received information, using it within its own search and, thus, transforming it before passing it to other solvers. As further detailed in the following sections, various cooperation mechanisms have been proposed: diffusion among "neighboring" solution methods arrayed in particular communication architectures, e.g., fine-grained, cellular GA (e.g., [21, 108]) and multilevel cooperative search [165]; direct exchanges among processes as in coarse-grained, island GA [21, 108], A-teams [158, 159], and collegial asynchronous strategies [42, 43]; and indirect exchanges through a common data repository and management structure such as the *adaptive* [6, 12, 142] and *central memory* [42, 43, 46, 93, 94, 100] strategies. The global search behavior of the cooperative parallel meta-heuristic then emerges from the local interactions among the independent solvers, yielding a "new" meta-heuristic in its own right [39]. The similarity between this behavior and that of systems where decisions emerge from interactions among autonomous and equal "colleagues" has inspired the name *collegial* associated to cooperative search strategies in the taxonomy used in this chapter.

The exchanged information has to be *meaningful* and *timely*. The goals are twofold. First is to enhance the performance of the receiving solvers. Second is to create a global image of the status of the cooperative search as "complete" as possible, which would provide the means to guide the global search toward better performance in terms of computing time and solution quality than the simple concatenation of the results of the individual solvers. Of course, one desires to achieve these goals without excessive overhead. Toulouse, Crainic, and Gendreau [162] proposed a list of fundamental issues to be addressed when designing cooperative parallel strategies for meta-heuristics: What information is exchanged? Between what processes is it exchanged? When is information exchanged? How is it exchanged? How is the imported data used? Implicit in their taxonomy and explicitly stated in later papers, the issue of whether the information is modified during exchanges or whether new information is created completes this list.

"Good" solutions make up the most often exchanged type of information. This usually takes the form of the local current best solution of a given solver or the overall best. The question of when to send such solutions has to be carefully addressed, however, particularly when the receiving process is supposed to act momentarily on the incoming information. One should thus avoid sending all local current best solutions, particularly when the solver is performing a series of improving moves or generations, as successive solutions are generally "similar" and the receiving solvers have no chance to actually act on the incoming information. Sending the overall current best solution to all cooperating solvers should also be avoided, as it rapidly decreases the diversity of the exploration and, thus, increases the amount of worthless computational work (many solvers will search within the same region) and brings an early "convergence" to a not-so-good solution. Sending out local optima only, exchanging groups of solutions, implementing randomized

selection procedures (generally biased toward good or good-and-diverse solutions) of the solutions to share, and having the cooperating solvers treat differently the received information are among the strategies aimed at addressing these issues.

Context information may also be profitably shared and integrated into the mechanisms used to guide the overall search. Context information is routinely collected by meta-heuristics during their search. It may consist in statistical information relative to the presence of particular solution elements in improving solutions (e.g., the medium- and long-term memories of tabu search), the impact of particular moves on the search trajectory (e.g., the scores of the moves of large adaptive neighborhood search), population diversity measures, individual resilience across generations, etc. A limited number of studies indicate the interest of context information exchanges (see section "Advanced Cooperation Strategies: Creating New Knowledge"), but more research is needed on this topic.

Cooperating solvers may communicate and exchange information directly or indirectly. *Direct* exchanges of information occur either when the concerned solvers agree on a meeting point in time to share information or when a solver broadcasts its information to one or several other solvers without prior mutual agreement. The latter case is generally not performing well, except when solvers include costly mechanisms to store such information without disturbing their own execution until ready to consider it.

Indirect exchanges of information are performed through independent data structures that become shared sources of information solvers may access according to their own internal logic to post and retrieve information. Such data structures are known under various names, e.g., *blackboard* in computer science and artificial intelligence vocabulary and *memory*, *pool*, and *data warehouse* (the terms *reference* and *elite set* are also sometimes used) in the parallel meta-heuristic literature. The term *memory* is used in the following.

Centralized memory is the implementation of choice reported in the literature. Distributed memory mechanisms may be contemplated, where a number of memories are interconnected, each servicing a number of solvers. Such hierarchical structures, involving several layers of solvers and memories, appear interesting when a large number of processors are to be involved, for integrative cooperation strategies, or when computations are to take place on grids or loosely coupled distributed systems. Issues related to data availability, redundancy, and integrity must be then addressed, as well as questions relative to the balancing of workloads and the volume of information exchanged. More research is needed on this topic.

The logical intercommunication network corresponding to the selected cooperation strategy takes the form of a *communication graph*. A node of the graph represents a solver or a memory. Edges define the pairs of solvers or of a solver and a memory that may communicate directly. The projection of the communication graph on the physical interconnection topology of the parallel computer executing the parallel program – complete graph, ring, grid, torus, and star are most often encountered in the literature – is normally part of the implementation process.

When and how information is exchanged specifies how frequently cooperation activities are initiated, by whom, and whether all concerned solvers must

simultaneously engage in communications or not. *Synchronous* and *asynchronous* communications are the two classes of communication exchanged and are discussed in the following sections. The accumulated knowledge of the field indicates for both classes that exchanges should not be too frequent to avoid excessive communication overheads and premature "convergence" to local optima [42, 43, 162].

Three remarks complete this section. First, "simple" cooperation designs based, for example, on synchronization or on exchanging current best solutions only often appear biased toward intensifying the search in already-explored regions where interesting solutions have been identified. Diversification capabilities, e.g., probabilistic or diversity-driven selection of exchanged solutions, are thus an important component of cooperative p-control strategies.

One also observes that the main principles of cooperative parallel strategies are the same for neighborhood- and population-based meta-heuristics, even though denominations and implementation approaches may differ. The terms *coarse-* and *fine-grained island* are thus used to identify the amplitude of the population (large or small, down to single individual eventually, respectively) of participating solvers in genetic-based cooperation. Similarly, *multi-colony* is the term generally used for cooperation in the ant colony community. The next sections are thus structured around classes of strategies rather than by meta-heuristic type.

Finally, one should notice that cooperation takes place at two different levels. The first is the *explicit* information sharing specified by the design of cooperation mechanism. *Implicit* cooperation makes up the second level, where information spreads across the cooperating solvers through a diffusion process and correlated interactions [163, 164, 166, 167]. Implicit cooperation is **not** specified explicitly in the design of the algorithm. It is thus a bottom-up, global emergent phenomenon produced by the correlated interactions among searches. Important research issues and challenges are related to how to harness indirect cooperation to enhance the optimization capabilities of cooperative search. For example, how should one select solvers and direct cooperation mechanisms to yield a system-wide emergent behavior providing an efficient exploration of the solution space from an optimization point of view? Learning and dynamic self-adaptation, at the level of both individual solvers and the cooperating meta-heuristic, appear to be part of the answer. Several other areas of research study systems displaying emergent behaviors, e.g., decentralized autonomic computing, social robotics, swarm intelligence, clustering logistics activities in supply chains, etc., and cross-fertilization appear promising. Empirical and theoretical research in this area should yield design principles and tools for more powerful (parallel) meta-heuristics.

Synchronous Cooperation

Synchronous cooperation follows a pC/KS scheme, with any of the SPDS, MPSS, or MPDS search differentiation approaches, according to which the independent cooperating meta-heuristics synchronize at particular moments to initiate an information exchange phase. Synchronize here means that every solver but the last stops

its activities and waits for all others to be ready. The synchronization moments may be generated based on conditions external to all solvers (e.g., number of iterations since the last synchronization) or detected by a specially designated solver. The information exchange phase must be completed before any solver can restart its exploration from the respective synchronization point.

Synchronization may use a complete communication graph or a more restricted, less densely connected communication topology, e.g., a ring, torus, or grid graph. *Global* exchanges of information among all solvers take place in the former case, while information follows a *diffusion* process through direct *local* exchanges of information among neighboring processes in the latter. In all cases, the objective is to re-create a state of complete knowledge at particular moments in the global search and, thus, to hopefully guide it toward a coordinated evolution to the desired solution to the problem. This goal is rarely attained, however, and the price in computing-time efficiency may be significant, as communications cannot be initiated before the slowest solver is ready to proceed.

Global Information Exchanges

Many pC/KS cooperative search meta-heuristics in the literature implement the strategy according to the master-slave model. The master process, which may or may not include one of the participating solvers, initiates the other processes, stops them at synchronization points, gathers the information to be shared, updates the global data, decides on the termination of the search and, either effectively terminates it or distributes the shared information (a good solution, generally, the overall best solution in many cases) to the solvers for the continuation of the search.

The VNS pC/KS method for the p-median problem proposed in [71] followed this idea, as well as the tabu search-based algorithms proposed for the TSP [109], the VRP (using ejection chains) [135, 136], the QAP [55] and the task mapping problem [56], the last two contributions attempting to overcome the limitations of the master-slave implementation by allowing processes, on terminating their local search phases, to synchronize and exchange best solutions with processes running on neighboring processors (this idea represents a step toward a "true" pC/KS design using a partial solution-diffusion process). This idea was also used to implement coarse-grained island-based cooperating genetic methods [52, 148], where the master stopped the cooperating meta-heuristics to initiate a *migration* operator exchanging among the independent populations the best or a small subset of the best individuals in each. Applied to ant colony systems [64], this strategy divided the colony into several subcolonies, each assigned to a different processor. Each independent ant colony meta-heuristic sent to the master its best solution once its ants finished searching. The master updated the pheromone matrix and started a new search phase. A more sophisticated pC/KS approach was proposed in [120] for the 0–1 multidimensional knapsack problem, where the master dynamically adjusted the parameters of the cooperating tabu searches according to the results each had obtained so far. Computational results showed this dynamic adjustment to be beneficial.

Alternatively, pC/KS schemes can be implemented in "true" collegial fashion by empowering each cooperating solver to initiate synchronization once it reaches a predetermined status. It then broadcasts its data, followed by similar broadcasts performed by the other solvers. Once all broadcasts are completed and information is shared, each solver performs its own import procedures on the received data and proceeds with its exploration of the search space until the next synchronization event.

Such an approach, exchanging the current best solution or group of solutions, was proposed for simulated annealing [58], where the solvers transmitted their best solutions every n steps and restarted the search after updating their respective best solutions (see also [101–104] for the graph partitioning problem). For tabu search applied to location problems with balancing requirements [41, 42], solvers synchronized after a number of iterations either predefined or dynamically determined. Most synchronous coarse-grained island genetic parallel methods applied this strategy, migration operators being applied at regular intervals, e.g., [171] for satisfiability problems (the best individual of each population migrated to replace the worst of the receiving population), [67] for multi-objective telecommunication network design with migration following each generation, and [27–29, 89, 107] for graph partitioning, the latter implementing a hierarchical method, where the fitness computation was performed at the second level (through a master-slave implementation; the overhead due to the parallelization of the fitness became significant for larger numbers of processors). A similar strategy was proposed for the multi-ant colony algorithms [112, 113]. Each colony has its own pheromone matrix and may (homogeneous) or may not (heterogeneous) use the same update rule. Colonies synchronize after a fixed number of iterations to exchange elite solutions that are used to update the pheromone matrix of the receiving colony.

Synchronization involved the exchange of not only good solutions but also of important search parameters in the pC/RS/MPDS parallel iterated tabu search proposed for the vehicle routing problem (VRP) [30]. The iterated tabu solvers started from different initial solutions and used different search parameters. They synchronized based on the number of consecutive iterations without improvement used to determine the stopping moment of the individual improvement phases. This provided the means to more equally distribute the work among cooperating processes. The solvers exchanged their best solutions, each solver probabilistically selecting the working solution for the next improvement phase among the received ones and its own. This method proved to be both flexible and efficient for several classes of routing problem settings with several depots, periodicity of demands, and time windows.

Most studies cited above compared several parallel strategies for the metaheuristic and problem setting at hand [27–29,41,42,101–104,107]. They contributed to show that synchronous pC/KS strategies with global information exchanges outperform independent search approaches, as well as the respective sequential version, particularly with respect to solution quality. These studies also pointed out, however, the benefit of dynamically determined synchronization points, as well as the superiority of asynchronous communications.

Diffusion

The previous strategies are based on global exchanges of information, gathered at synchronization points during the computation and distributed to all search threads. The interest of global information-sharing strategies resides in the best information available at the synchronization moment being available to all the solvers involved in cooperation. The main drawback results from this same characteristic, however, as solvers relying heavily on the same information (a set of best solutions in most cases) tend to focus on the same regions of the search space. This generally results in a search lacking in diversity that, more often than not, proves inefficient.

Synchronized cooperation strategies based on *diffusion* of information through local exchanges among "neighboring" solvers have therefore been proposed. Such approaches are defined on sparse communication graphs displaying particular topologies, such as ring, torus, or grid graphs, where each node is directly linked to only a few other nodes. A solver is then a neighbor of another solver if there is a direct link between the two nodes on which they run, that is, if their nodes are adjacent in the communication graph.

Synchronization still means that all solvers stop and exchange information, but here they perform it with their neighbors exclusively. Consequently, the quantity of information each solver processes and relies upon is significantly reduced, while the exchanges between nonadjacent solvers are performed at the speed of diffusion through possibly several chains of local exchanges and data modifications.

This idea has been much less explored compared to the global-exchange strategy, even though synchronous cooperative mechanisms based on local exchanges and diffusion have a less negative impact on the diversity of the search-space exploration. A number of applications were proposed with good results for coarse-grained [19,161] and fine-grained [3,63,68,69,117,118] genetic-based evolutionary methods, as well as for ant colony optimization [113].

Cooperation based on asynchronous information sharing generally outperforms synchronous methods, however, and is the topic of the next subsection.

Asynchronous Cooperation

Asynchronous strategies largely define the "state of the art" in parallel multi-search meta-heuristics. Solvers initiate asynchronous cooperation activities according to their own design logic and current internal state only, without coordination with other solvers or memories. Information sharing then proceeds either by direct inter-solver exchanges or through a data repository structure. These strategies belong to either the pC/C, described in this section, or the pC/KC, described in the next section, collegial classes of the taxonomy, the latter using the shared information to generate new knowledge and global search guidance.

Two main benefits are obtained when relying on asynchronous communications. First, this provides the means to build cooperation and information sharing among solvers without incurring the overheads associated to synchronization. Second, it increases the adaptability of cooperative meta-heuristics, as their capability to react

and dynamically adapt to the exploration of the search space by the cooperating solvers is significantly increased compared to the other parallelization strategies. Of course, these benefits come with potential issues one must care for. For example, the information gathered during the search will seldom, if ever, be completely available when a process must decide. Also, too frequent data exchanges, combined to simple acceptance rules for incoming information, may induce an erratic solver behavior, the corresponding search trajectories becoming similar to random walks. Hence the interest for applying information-sharing quality, meaningfulness, and parsimony principles [42, 43, 162].

In the basic asynchronous strategies discussed in this section, the shared information generally corresponds to a locally improving solution or individual(s). Most successful implementations have their cooperating solvers send out new local optima only. This limits the quantity of information sent and received, as well as the amount of work required to process it. Moreover, it avoids the case where a solver reorients its search based on one of a series of improving solutions and ends up developing a trajectory similar to the one followed by the solver that originated the data.

The abovementioned principles also explain the interest in diversifying the shared information [42]. Thus, always selecting the best available solution out of an elite set of good solutions, sent by potentially different solvers, proved less efficient in terms of quality of the final solution than a strategy that randomly, biased by quality, selected among the same elite set.

Finally, when to initiate cooperation activities and how to use incoming information is particular to each type of meta-heuristic involved in the cooperation. Yet, common to most strategies proposed in the literature is to perform jointly the sending and requesting of information. There is no absolute need to do this, however, even though it might decrease the amount of communication work. It might thus be interesting for neighborhood-based methods to make available right away their newly found local optima or improved overall solution and not wait for the algorithmic step where examining external information is appropriate. Similarly, population-based methods could migrate a number of individuals when a significant improvement is observed in the quality and diversity of their elite group of individuals.

With respect to when to request external information, the parsimony principle implies selecting only moments when the status of the search changes significantly, such as when the best solution or the elite subpopulation did not improve for a number of iterations.

The solver then engages into a so-called *search-diversification* phase, e.g., diversification in tabu search, change of neighborhood in variable neighborhood search, and complete or partial regeneration of population in population-based meta-heuristics, involving the choice or modification of the solution to initiate the new phase. Examining the contribution of external information is appropriate in this context. Notice that it is always possible to use simply a prefixed number of iterations to initiate communications, but this approach should be restricted to meta-heuristics without search-diversification steps, e.g., tabu search based on continuous diversification.

Direct-exchange strategies are generally implemented over a complete communication graph, each solver sending out information to all other solvers or to a subset of them; this subset may be predefined or selected dynamically during the search. Particular communication graphs and information-diffusion processes could also be used but, despite encouraging results, too few experiments have been reported yet (e.g., [146] proposing VNS pC/C strategies over uni- and bidirectional ring topologies). Each solver was executing the basic VNS steps and, on competing them, was passing its solution to its next neighbor (uni) or its two neighbors (bi), while receiving a solution from its predecessor neighbor (uni) or its two neighbors (bi). The received solution was kept as initial solution of the next VNS run in the unidirectional case, while the best of the two received ones and the local one was kept in the bidirectional ring implementation. The latter strategy proved the most successful.

Information exchanges within pC/C strategies based on indirect communications are generally performed through a data repository structure, often called *central memory* [32, 42, 43]. A solver involved in such a cooperation deposits (sends) good solutions, local optima generally, into the central memory, from where, when needed, it also retrieves information sent by the other cooperating solvers. The central memory accepts incoming solutions for as long as it is not full, acceptance becoming conditional to the relative interest of the incoming solution compared to the "worst" solution in the memory, otherwise. Evaluation is performed using the evaluation function for the global search space (or the objective function of the original problem). Diversity criteria are increasingly considered is this evaluation, a slightly worst solution being preferred if it increases the diversity of solutions in the central memory. Population culling may also be performed (deleting, e.g., the worst half the solutions in memory).

Both approaches may be applied to any meta-heuristic but, historically, most pC/C genetic-based evolutionary asynchronous cooperative meta-heuristics implemented a coarse-grained island model with direct inter-solver exchanges. An early comparative study of coarse-grained parallel genetic methods for the graph partitioning problem numerically showed the superiority of the pC/C strategy (with migration toward a subset of populations) over synchronous approaches [107].

The indirect-exchange communication model is found at the core of most asynchronous cooperative search strategies outside the genetic-evolutionary community, including simulated annealing for graph partitioning [101–104] and the TSP [143] and VNS applied to the VRPTW [127] and the p-median problem [44]. A master process was associated to the central memory in the latter method, which kept, updated, and communicated the current overall best solution (it also initiated and terminated the algorithm). Individual solvers proceeded with the VNS exploration for as long as the solution was improved. When this was no longer the case, the current best was communicated to the master (if better than the one at the last communication), and the overall best solution was requested from it. The best solution between the local and imported ones was selected, and the search was then continued in the current (local) neighborhood. Computational results on TSPLIB problem instances with up to 11,849 customers showed that the cooperative strategy

yielded significant gains in terms of computation time without losing on solution quality.

Apparently, [42] proposed the first central memory asynchronous tabu search. The tabu search solvers addressed a multicommodity location problem with balancing requirements. Each solver sent to the memory its local-best solution when improved and imported a probabilistically selected (rank-biased) solution from the memory before engaging in a diversification phase. This method outperformed in terms of solution quality the sequential version, several synchronous variants, and a broadcast-based asynchronous pC/C cooperative strategy. The same approach was applied to the fixed cost, capacitated, multicommodity network design problem with similar results [35]. Similar approaches were proposed for a broad range of problem settings, including the partitioning of integrated circuits for logical testing [1], two-dimensional cutting [13], the loading of containers [15], labor-constrained scheduling [22], the VRPTW [99], and the capacitated VRP [93].

Solvers involved in pC/C strategies may not be restricted to a single metaheuristic. Thus, the solvers in the two-phase approach of [75–77,91] for the VRPTW first applied an evolution strategy to reduce the number of vehicles, followed by a tabu search to minimize the total distance traveled. A different version of the same idea may be found in [10] for the Steiner problem, where each phase of the two phase is designed as a pC/C asynchronous central memory strategy, only the change from one phase to the next being synchronized. Solvers run reactive tabu search and path relinking meta-heuristics in the first and second phases, respectively.

Multilevel cooperative search proposes a different pC/C asynchronous cooperative strategy based on controlled diffusion of information [165]. Solvers are arrayed in a linear, conceptually vertical, communication graph, and a local memory is associated to each. Each solver works on the original problem but at a different level of aggregation (the solver on the conceptually first level works on the complete original problem) and communicates exclusively with the solvers directly above and below, that is, at higher and lower aggregation levels, respectively. The local memories are used to send information to the immediate neighbors and to access the incoming data from the same, at moments dynamically determined according to the internal logic of the respective solver. In the original implementation, solvers were exchanging improved solutions, incoming solutions not being transmitted further until modified locally for a number of iterations to enforce the controlled diffusion of information. Excellent results have been obtained for various problem settings including graph and hypergraph partitioning problems [122, 123], network design [47], feature selection in biomedical data [121], and covering design [53]. It is noteworthy that one can implement multilevel cooperative search using a central memory by adequately defining the communication protocols. Although not yet fully defined and tested, this idea is interesting as it opens the possibility of richer exchange mechanisms combining controlled diffusion and general availability of global information.

The central memory pC/C asynchronous cooperation strategy has proved worthy by several criteria. It yields high-quality solutions and is computationally efficient as no overhead is incurred for synchronization. It also helps to address the issue

of "premature convergence" in cooperative search, by diversifying the information received by the participating solvers through probabilistic selection from the memory and by a somewhat large and diverse population of solutions in the central memory (solvers may thus import different solutions even when their cooperation activities are taking place within a short time span).

The performance of central memory cooperation and the availability of exchanged information (kept in the memory) have brought the question of whether one could design more advanced cooperation mechanisms taking advantage of the information exchanged among cooperating solvers. The pC/KC strategies described in the next section are the result of this area of research.

Advanced Cooperation Strategies: Creating New Knowledge

Cooperation, in particular, memory-based asynchronous cooperation, offers a rich framework to combine solvers of different meta-heuristic and exact types, together with a population of elite solutions of continuously increased quality. But, is the development effort worthwhile?

An interesting proof of concept is found in the study of Crainic and Gendreau [34] combining a genetic-method solver and an asynchronous pC/C tabu search for multicommodity location-allocation with balancing requirements [42]. The tabu search solvers were aggressively exploring the search space, building the elite solution population in the central memory. The genetic method initialized its population with the one in the central memory once it contained a certain number of solutions. Its aim was to create new solutions to hopefully enrich the quality and diversity of the solutions exchanged among the cooperating solvers. Asynchronous migration transferred the best solution of the genetic population to the central memory, as well as solutions from the central memory toward the genetic population. This strategy did perform well, especially on larger instances. Most importantly, it showed that, while the overall best solution was never found by the genetic solver, the GA-enhanced cooperation yielded higher-quality solutions compared to the cooperation involving the tabu searches only. It appeared that the newly created solutions offered a more diverse set of diversification opportunities to the tabu search solvers, translating into a more diverse global search yielding better solutions.

The conclusion of that paper was not only that it is worthwhile to involve solvers of different types in the cooperation but also that it is beneficial to create new solutions out of the set of elite solutions in the central memory. The new solutions are different from their parent solutions and are added to the central memory if they improve compared to them. The process is thus doubly beneficial as better solutions in the memory directly enhance the quality of the global search, while increasing the diversity of solutions in memory provides the opportunity for cooperating solvers to explore new regions of the search space.

A second idea on developing advanced cooperation mechanisms concerns the information that may be extracted from the exchanged solutions, and the context information, eventually. It has thus been observed that optimal or near-optimal

solutions are often similar in the values taken by a large number of variables. Moreover, it is well known in the meta-heuristic field that one can learn from the characteristics of the solutions generated during the search, out of the best ones in particular, and use this learning to guide the search (see, e.g., the studies on memory and learning performed for tabu search [82]). Applied to cooperative search, it appeared promising to apply these learning techniques to the elite solutions in the population gradually built in the central memory and to use the resulting information to guide the search performed by the cooperating solvers.

Asynchronous cooperative strategies that include mechanisms to create new solutions and to extract information out of the exchanged solutions make up the p-control knowledge collegial (pC/KC) class. In most developments in this field, cooperating solvers work on the complete problem formulation and data. A recent research trend addresses rich multi-attribute problem settings and proposes pC/KC strategies where different solvers work on particular parts of the initial problem or on integrating the resulting partial solutions into complete ones. The next subsections describe these two cases.

pC/KC with Solvers Working on the Complete Problem

Two main classes of pC/KC cooperative mechanisms are found in the literature differing in the information that is kept in memory. *Adaptive memory* methods store partial elements of good solutions [142], while complete ones are kept in *central memory* methods [32, 38, 42]. The latter method generalizes the former and the vocabulary used in the various papers notwithstanding the two approaches is becoming increasingly unified.

The *adaptive memory* terminology was coined by Rochat and Taillard [142] (see also [81, 153, 154]). The method was targeting the VRP and the VRPTW, and it marked a changing point in the state of the art at the time. The main idea was to keep in the memory the individual components (vehicle routes in the initial application) of the solutions produced by the cooperating solvers (tabu search methods in [142]). Two types of information were recorded for each solution element kept in memory, a frequency counter of its appearance in the best solutions encountered so far, and its rank within the elite population in memory based on the characteristics (mainly the objective function value) of the solution from which it was extracted. Solvers constructed new complete solutions out of randomly (rank-biased) selected partial elements, improved these new solutions, and returned the best ones to the memory. The rank-biased random selection of elements assured that the new solution was composed in most cases of routes from different elite solutions, thus inducing a powerful diversification effect.

Several interesting developments were proposed, and conclusions were drawn within the context of successful adaptive memory pC/KC algorithms. A set-covering heuristic was thus proposed as selection mechanism for the elements (VRPTW routes) used by solvers to generate new initial solutions [145]. This mechanism proved very successful and has been used several times since (e.g., [87]). A

two-level parallel scheme was proposed for the real-time vehicle routing and dispatching [79]. A pC/KC/MPSS cooperating adaptive memory method made up the first level, while the slave processors attached to each solver, a tabu search method based on route decomposition [151], made up the second level. The performance of this method is noteworthy also because, while many papers mention the possibility of hierarchical parallel schemes, very few actual implementations are found in the literature. Equally for the VRPTW, the adaptive memory approach of [6] yielded a number of interesting findings relative to the implementation of cooperative methods. Thus, when individual solvers are fast, as is generally the case for routing problems, it is beneficial to run several solvers on the same processor and group the exchanges with the central adaptive memory (avoiding or reducing access bottlenecks to the latter). On the other hand, running the memory and solvers on the same processor is to be avoided (the solver execution reduces the response efficiency of the memory).

Solvers in *central memory* methods indirectly exchange complete elite solutions and context information through the central memory data repository device. Solvers may include constructive, improving, and post-optimization heuristics (e.g., [99, 100]), neighborhood (e.g., tabu search [57, 93, 94]) and population-based methods (e.g., genetic algorithms [57, 99, 100], and path relinking [46]), as well as exact solution methods, on restricted versions of the problem, eventually. The particular solvers to include depend on the particular application. They should be efficient for the problem at hand. They should also contribute to build and enhance solutions that may contribute to improve both the quality and the diversity of the elite population being built in the central memory.

The central memory keeps full solutions, solution attributes and context information, both received from the solvers and newly created out of the exchanged information. To more clearly distinguish between the data warehousing and the information creating functions of central memory mechanisms, let the *search coordinator* (*SC*) be the process in charge of the latter function. The simplest version of the SC was used in the pC/C strategies of the previous section, where solutions in memory were ordered (generally according to the value of the objective function) and rank-biased randomly extracted to answer solver requests. The functions of the SC in pC/KC methods include creating new solutions; extracting appropriate solution elements; building statistics on the presence and performance of solutions, solution elements, and solvers (these belong to the family of memories well known in the meta-heuristic community); and creating the information to return when answering solver requests (the latter are the so-called *guidance mechanisms*).

The cooperative meta-heuristic proposed by [99] for the VRPTW used a simple pC/KC mechanism. Four solvers, two simple genetic algorithms with order and edge recombination crossovers, respectively, and two tabu search methods that perform well sequentially, the unified tabu [31] and TABUROUTE [78]. The solvers sent their improved best solutions to the central memory and requested solutions from the same when needed (the genetic algorithms for crossover operations, at regular intervals for the unified tabu, and at diversification time for TABUROUTE). Besides ordering and selecting the solutions to return, the SC was only performing

post-optimization (2-opt, 3-opt, or-opt, and ejection-chain procedures to reduce the number of vehicles and the total traveled distance) on the received solutions. Without any calibration or tailoring, this algorithm proved to be competitive with the best meta-heuristics of its day in linear speedups.

A more complete SC was proposed in [100] also for the VRPTW. The goal was for a guidance mechanism that, first, extracted and returned to solvers meaningful information in terms of individual guidance and global search performance and, second, was independent of problem characteristics, routes in particular, and could be broadly applied to network-based problem settings. To work toward the second goal, the SC mechanism targeted an atomic element in network optimization, the arc. The basic idea of the SC mechanism was that an arc that appears often in good solutions and less frequently in bad solutions may be worthy of consideration for inclusion in a tentative solution, and vice versa. To implement this idea, the authors considered the evolution of the "appearance" of arcs in solutions of different qualities. Appearance was measured by means of frequencies of inclusion of arcs in the elite (e.g., the 10 % best), average (between the 10 % and 90 % best), and worst (the last 10 %) groups of solutions in the central memory. *Patterns* of arcs were then defined representing subsets of arcs (not necessarily adjacent) with similar frequencies of inclusion in particular population groups. Guidance was obtained by transmitting arc patterns to the individual solvers indicating whether the arcs in the pattern should be "fixed" or "prohibited" to intensify or diversify the search, respectively. The solvers accounted for the "fix" and "prohibit" instructions by using the patterns to bias the selection of arcs for move or reproduction operations. A four-phase fixed schedule (two phases of diversification at the beginning to broaden the search, followed by two intensification phases to focus the search around promising regions) was used (see [98] for a dynamic version of this mechanism). Excellent performances in terms of solution quality and computing efficiency were observed compared to the best performing methods of the day.

A different SC was proposed in [94] for the capacitated VRP with tabu search solvers. Solvers periodically (after a number of iterations or when the solution has not been improved for a number of iterations) sent best solutions to central memory, and received a solution back from it, the search being resumed from the received solution. The SC mechanism aimed to identify and extract information from the solutions in memory to guide solvers toward intensification and diversification phases. This was obtained by clustering solutions, dynamically when solutions were received, according to the number of edges in common. Thus, solutions in a given cluster have a certain number of edges in common, this cluster of edges and solutions being assumed to represent a region of the search space. Search history indicators were also associated to clusters giving the number of solutions in the cluster and the quality of the solutions. This information was used to infer how thoroughly the corresponding region had been explored and how promising it appeared. Clusters were actually sorted according to the average solution value of their feasible solutions. The cluster with the lowest average value, that is, with a largest number of very good solutions, was selected for intensification, while the solution with the lowest number of good solutions was selected for diversification.

A solution was selected in the corresponding cluster, and it was sent to the requesting solver. Excellent results were obtained in terms of solution quality and computation effort (an almost linear speedup was observed up to 240 processors) compared to the state-of-the-art methods of the day (including the parallel method of [87]).

A pC/KC/MPDS method proposed in [87] for the VRP demonstrates how specialized solvers may address different issues in a cooperative meta-heuristic, including the generation of new knowledge. Two types of solvers were defined in this scheme. The so-called heuristic solvers improved solutions received from the SC associated to the central memory (called master in [87]), through a record-to-record meta-heuristic [26, 83, 106]. On completing the task, solvers returned both a number (50) of the best solutions found and the corresponding routes (a post-optimization procedure was first run on each route). Simultaneously, set-covering solvers aimed to identify new solutions by solving a series of set-covering problems starting from a limited set of routes. Each time a set-covering problem was solved, the solution was returned to the central memory and the set of the current ten best solutions was retrieved for the next run. Set-covering solvers had also access to the ordered list of best routes in memory, and they selected within to complete their problems. The number of routes admitted to set up a set-covering problem was dynamically modified during the search to control the corresponding computational effort. The SC kept and ordered the received solutions and routes and selected the solutions to make available to solvers (routes were always available; an efficient file system was used to facilitate access to this data). The method performed very well, both in terms of solution quality and computational effort (an almost linear speedup was observed).

The contributions described in this section emphasize the interest of asynchronous knowledge-generating cooperative meta-heuristics. The cooperation and guidance mechanisms, as well as the role of learning and statistical performance data, require additional and systematic studies, preferably on a broader range of problem settings. The contributions aimed at addressing multi-attribute problem settings are described in the next subsection.

pC/KC with Partial Solvers: The Integrative Cooperative Search

The versatility and flexibility of the central memory concept has raised the interest in generalizing it to address so-called *rich* combinatorial optimization problems displaying a large number of attributes characterizing their feasibility and optimality structures. The general idea is to decompose the initial problem formulation along sets of decision variables, called *decision-set attribute decomposition* in [96], yielding simpler but meaningful problem settings, in the sense that efficient solvers, can be "easily" obtained for these partial problems either by opportunistically using existing high-performing methods or by developing new ones. The central memory cooperative search framework then brings together these partial problems and their associated solvers, together with integration mechanisms, reconstructing complete solutions, and search-guidance mechanisms.

The first effort in this direction is probably the work of [46] (see also [57]) for the design of wireless networks, where seven attributes were considered simultaneously. The proposed pC/KC/MPDS cooperative meta-heuristic had tabu search solvers work on limited subsets of attributes only, while a genetic method amalgamated the partial solutions sent by the tabu search solvers to the central memory, into complete solutions to the initial problem.

The general method, called *integrative cooperative search ICS*) by its authors, has been fully defined in [96] (see also [48,49]). The brief presentation that follows describes ICS according to [96] through an application to the multi-depot periodic vehicle routing problem (MDPVRP), which simultaneously decides on (1) selecting a visit *pattern* for each customer, specifying the particular periods the customer is to be visited over the multi-period planning horizon, and (2) assigning each customer to a depot for each visit [115, 169].

The main components of ICS, which must be instantiated for each application, are the (1) decomposition rule; (2) *partial solver groups* (*PSGs*) addressing the partial problems resulting from the decomposition; (3) *integrators*, which select partial solutions from PSGs, combine them to create complete ones, and send them to the *complete solver group* (*CSG*); and (4) the CSG, which corresponds to the central memory of ICS and has as prime function to manage the pool of complete solutions and the context information received from the PSGs and integrators and to extract out of these the information required to guide the partial and global searches. Guidance is performed by the *global search coordinator* (*GSC*) associated to the CSG. Notice that, in order to facilitate the cooperation, a unique solution representation is used throughout ICS. This representation is obtained by fixing rather than eliminating variables when defining partial problems.

The selection of the decision sets is specific to each application case, decision variables being clustered to yield known or identifiable optimization problem settings. An opportunistic selection decomposes the MDPVRP along the depot and period decision sets to create two partial problems. Thus, fixing the customer-to-depot assignments yields a periodic VRP (PVRP), while fixing the patterns for all customers yields a multi-depot VRP (MDVRP). High-quality solvers exist in the literature for both problems.

Two PSGs were defined for the partial problems, one for the PVRP and the other for the MDVRP. Each PSG was organized according to the pC/KC paradigm and was thus composed of a set of *partial solvers*, a central memory where elite solutions were kept, and a *local search coordinator* (*LSC*) managing the central memory and interfacing with the global search coordinator.

Two algorithms were used in the implementation described in [96] for both complete and partial solvers, the HGSADC of [169] and GUTS, a generalized version of the unified tabu search [31]. Briefly, HGSADC combines the exploration capability of population-based evolutionary search, the aggressive-improvement strength of neighborhood-based local search to enhance solutions newly created by genetic operators, and a solution evaluation function driven by both solution quality and contribution to the population diversity, which contributes to progress toward diverse and good solutions. GUTS is a tabu search-based meta-heuristic

implementing advanced insertion neighborhoods and allowing the exploration of unfeasible solutions by dynamically adjusting penalties on violations of vehicle capacity and route duration constraints. Both methods use relaxation of vehicle capacity and route duration constraints combined to penalization of infeasibilities in the evaluation function. They also use well-known VRP local neighborhoods based on pattern change, depot change, and inter- and intra-route movements.

Integrators build complete solutions by mixing partial solutions with promising features obtained within the PSGs. Integrators aim for solution quality, the transmission of critical features extracted from the partial solutions, and computational efficiency. Several Integrators can be involved in an ICS implementation, contributing to these goals and increasing the diversity of the population of complete solutions.

The simplest integrator consists in selecting high-quality partial solutions (with respect to solution value or the inclusion of particular decision combinations) and passing them directly to the complete solver group. Meta-heuristics, population-based methods in particular, e.g., genetic algorithms [169] and path relinking [131], may also be used, having proved their flexibility and stability in combining solution characteristics to yield high-quality solutions. Finally, a new methodology was proposed recently [65]. It proceeds through particular optimization models that preserve desired critical variables, defined as the variables whose values in the respective solution represent desired attributes, present in the partial solutions.

Four integrators were included in the MDPVRP application, the simple one passing good solutions to the CSG, and three others starting from pairs of partial solutions randomly selected among the best 25 % of the solutions in the central memories of the two PSGs. The second integrator applied the crossover operator of HGSADC and enhanced the new solution through the local search education operator of the same method. The third and fourth integrators applied the methodology of [65], the former aiming to transmit the attributes for which there was "consensus" in the input solutions, while the latter "promoted" them only through penalties added to the objective function.

The complete solver group (CSG) included a central memory, which included the complete solution set, as well as the context information and the guiding solutions built by the global search coordinator (GSC). The CSG received complete solutions from integrators and, when solvers were included (e.g., GUTS and HGSADC in the present case), enhanced them thus creating new ones. It was the global search coordinator which (1) built the search contextual information (e.g., the frequency of appearance of each (customer, depot, pattern) triplet in the complete solution set, together with the cost of the best solution containing it), (2) built new guiding solutions to orient the search toward promising features, and (3) monitored the status of the solver groups, sending guiding instructions (solutions) when necessary.

Monitoring is performed by following the evolution of the PSGs by, e.g., interrogating the central memories of the PSGs for the number of improving solutions generated during a certain time period. Monitoring provides the means to detect undesired situations, e.g., loss of diversity in the partial or complete populations, stagnation in improving the quality of the current best solution, awareness that some

zones of the solution space – defined by particular values for particular decision sets – have been scarcely explored, if at all, and that the search should be diversified in that direction, and so on. Whenever one of these criteria is not fulfilled, the GSC sends guidance "instructions" to the particular PSG. The particular type of guidance is application specific, but one may modify the values of the fixed attributes for the PSG to orient its search toward a different area or, more rarely, change the attribute subset under investigation (i.e., change the decomposition of the decision-set attributes) or modify/replace the solution method in a partial solver or integrator.

In the present case, the GSC guided the search trajectory of a particular PSG by sending three solutions, which were either randomly selected (equiprobably) from the complete solution set, or were three *guiding* solutions built by the GSC. The receiving PSG added directly these solutions to its own central memory, after resetting its population, all solutions being replaced by new randomly generated ones. Guiding solutions were continuously generated, and stored in a particular pool, to reflect the current status and the history of the search represented by the context information. The process proceeded by selecting promising triplets with respect to the search history, that is, triplets that appeared in at least one complete solution with a cost close (less than 3 % distant) to the current best solution. The promising triplets were used to create feasible pattern and depot customer assignments, routes being then generated by the local search of HGSADC to complete the solutions. These solutions were then individually enhanced by a short execution of GUTS or HGSADC.

Extensive experimental analyses were conducted to (1) assess the performance of ICS when compared to state-of-the-art sequential methods and (2) investigate a number of implementation alternatives. The general conclusions were that ICS performed extremely well. It obtained very good results even when compared to the state-of-the-art HGSADC meta-heuristic, obtaining several new best-known solutions in shorter computing times. The experiments also indicated that (1) one should use solvers displaying similar time performances in order to have all solvers contributing reasonably equally to the cooperation; (2) when using genetic solvers in a PSG, it is preferable for long runs to define a local population for each such solver and reserve the central memory of the PSG for communications and guidance only, while using the central memory as population for all cooperating genetic solvers is better for short runs; and (3) embedding good solvers (HGSADC in the present case) in the CSG enhances slightly the already excellent performance of the ICS parallel meta-heuristic.

Conclusions

This chapter presented an overview and state-of-the-art survey of the main parallel meta-heuristic ideas, discussing general concepts and algorithm design principles and strategies. The presentation was structured along the lines of a taxonomy of parallel meta-heuristics, which provided a rich framework for analyzing these

design principles and strategies, reviewing the literature, and identifying trends and promising research directions.

Four main classes of parallel meta-heuristics strategies may be identified: low-level decomposition of computing-intensive tasks with no modification to the original algorithm, decomposition of the search space, independent multi-search, and cooperative (multi) search, the later encompassing synchronous, asynchronous collegial, and knowledge-creating asynchronous collegial. It is noteworthy that this series also reflects the historical sequence of the development of parallel meta-heuristics. One should also note that, while the initial developments targeted genetic methods, simulated annealing, and tabu search, research is not addressing the full range of meta-heuristics. Furthermore, parallel meta-heuristics, cooperative search in particular, are now acknowledged as making up their own class of meta-heuristics.

Many important research questions and challenges exist for parallel meta-heuristics, in terms of general design methodology, instantiation to particular meta-heuristic frameworks and problem settings, and implementation on various computing architectures.

It is indeed noteworthy that despite the many years of research on these issues, there are still many gaps in knowledge, as well as in the studied meta-heuristic frameworks and problem classes. One may single out the many variants of swarm-based optimization and nongenetic population-based methods, scatter search and path relinking in particular. But one should not overlook the more classic meta-heuristic classes, as one still misses systematic and comprehensive/comparative studies of these issues. A large part of the studies present in the literature targeted combinatorial optimization problems with relatively few attributes and a single level of decision variables, e.g., vehicle routing problems. This is to be understood, these problems being important for science and practice and displaying large search spaces. Significant less research has been dedicated to multi-attribute problem settings, like the rich VRPs one increasingly has to tackle, and formulations with several levels of decisions like the single- and multilevel network design.

The community misses not only studies targeting particular meta-heuristic frameworks and problem classes but also transversal studies comparing the behavior and performance of particular parallel meta-heuristic strategies over different problem classes and of different parallel strategies and implementations for the same problem class. One increasingly finds such studies for sequential solution methods; we need them for parallel methods.

With respect to the four strategy classes, one should not forget that each fulfills a particular type of task and all are needed at some time. Thus, the idea that everything seems to be known regarding low-level parallelization strategies is not true. First, most studies on accelerating computing-intensive tasks targeted the evaluation of a population or neighborhood in classic meta-heuristic frameworks. These techniques should prove very valuable for swarm-based optimization, and more research is required in this field. Second, as shown in recent studies, the best strategy to accelerate a local search procedure may prove less effective when the local search is embedded into a full meta-heuristics or hierarchical solution methods. Third, the evolution of computing infrastructure opens up interesting but

challenging perspectives. Let's emphasize the possibilities offered by the graphic processing units, which increase continuously in power and are present everywhere, as surveyed in [16, 17].

Search-space decomposition also seems to have been thoroughly studied and has been overlooked in the last years, maybe due to the rapid and phenomenal increase in the memory available and the speed of access. Let's not forget, however, that most optimization problems of interest are complex and that the dimensions of the instances one faces in practice keep increasing. Research challenges exist in dynamic search-space decomposition and the combination of cooperative search and search-space decomposition. The integrative cooperative search is a first answer in this direction, but more research is needed.

Asynchronous cooperation, particularly when relaying on memories as inter-solver communication mechanisms, provides a powerful, flexible, and adaptable framework for parallel meta-heuristics that consistently achieved good results in terms of computing efficiency and solution quality for many meta-heuristic and problem classes. Other than the general research issues discussed above that are of particular interest in this context, a number of additional research issues and challenges are worth investigating.

A first issue concerns the exchange and utilization of context data locally generated by the cooperating solvers, to infer an image of the status of the global search and generate appropriate guiding instructions. Thus, contrasting the various local context data may be used to identify regions of the search space that were neglected or over explored. The information could also be used to evaluate the relative performance of the solvers conducting, eventually, to adjust the search parameters of particular solvers or even change the search strategy. So-called "strategic" decision variables or parameters could thus be more easily identified, which could prove very profitable in terms of search guidance.

A related issue concerns the learning processes and the creation of new information out of the shared data. Important questions concern the identification of information that may be derived from the exchanged solutions and context information and which is meaningful for, on the one hand, evaluating the status of the global search and, on the other hand, sending to solvers to guide their own search as part of the global optimization effort. Research in this direction is still at the very beginning but has already proved its worth, in particular in the context of the integrative cooperative methods.

A third broad issue concerns the cooperation of different types of meta-heuristics and of these exact solution methods. The so-called hybrid and matheuristic methods, representing the former and latter types of method combination, respectively, are trendy in the sequential optimization field. Very few studies explicitly target parallel methods. How different methods behave when involved in cooperative search and how the latter behaves given various combinations of methods is an important issue that should yield valuable insights into the design of parallel meta-heuristic algorithms, integrative cooperative search in particular. Actually, more research is required into ICS, both regarding its structure and components, and its application to various problem settings. A particularly challenging but fascinating direction for

cooperative search and ICS is represented by the multi-scenario representation of stochastic optimization formulations, for which almost nothing beyond low-level scenario decomposition has been proposed.

Finally, the issue of understanding cooperation on some fundamental level, giving the means to formally define and analyze it in order to design better, more efficient algorithms. As mentioned earlier, this work parallels efforts in many other scientific domains addressing issues related to emerging decision and behavior out of the decisions and behaviors or several independent entities. Theoretical and empirical work is needed in order to address this fascinating and difficult question.

Acknowledgments The author wishes to acknowledge the contributions of colleagues and students, in particular Professors Michel Gendreau, Université de Montréal, Canada, and Michel Toulouse, the Vietnamese-German University, Vietnam, who collaborated over the years to the work on parallel meta-heuristics for combinatorial optimization. All errors are solely and entirely due to the author, however.

Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant and the Discovery Accelerator Supplements programs, and the Strategic Clusters program of the Fonds québécois de la recherche sur la nature et les technologies. The author thanks the two institutions for supporting this research.

References

1. Aiex RM, Martins SL, Ribeiro CC, Rodriguez NR (1998) Cooperative multi-thread parallel tabu search with an application to circuit partitioning. In: Proceedings of IRREGULAR'98 – 5th international symposium on solving irregularly structured problems in parallel. Lecture notes in computer science, vol 1457. Springer, Berlin/New York, pp 310–331
2. Alba E (ed) (2005) Parallel metaheuristics: a new class of algorithms. Wiley, Hoboken
3. Alba E, Dorronsoro B (2004) Solving the vehicle routing problem by using cellular genetic algorithms. In: Gottlieb J, Günther RR (eds) Evolutionary computation in combinatorial optimization, 4th European conference, EvoCOP 2004, Coimbra, 5–7 Apr 2004. Lecture notes in computer science, vol 3004. Springer, Heidelberg, pp 11–20
4. Alba E, Luque G, Nesmachnow S (2013) Parallel metaheuristics: recent advances and new trends. Int Trans Oper Res 20(1):1–48
5. Azencott R (1992) Simulated annealing parallelization techniques. Wiley, New York
6. Badeau P, Gendreau M, Guertin F, Potvin JY, Taillard E (1997) A parallel tabu search heuristic for the vehicle routing problem with time windows. Transp Res C: Emerg Technol 5(2): 109–122
7. Banos R, Gil C, Ortega J, Montoya FG (2004) A parallel multilevel metaheuristic for graph partitioning. J Heuristics 10(4):315–336
8. Banos R, Gil C, Ortega J, Montoya FG (2004) Parallel heuristic search in multilevel graph partitioning. In: Proceedings of the 12th Euromicro conference on parallel, distributed and network-based processing, A Coruña, pp 88–95
9. Barr RS, Hickman BL (1993) Reporting computational experiments with parallel algorithms: issues, measures, and experts opinions. ORSA J Comput 5(1):2–18
10. Bastos MP, Ribeiro CC (1999) Reactive tabu search with path-relinking for the Steiner problem in graphs. In: Voß S, Martello S, Roucairol C, Osman IH (eds) Meta-heuristics 98: theory & applications. Kluwer Academic, Norwell, pp 31–36
11. Battiti R, Tecchiolli G (1992) Parallel based search for combinatorial optimization: genetic algorithms and TABU. Microprocessors Microsyst 16(7):351–367

12. Berger J, Barkaoui M (2004) A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Comput Oper Res 31(12):2037–2053
13. Blazewicz J, Moret-Salvador A, Walkowiak R (2004) Parallel tabu search approaches for two-dimensional cutting. Parallel Process Lett 14(1):23–32
14. Bock S, Rosenberg O (2000) A new parallel breadth first tabu search technique for solving production planning problems. Int Trans Oper Res 7(6):625–635
15. Bortfeldt A, Gehring H, Mack D (2003) A parallel tabu search algorithm for solving the container loading problem. Parallel Comput 29:641–662
16. Brodtkorb AR, Hagen TR, Schulz C, Hasle G (2013) GPU computing in discrete optimization. Part I: introduction to the GPU. EURO J Transp Logist 2(1–2):129–157
17. Brodtkorb AR, Hagen TR, Schulz C, Hasle G (2013) GPU computing in discrete optimization. Part II: survey focussed on routing problems. EURO J Transp Logist 2(1–2):159–186
18. Bullnheimer B, Kotsis G, Strauß C (1999) Parallelization strategies for the ant system. In: De Leone R, Murli A, Pardalos P, Toraldo G (eds) High performance algorithms and software in nonlinear optimization. Applied optimization, vol 24, Kluwer Academic, Dordrecht, pp 87–100. http://www.bwl.univie.ac.at/bwl/prod/papers/pom-wp-9-97.ps
19. Calégari P, Guidec F, Kuonen P, Kuonen D (1997) Parallel Island-based genetic algorithm for radio network design. J Parallel Distrib Comput 47(1):86–90
20. Cantú-Paz E (1998) A survey of parallel genetic algorithms. Calculateurs Parallèles, Réseaux et Systèmes répartis 10(2):141–170
21. Cantú-Paz E (2005) Theory of parallel genetic algorithms. In: Alba E (ed) Parralel metaheuristics: a new class of algorithms. Wiley, Hoboken, pp 425–445
22. Cavalcante CBC, Cavalcante VF, Ribeiro CC, Souza MC (2002) Parallel cooperative approaches for the labor constrained scheduling problem. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer Academic, Norwell, pp 201–225
23. Chakrapani J, Skorin-Kapov J (1992) A connectionist approach to the quadratic assignment problem. Comput Oper Res 19(3/4):287–295
24. Chakrapani J, Skorin-Kapov J (1993) Connection machine implementation of a tabu search algorithm for the traveling salesman problem. J Comput Inf Technol 1(1):29–36
25. Chakrapani J, Skorin-Kapov J (1993) Massively parallel tabu search for the quadratic assignment problem. Ann Oper Res 41:327–341
26. Chao IM, Golden B L, Wasil EA (1995) An improved heuristic for the period vehicle routing problem. Networks 26(1):25–44
27. Cohoon J, Hedge S, Martin W, Richards D (1987) Punctuated equilibria: a parallel genetic algorithm. In: Grefenstette J (ed) Proceedings of the second international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, Hillsdale, pp 148–154
28. Cohoon J, Hedge S, Richards D (1991) Genetic algorithm and punctuated equilibria in VLSI. In: Schwefel H-P, Männer R (eds) Parallel problem solving from nature. Lecture notes in computer science, vol 496. Springer, Berlin, pp 134–144
29. Cohoon J, Hedge S, Richards D (1991) A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes. In: Belew R, Booker L (eds) Proceedings of the fourth international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 134–144
30. Cordeau JF, Maischberger M (2012) A parallel iterated tabu search heuristic for vehicle routing problems. Comput Oper Res 39(9):2033–2050
31. Cordeau JF, Laporte G, Mercier A (2001) A unified tabu search heuristic for vehicle routing problems with time windows. J Oper Res Soc 52:928–936
32. Crainic TG (2005) Parallel computation, co-operation, tabu search. In: Rego C, Alidaee B (eds) Metaheuristic optimization via memory and evolution: tabu search and scatter search. Kluwer Academic, Norwell, pp 283–302
33. Crainic TG (2008) Parallel solution methods for vehicle routing problems. In: Golden BL, Raghavan S, Wasil EA (eds) The vehicle routing problem: latest advances and new challenges. Springer, New York, pp 171–198

34. Crainic TG, Gendreau M (1999) Towards an evolutionary method – cooperating multi-thread parallel tabu search hybrid. In: Voß S, Martello S, Roucairol C, Osman IH (eds) Meta-heuristics 98: theory & applications. Kluwer Academic, Norwell, pp 331–344
35. Crainic TG, Gendreau M (2002) Cooperative parallel tabu search for capacitated network design. J Heuristics 8(6):601–627
36. Crainic TG, Hail N (2005) Parallel meta-heuristics applications. In: Alba E (ed) Parallel metaheuristics: a new class of algorithms. Wiley, Hoboken, pp 447–494
37. Crainic TG, Toulouse M (1998) Parallel metaheuristics. In: Crainic TG, Laporte G (eds) Fleet management and logistics. Kluwer Academic, Norwell, pp 205–251
38. Crainic TG, Toulouse M (2003) Parallel strategies for meta-heuristics. In: Glover F, Kochenberger G (eds) Handbook in metaheuristics. Kluwer Academic, Norwell, pp 475–513
39. Crainic TG, Toulouse M (2008) Explicit and emergent cooperation schemes for search algorithms. In: Maniezzo V, Battiti R, Watson J-P (eds) Learning and intelligent optimization. Lecture notes in computer science, vol 5315. Springer, Berlin, pp 95–109
40. Crainic TG, Toulouse M (2010) Parallel meta-heuristics. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, 2nd edn. Springer, New York, pp 497–541
41. Crainic TG, Toulouse M, Gendreau M (1995) Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. OR Spektrum 17(2/3):113–123
42. Crainic TG, Toulouse M, Gendreau M (1996) Parallel asynchronous tabu search for multicommodity location-allocation with balancing requirements. Ann Oper Res 63:277–299
43. Crainic TG, Toulouse M, Gendreau M (1997) Towards a taxonomy of parallel tabu search algorithms. INFORMS J Comput 9(1):61–72
44. Crainic TG, Gendreau M, Hansen P, Mladenović N (2004) Cooperative parallel variable neighborhood search for the p-median. J Heuristics 10(3):293–314
45. Crainic TG, Gendreau M, Potvin JY (2005) Parallel tabu search. In: Alba E (ed) Parallel metaheuristics. Wiley, Hoboken, pp 298–313
46. Crainic TG, Di Chiara B, Nonato M, Tarricone L (2006) Tackling electrosmog in completely configured 3G networks by parallel cooperative meta-heuristics. IEEE Wirel Commun 13(6):34–41
47. Crainic TG, Li Y, Toulouse M (2006) A first multilevel cooperative algorithm for the capacitated multicommodity network design. Comput Oper Res 33(9):2602–2622
48. Crainic TG, Crisan GC, Gendreau M, Lahrichi N, Rei W (2009) A concurrent evolutionary approach for cooperative rich combinatorial optimization. In: Genetic and evolutionary computation conference – GECCO 2009, Montréal, 8–12 July. ACM, cD-ROM
49. Crainic TG, Crisan GC, Gendreau M, Lahrichi N, Rei W (2009) Multi-thread integrative cooperative optimization for rich combinatorial problems. In: The 12th international workshop on nature inspired distributed computing – NIDISC'09, 25–29 May, Rome, cD-ROM
50. Crainic TG, Davidović T, Ramljak D (2014) Designing parallel meta-heuristic methods. In: Despotovic-Zrakic M, Milutinovic V, Belic A (eds) High performance and cloud computing in scientific research and education. IGI Global, Hershey, pp 260–280
51. Cung VD, Martins SL, Ribeiro CC, Roucairol C (2002) Strategies for the parallel implementations of metaheuristics. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer Academic, Norwell, pp 263–308
52. Czech ZJ (2000) A parallel genetic algorithm for the set partitioning problem. In: 8th Euromicro workshop on parallel and distributed processing, Rhodos, pp 343–350
53. Dai C, Li B, Toulouse M (2009) A multilevel cooperative tabu search algorithm for the covering design problem. J Comb Math Comb Comput 68:35–65
54. Davidović T, Crainic TG (2015) Parallel local search to schedule communicating tasks on identical processors. Parallel Comput 48:1–14
55. De Falco I, Del Balio R, Tarantino E, Vaccaro R (1994) Improving search by incorporating evolution principles in parallel tabu search. In: Proceedings international conference on machine learning, New Brunswick, pp 823–828

56. De Falco I, Del Balio R, Tarantino E (1995) Solving the mapping problem by parallel tabu search. Report, Istituto per la Ricerca sui Sistemi Informatici Paralleli-CNR
57. Di Chiara B (2006) Optimum planning of 3G cellular systems: radio propagation models and cooperative parallel meta-heuristics. PhD thesis, Dipartimento di ingegneria dell'innovatione, Universitá degli Studi di Lecce, Lecce
58. Diekmann R, Lüling R, Monien B, Spräner C (1996) Combining helpful sets and parallel simulated annealing for the graph-partitioning problem. Int J Parallel Program 8:61–84
59. Doerner K, Hartl RF, Kiechle G, Lucka M, Reimann M (2004) Parallel ant systems for the capacitated vehicle routing problem. In: Gottlieb J, Raidl GR (eds) Evolutionary computation in combinatorial optimization: 4th European conference, EvoCOP 2004. Lecture notes in computer science, vol 3004. Springer, Berlin, pp 72–83
60. Doerner KF, Hartl RF, Lucka M (2005) A parallel version of the D-ant algorithm for the vehicle routing problem. In: Vajtersic M, Trobec R, Zinterhof P, Uhl A (eds) Parallel numerics'05. Springer, New York, pp 109–118
61. Doerner KF, Hartl RF, Benkner S, Lucka M (2006) Cooperative savings based ant colony optimization – multiple search and decomposition approaches. Parallel Process Lett 16(3):351–369
62. Dorigo M, Stuetzle T (2003) The ant colony metaheuristic. Algorithms, applications, and advances. In: F Glover, G Kochenberger (eds) Handbook in metaheuristics. Kluwer Academic, Norwell, pp 251–285
63. Dorronsoro B, Arias F, Luna A, Nebro AJ, Alba E (2007) A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. In: Smari W (ed) High performance computing & simulation conference HPCS 2007 within the 21st European conference on modelling and simulation ECMS 2007, pp 759–765. http://www.scs-europe.net/conf/ecms2007/ecms2007-cd/ecms2007/ecms2007finalpapers.html
64. Drias H, Ibri A (2003) Parallel ACS for weighted MAX-SAT. In: Mira J, Álvarez J (eds) Artificial neural nets problem solving methods – proceedings of the 7th international work-conference on artificial and natural neural networks. Lecture notes in computer science, vol 2686. Springer, Heidelberg, pp 414–421
65. El Hachemi N, Crainic TG, Lahrichi N, Rei W, Vidal T (2014) Solution integration in combinatorial optimization with applications to cooperative search and rich vehicle routing. Publication CIRRELT-2014-40, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal
66. Fiechter CN (1994) A parallel tabu search algorithm for large travelling salesman problems. Discrete Appl Math 51(3):243–267
67. Flores CD, Cegla BB, Caceres DB (2003) Telecommunication network design with parallel multi-objective evolutionary algorithms. In: IFIP/ACM Latin America networking conference 2003, La Paz
68. Folino G, Pizzuti C, Spezzano G (1998) Combining cellular genetic algorithms and local search for solving satisfiability problems. In: Proceedings of the tenth IEEE international conference on tools with artificial intelligence. IEEE Computer Society Press, Piscataway, pp 192–198
69. Folino G, Pizzuti C, Spezzano G (1998) Solving the satisfiability problem by a parallel cellular genetic algorithm. In: Proceedings of the 24th EUROMICRO conference. IEEE Computer Society Press, Los Alamitos, pp 715–722
70. Garcia BL, Potvin JY, Rousseau JM (1994) A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. Comput Oper Res 21(9):1025–1033
71. García-López F, Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM (2002) The parallel variable neighborhood search for the p-median problem. J Heuristics 8(3):375–388
72. García-López F, Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM (2003) Parallelization of the scatter search for the p-median problem. Parallel Comput 29:575–589
73. García-López F, García Torres M, Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM (2005) Parallel scatter search. In: Alba E (ed) Parallel metaheuristics: a new class of metaheuristics. Wiley, Hoboken, pp 223–246

74. García-López F, García Torres M, Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM (2006) Solving feature subset selection problem by a parallel scatter search. Eur J Oper Res 169:477–489
75. Gehring H, Homberger J (1997) A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Miettinen K, M akel]:a M, Toivanen J (eds) Proceedings of EUROGEN99 – short course on evolutionary algorithms in engineering and computer science, Jyvskyla, pp 57–64
76. Gehring H, Homberger J (2001) A parallel two-phase metaheuristic for routing problems with time windows. Asia-Pac J Oper Res 18(1):35–47
77. Gehring H, Homberger J (2002) Parallelization of a two-phase metaheuristic for routing problems with time windows. J Heuristics 8:251–276
78. Gendreau M, Hertz A, Laporte G (1994) A tabu search heuristic for the vehicle routing problem. Manag Sci 40:1276–1290
79. Gendreau M, Guertin F, Potvin JY, Taillard ÉD (1999) Tabu search for real-time vehicle routing and dispatching. Transp Sci 33(4):381–390
80. Gendreau M, Laporte G, Semet F (2001) A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. Parallel Comput 27(12):1641–1653
81. Glover F (1996) Tabu search and adaptive memory programming – advances, applications and challenges. In: Barr R, Helgason R, Kennington J (eds) Interfaces in computer science and operations research. Kluwer Academic, Norwell, pp 1–75
82. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Norwell,
83. Golden B L, Wasil EA, Kelly JP, Chao IM (1998) Metaheuristics in vehicle routing. In: Crainic T, Laporte G (eds) Fleet management and logistics. Kluwer Academic, Norwell, pp 33–56
84. Greening DR (1989) A taxonomy of parallel simulated annealing techniques. Technical report No. RC 14884, IBM
85. Greening DR (1990) Asynchronous parallel simulated annealing. Lect Complex Syst 3:497–505
86. Greening DR (1990) Parallel simulated annealing techniques. Physica D 42:293–306
87. Groër C, Golden B (2011) A parallel algorithm for the vehicle routing problem. INFORMS J Comput 23(2):315–330
88. Herdy M (1992) Reproductive isolation as strategy parameter in hierarchical organized evolution strategies. In: Männer R, Manderick B (eds) Parallel problem solving from nature, 2. North-Holland, Amsterdam, pp 207–217
89. Hidalgo JI, Prieto M, Lanchares J, Baraglia R, Tirado F, Garnica O (2003) Hybrid parallelization of a compact genetic algorithm. In: Proceedings of the 11th Euromicro conference on parallel, distributed and network-based processing, Genova, pp 449–455
90. Holmqvist K, Migdalas A, Pardalos PM (1997) Parallelized heuristics for combinatorial search. In: Migdalas A, Pardalos P, Storoy S (eds) Parallel computing in optimization. Kluwer Academic, Norwell, pp 269–294
91. Homberger J, Gehring H (1999) Two evolutionary metaheuristics for the vehicle routing problem with time windows. INFOR 37:297–318
92. Janson S, Merkle D, Middendorf M (2005) Parallel ant colony algorithms. In: Alba E (ed) Parallel metaheuristics: a new class of metaheuristics. Wiley, Hoboken, pp 171–201
93. Jin J, Crainic TG, Løkketangen A (2012) A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. Eur J Oper Res 222(3):441–451
94. Jin J, Crainic TG, Løkketangen A (2014) A cooperative parallel metaheuristic for the capacitated vehicle routing problems. Comput Oper Res 44:33–41
95. Laganière R, Mitiche A (1995) Parallel tabu search for robust image filtering. In: Proceedings of IEEE workshop on nonlinear signal and image processing (NSIP'95), Neos Marmaras, vol 2, pp 603–605
96. Lahrichi N, Crainic TG, Gendreau M, Rei W, Crisan CC, Vidal T (2015) An integrative cooperative search framework for multi-decision-attribute combinatorial optimization. Eur J Oper Res 246(2):400–412

97. Laursen PS (1996) Parallel heuristic search – introductions and a new approach. In: Ferreira A, Pardalos P (eds) Solving combinatorial optimization problems in parallel. Lecture notes in computer science, vol 1054. Springer, Berlin, pp 248–274
98. Le Bouthillier A (2007) Recherches coopératives pour la résolution de problèmes d'optimisation combinatoire. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal
99. Le Bouthillier A, Crainic TG (2005) A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Comput Oper Res 32(7):1685–1708
100. Le Bouthillier A, Crainic TG, Kropf P (2005) A guided cooperative search for the vehicle routing problem with time windows. IEEE Intell Syst 20(4):36–42
101. Lee KG, Lee SY (1992) Efficient parallelization of simulated annealing using multiple Markov chains: an application to graph partitioning. In: Mudge TN (ed) Proceedings of the international conference on parallel processing. Algorithms and applications, vol III. CRC Press, Boca Raton, pp 177–180
102. Lee SY, Lee KG (1992) Asynchronous communication of multiple Markov chains in parallel simulated annealing. In: Mudge TN (ed) Proceedings of the international conference on parallel processing. Algorithms and applications, vol III. CRC Press, Boca Raton, pp 169–176
103. Lee KG, Lee SY (1995) Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. In: Brandenburg F (ed) Graph drawing – proceedings GD '95, symposium on graph drawing, Passau. Lecture notes in computer science, vol 1027. Springer, Berlin, pp 396–408
104. Lee SY, Lee KG (1996) Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. IEEE Trans Parallel Distrib Syst 7(10):993–1007
105. Li Y, Pardalos PM, Resende MGC (1994) A greedy randomized adaptive search procedure for quadratic assignment problem. In: DIMACS implementation challenge. DIMACS series on discrete mathematics and theoretical computer science, vol 16. American Mathematical Society, pp 237–261
106. Li F, Golden B L, Wasil EA (2005) A very large-scale vehicle routing: new test problems, algorithms, and results. Comput Oper Res 32(5):1165–1179
107. Lin SC, Punch WF, Goodman ED (1994) Coarse-grain parallel genetic algorithms: categorization and new approach. In: Sixth IEEE symposium on parallel and distributed processing. IEEE Computer Society Press, Los Alamitos, pp 28–37
108. Luque G, Alba E, Dorronsoro B (2005) Parallel genetic algorithms. In: Alba E (ed) Parralel metaheuristics: a new class of algorithms. Wiley, Hoboken, pp 107–125
109. Malek M, Guruswamy M, Pandya M, Owens H (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann Oper Res 21:59–84
110. Martins SL, Ribeiro CC, Souza MC (1998) A parallel GRASP for the Steiner problem in graphs. In: Ferreira A, Rolim J (eds) Proceedings of IRREGULAR'98 – 5th international symposium on solving irregularly structured problems in parallel. Lecture notes in computer science, vol 1457. Springer, Berlin/New York, pp 285–297
111. Martins SL, Resende MGC, Ribeiro CC, Parlados PM (2000) A parallel grasp for the Steiner tree problem in graphs using a hybrid local search strategy. J Glob Optim 17:267–283
112. Michels R, Middendorf M (1999) An ant system for the shortest common supersequence problem. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw-Hill, London, pp 51–61
113. Middendorf M, Reischle F, Schmeck H (2002) Multi colony ant algorithms. J Heuristics 8(3):305–320. doihttp://dx.doi.org/10.1023/A:1015057701750
114. Miki M, Hiroyasu T, Wako J, Yoshida T (2003) Adaptive temperature schedule determined by genetic algorithm for parallel simulated annealing. In: CEC'03 – the 2003 congress on evolutionary computation, Canberra, vol 1, pp 459–466
115. Mingozzi A (2005) The multi-depot periodic vehicle routing problem. In: Abstraction, reformulation and approximation. Lecture notes in computer science. Springer, Berlin/Heidelberg, pp 347–350

116. Moreno-Pérez JA, Hansen P, Mladenović N (2005) Parallel variable neighborhood search. In: Alba E (ed) Parallel metaheuristics: a new class of metaheuristics. Wiley, Hoboken, pp 247–266
117. Mühlenbein H (1989) Parallel genetic algorithms, population genetics and combinatorial optimization. In: Schaffer J (ed) Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 416–421
118. Mühlenbein H (1991) Parallel genetic algorithms, population genetics, and combinatorial optimization. In: Becker JD, Eisele I, Mündemann FW (eds) Parallelism, learning, evolution. Workshop on evolutionary models and strategies – WOPPLOT 89. Springer, Berlin, pp 398–406
119. Mühlenbein H (1992) Parallel genetic algorithms in combinatorial optimization. In: Balci O, Sharda R, Zenios S (eds) Computer science and operations research: new developments in their interface. Pergamon Press, New York, pp 441–456
120. Niar S, Fréville A (1997) A parallel tabu search algorithm for the 0–1 multidimensional knapsack problem. In: 11th international parallel processing symposium (IPPS '97), Geneva. IEEE, pp 512–516
121. Oduntan I, Toulouse M, Baumgartner R, Bowman C, Somorjai R, Crainic TG (2008) A multilevel tabu search algorithm for the feature selection problem in biomedical data sets. Comput Math Appl 55(5):1019–1033
122. Ouyang M, Toulouse M, Thulasiraman K, Glover F, Deogun JS (2000) Multi-level cooperative search: application to the netlist/hypergraph partitioning problem. In: Proceedings of international symposium on physical design. ACM, New York, pp 192–198
123. Ouyang M, Toulouse M, Thulasiraman K, Glover F, Deogun JS (2002) Multilevel cooperative search for the circuit/hypergraph partitioning problem. IEEE Trans Comput-Aided Des 21(6):685–693
124. Pardalos PM, Li Y, KA M (1992) Computational experience with parallel algorithms for solving the quadratic assignment problem. In: Balci O, Sharda R, Zenios S (eds) Computer science and operations research: new developments in their interface. Pergamon Press, New York, pp 267–278
125. Pardalos PM, Pitsoulis L, Mavridou T, Resende MGC (1995) Parallel search for combinatorial optimization: genetic algorithms, simulated annealing, tabu search and GRASP. In: Ferreira A, Rolim J (eds) Proceedings of workshop on parallel algorithms for irregularly structured problems. Lecture notes in computer science, vol 980. Springer, Berlin, pp 317–331
126. Pardalos PM, Pitsoulis L, Resende MGC (1995) A parallel GRASP implementation for the quadratic assignment problem. In: Ferreira A, Rolim J (eds) Solving irregular problems in parallel: state of the art. Kluwer Academic, Norwell, pp 115–130
127. Polacek M, Benkner S, Doerner KF, Hartl RF (2008) A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows. Bus Res 1(2):1–12
128. Porto SCS, Ribeiro CC (1995) A tabu search approach to task scheduling on heteregenous processors under precedence constraints. Int J High-Speed Comput 7:45–71
129. Porto SCS, Ribeiro CC (1996) Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. J Heuristics 1(2):207–223
130. Porto SCS, Kitajima JPFW, Ribeiro CC (2000) Performance evaluation of a parallel tabu search task scheduling algorithm. Parallel Comput 26:73–90
131. Rahimi Vahed A, Crainic TG, Gendreau M, Rei W (2013) A path relinking algorithm for a multi-depot periodic vehicle routing problem. J Heuristics 19(3):497–524
132. Rahoual M, Hadji R, Bachelet V (2002) Parallel ant system for the set covering problem. In: Dorigo M, Di Caro G, Sampels M (eds) Ant algorithms – proceedings of the third international workshop, ANTS 2002. Lecture notes in computer science, vol 2463. Springer, Berlin, pp 262–267
133. Ram DJ, Sreenivas TH, Subramaniam KG (1996) Parallel simulated annealing algorithms. J Parallel Distrib Comput 37:207–212

134. Randall M, Lewis A (2002) A parallel implementation of ant colony optimisation. J Parallel Distrib Comput 62:1421–1432
135. Rego C (2001) Node ejection chains for the vehicle routing problem: sequential and parallel algorithms. Parallel Comput 27:201–222
136. Rego C, Roucairol C (1996) A parallel tabu search algorithm using ejection chains for the VRP. In: Osman I, Kelly J (eds) Meta-heuristics: theory & applications. Kluwer Academic, Norwell, pp 253–295
137. Reimann M, Stummer M, Doerner K (2002) A savings based ants system for the vehicle routing problem. In: Langton C, Cantú-Paz E, Mathias KE, Roy R, Davis L, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke EK, Jonoska N (eds) GECCO 2002: proceedings of the genetic and evolutionary computation conference, New York, 9–13 July 2002. Morgan Kaufmann, San Francisco, pp 1317–1326
138. Reimann M, Doerner K, Hartl R (2004) D-ants: savings based ants divide and conquer the vehicle routing problem. Comput Oper Res 31(4):563–591
139. Ribeiro CC, Rosseti I (2002) A parallel GRASP heuristic for the 2-path network design problem. 4 journée ROADEF, Paris, 20–22 Feb
140. Ribeiro CC, Rosseti I (2002) A parallel GRASP heuristic for the 2-path network design problem. Third meeting of the PAREO Euro working group, Guadeloupe, May
141. Ribeiro CC, Rosseti I (2002) Parallel grasp with path-relinking heuristic for the 2-path network design problem. In: AIRO'2002, L'Aquila, Sept
142. Rochat Y, Taillard ED (1995) Probabilistic diversification and intensification in local search for vehicle routing. J Heuristics 1(1):147–167
143. Sanvicente-Sánchez H, Frausto-Solís J (2002) MPSA: a methodology to parallelize simulated annealing and its application to the traveling salesman problem. In: Coello Coello C, de Albornoz A, Sucar L, Battistutti O (eds) MICAI 2002: advances in artificial intelligence. Lecture notes in computer science, vol 2313. Springer, Heidelberg, pp 89–97
144. Schlierkamp-Voosen D, Mühlenbein H (1994) Strategy adaptation by competing subpopulations. In: Davidor Y, Schwefel H-P, Männer R (eds) Parallel problem solving from nature III. Lecture notes in computer science, vol 866. Springer, Berlin, pp 199–208
145. Schulze J, Fahle T (1999) A parallel algorithm for the vehicle routing problem with time window constraints. Ann Oper Res 86:585–607
146. Sevkli M, Aydin ME (2007) Parallel variable neighbourhood search algorithms for job shop scheduling problems. IMA J Manag Math 18(2):117–133
147. Shonkwiler R (1993) Parallel genetic algorithms. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 199–205
148. Solar M, Parada V, Urrutia R (2002) A parallel genetic algorithm to solve the set-covering problem. Comput Oper Res 29(9):1221–1235
149. Stutzle T (1998) Parallelization strategies for ant colony optimization. In: Eiben AE, Back T, Schoenauer M, Schwefel H-P (eds) Proceedings of parallel problem solving from nature V. Lecture notes in computer science, vol 1498. Springer, Heidelberg, pp 722–731
150. Taillard ÉD (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17:443–455
151. Taillard ÉD (1993) Parallel iterative search methods for vehicle routing problems. Networks 23:661–673
152. Taillard ÉD (1994) Parallel taboo search techniques for the job shop scheduling problem. ORSA J Comput 6(2):108–117
153. Taillard ÉD, Gambardella LM, Gendreau M, Potvin JY (1997) Adaptive memory programming: a unified view of metaheuristics. Eur J Oper Res 135:1–10
154. Taillard ÉD, Gambardella LM, Gendreau M, Potvin JY (1998) Programmation à mémoire adaptative. Calculateurs Parallèles, Réseaux et Systèmes répartis 10:117–140
155. Talbi EG (ed) (2006) Parallel combinatorial optimization. Wiley, Hoboken

156. Talbi EG, Hafidi Z, Geib JM (1998) Parallel adaptive tabu search approach. Parallel Comput 24:2003–2019
157. Talbi EG, Roux O, Fonlupt C, Robillard D (1999) Parallel ant colonies for combinatorial optimization problems. In: Rolim J et al (ed) 11th IPPS/SPDP'99 workshops held in conjunction with the 13th international parallel processing symposium and 10th symposium on parallel and distributed processing, San Juan, 12–16 Apr. Lecture notes in computer science, vol 1586. Springer, Berlin, pp 239–247
158. Talukdar S, Baerentzen L, Gove A, de Souza P (1998) Asynchronous teams: cooperation schemes for autonomous agents. J Heuristics 4:295–321
159. Talukdar S, Murthy S, Akkiraju R (2003) Assynchronous teams. In: Glover F, Kochenberger G (eds) Handbook in metaheuristics. Kluwer Academic, Norwell, pp 537–556
160. ten Eikelder HMM, Aarts BJL, Verhoeven MGA, Aarts EHL (1999) Sequential and parallel local search for job shop scheduling. In: Voß S, Martello S, Roucairol C, Osman IH (eds) Meta-heuristics 98: theory & applications. Kluwer Academic, Norwell, pp 359–371
161. Tongcheng G, Chundi M (2002) Radio network design using coarse-grained parallel genetic algorithms with different neighbor topology. In: Proceedings of the 4th world congress on intelligent control and automation, Shanghai, vol 3, pp 1840–1843
162. Toulouse M, Crainic TG, Gendreau M (1996) Communication issues in designing cooperative multi thread parallel searches. In: Osman IH, Kelly JP (eds) Meta-heuristics: theory & applications. Kluwer Academic, Norwell, pp 501–522
163. Toulouse M, Crainic TG, Sansó B, Thulasiraman K (1998) Self-organization in cooperative search algorithms. In: Proceedings of the 1998 IEEE international conference on systems, man, and cybernetics. Omnipress, Madisson, pp 2379–2385
164. Toulouse M, Crainic TG, Sansó B (1999) An experimental study of systemic behavior of cooperative search algorithms. In: Voß S, Martello S, Roucairol C, Osman IH (eds) Meta-heuristics 98: theory & applications. Kluwer Academic, Norwell, pp 373–392
165. Toulouse M, Thulasiraman K, Glover F (1999) Multi-level cooperative search: a new paradigm for combinatorial optimization and an application to graph partitioning. In: Amestoy P, Berger P, Daydé M, Duff I, Frayssé V, Giraud L, Ruiz D (eds) 5th international Euro-par parallel processing conference. Lecture notes in computer science, vol 1685. Springer, Heidelberg, pp 533–542
166. Toulouse M, Crainic TG, Thulasiraman K (2000) Global optimization properties of parallel cooperative search algorithms: a simulation study. Parallel Comput 26(1):91–112
167. Toulouse M, Crainic TG, Sansó B (2004) Systemic behavior of cooperative search algorithms. Parallel Comput 30(1):57–79
168. Verhoeven MGA, Aarts EHL (1995) Parallel local search. J Heuristics 1(1):43–65
169. Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W (2012) A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. Oper Res 60(3):611–624
170. Voß S (1993) Tabu search: applications and prospects. In: Du DZ, Pardalos P (eds) Network optimization problems. World Scientific, Singapore, pp 333–353
171. Wilkerson R, Nemer-Preece N (1998) Parallel genetic algorithm to solve the satisfiability problem. In: Proceedings of the 1998 ACM symposium on applied computing. ACM, New York, pp 23–28

Theoretical Analysis of Stochastic Search Algorithms

29

Per Kristian Lehre and Pietro S. Oliveto

Contents

Introduction . . . 850
Computational Complexity of Stochastic Search Algorithms . . . 852
Evolutionary Algorithms . . . 853
Test Functions . . . 856
Tail Inequalities . . . 860
Artificial Fitness Levels (AFL) and the Level-Based Method . . . 862
AFL: Upper Bounds . . . 863
AFL: Lower Bounds . . . 864
Level-Based Analysis of Non-elitist Populations . . . 866
Drift Analysis . . . 871
Additive Drift Theorem . . . 872
Multiplicative Drift Theorem . . . 873
Variable Drift Theorem . . . 876
Negative-Drift Theorem . . . 878
Conclusions . . . 880
Final Overview . . . 880
Cross-References . . . 881
References . . . 882

Abstract

Theoretical analyses of stochastic search algorithms, albeit few, have always existed since these algorithms became popular. Starting in the 1990s, a systematic

P. K. Lehre (✉)
School of Computer Science, University of Birmingham, Birmingham, UK
e-mail: p.k.lehre@cs.bham.ac.uk

P. S. Oliveto
Department of Computer Science, University of Sheffield, Sheffield, UK
e-mail: p.oliveto@sheffield.ac.uk

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_35

approach to analyze the performance of stochastic search heuristics has been put in place. This quickly increasing basis of results allows, nowadays, the analysis of sophisticated algorithms such as population-based evolutionary algorithms, ant colony optimization, and artificial immune systems. Results are available concerning problems from various domains including classical combinatorial and continuous optimization, single and multi-objective optimization, and noisy and dynamic optimization. This chapter introduces the mathematical techniques that are most commonly used in the runtime analysis of stochastic search heuristics in finite, discrete spaces. Careful attention is given to the very popular artificial fitness levels and drift analyses techniques for which several variants are presented. To aid the reader's comprehension of the presented mathematical methods, these are illustrated by analysis of simple evolutionary algorithms for artificial example functions. The chapter is concluded by providing references to more complex applications and further extensions of the techniques for the obtainment of advanced results.

Keywords

Stochastic search algorithms · Computational complexity · Runtime analysis · Evolutionary algorithms · (1+1) EA · Drift analysis · Artificial fitness levels · Functions of unitation · Tail inequalities

Introduction

Stochastic search algorithms, also called randomized search heuristics, are general-purpose optimization algorithms that are often used when it is not possible to design a specific algorithm for the problem at hand. Common reasons are the lack of available resources (e.g., enough money and/or time) or because of an insufficient knowledge of the complex optimization problem which has not been studied extensively before. Other times, the only way of acquiring knowledge about the problem is by evaluating the quality of candidate solutions.

Well-known stochastic search algorithms are random local search (▸ Chap. 11, "Theory of Local Search") and simulated annealing. Other more complicated approaches are inspired by processes observed in nature. Popular examples are evolutionary algorithms (EAs) (▸ Chap. 14, "Evolutionary Algorithms") inspired by the concept of natural evolution, ant colony optimization (ACO) (▸ Chap. 13, "Ant Colony Optimization: A Component-Wise Overview") inspired by ant foraging behavior, and artificial immune systems (AIS) inspired by the immune system of vertebrates.

The main advantage of stochastic search heuristics is that, being general-purpose algorithms, they can be applied to a wide range of applications without requiring hardly any knowledge of the problem at hand. Also, the simplicity for which they can be applied implies that practitioners can use them to find high-quality solutions to a wide variety of problems without needing skills and knowledge of algorithm design. Indeed, numerous applications report high performance results

which make them widely used in practice. However, through experimental work and applications, it is difficult to understand the reasons for these successes. In particular, given a stochastic search algorithm, it is unclear on which kind of problems it will achieve good *performance* and on which it will perform poorly. Even more crucial is the lack of understanding of how the parameter settings influence the performance of the algorithms. The goal of a rigorous theoretical foundation of stochastic search algorithms is to answer questions of this nature by *explaining* the success or the failure of these methods in practical applications. The benefits of a theoretical understanding are threefold: (a) guiding the choice of the best algorithm for the problem at hand, (b) determining the optimal parameter settings and (c) aiding the algorithm design, ultimately leading to the achievement of better algorithms.

Theoretical studies of stochastic optimization methods have always existed, albeit few, since these algorithms became popular. In particular, the increasing popularity gained by evolutionary (▸ Chap. 14, "Evolutionary Algorithms") and genetic algorithms (▸ Chap. 15, "Genetic Algorithms") in the 1970s led to various attempts at building a theory for these algorithms. However, such initial studies attempted to provide insights on the *behavior* of evolutionary algorithms rather than estimating their performance. The most popular of these theoretical frameworks was probably the *schema theory* introduced by Holland [16] and made popular by Goldberg [14]. In the early 1990s, a very different approach appeared to the analysis of evolutionary algorithms and consequently randomized search heuristics in general, driven by the insight that these heuristics are indeed randomized algorithms, albeit general-purpose ones, and as such they should be analyzed in a similar spirit to that of classical randomized algorithms [31]. For the last 25 years, this field has kept growing considerably, and nowadays several advanced and powerful tools have been devised that allow the analysis of the performance of involved stochastic search algorithms for problems from various domains. These include problems from classical combinatorial and continuous optimization, dynamic optimization, and noisy optimization. The generality of the developed techniques has allowed their application to the analyses of several families of stochastic search algorithms including evolutionary algorithms, local search, metropolis, simulated annealing, ant colony optimization, artificial immune systems, particle swarm optimization, and estimation of distribution algorithms, among others. For complementary theoretical approaches, see also ▸ Chap. 11, "Theory of Local Search".

The aim of this chapter is to introduce the reader to the most common and powerful tools used in the performance analysis of randomized search heuristics. We will not consider deterministic search heuristics, or search heuristics operating in continuous or infinite search spaces, because these require different methods. Since the main focus is the understanding of the methods, these will be applied to the analysis of very simple evolutionary algorithms for artificial example functions. The hope is that the structure of the functions and the behavior of the algorithms are easy to grasp so the attention of the reader may be mostly focused on the mathematical techniques that will be presented. The techniques are highly general and can

be applied to other randomized search heuristics, such as simulated annealing, randomized local search, ant colony optimization, etc. At the end of the chapter, references to complex applications of the techniques yielding advanced results will be pointed out for further reading.

Computational Complexity of Stochastic Search Algorithms

From the perspective of computer science, stochastic search heuristics are randomized algorithms although more general than problem-specific ones. Hence, it is natural to analyze their performance in the classical way as done in computer science. From this perspective an algorithm should be *correct*, i.e., for every instance of the problem (the input), the algorithm halts with the correct solution (i.e., the correct output), and it should be efficient in terms of its *computational complexity*, i.e., the algorithm uses the computational resources wisely. The resources usually considered are the number of basic computations to find the solution (i.e., time) and the amount of memory required (i.e., space).

Differently from problem-specific algorithms, the goal behind general-purpose algorithms such as stochastic search heuristics is to deliver good performance independently of the problem at hand. In other words, a general-purpose algorithm is "correct" if it *visits the optimal solution* of *any problem* in finite time with probability one regardless of the initialization. If the optimum is never lost afterward, then a stochastic search algorithm is said to *converge* to the optimal solution. In a formal sense, the latter condition for convergence is required because most search heuristics are not capable of recognizing when an optimal solution has been found (i.e., they do not halt). It has been known for almost two decades that a few relatively simple conditions suffice to ensure convergence [40]. Furthermore, these conditions hold for a large class of stochastic search heuristics. Hence, what is more relevant and can make a huge difference on the usefulness of a stochastic search heuristic for a given problem is its *time complexity*. In each iteration the evaluation of the quality of a solution is generally far more expensive than its other algorithmic steps. As a result, it is very common to measure *time* as the number of evaluations of the fitness function (also called objective function) rather than counting the number of basic computations. Since randomized algorithms make random choices during their execution, the runtime of a stochastic search heuristic A to optimize a function f is a random variable $T_{A,f}$. The main measure of interest is:

1. The *expected runtime* $\mathbb{E}\left[T_{A,f}\right]$: the expected number of fitness function evaluations until the optimum of f is found;
   For skewed runtime distributions, the expected runtime may be a deceiving measure of algorithm performance. The following measure therefore provides additional information.
2. The *success probability in t steps* $\Pr\left(T_{A,f} \leq t\right)$: the probability that the optimum is found within t steps.

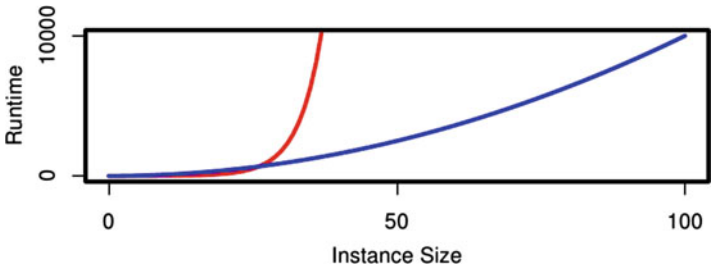


Fig. 1 An efficient search heuristic (blue) versus and inefficient search heuristic (red) for a given instance

Just like in the classical theory of efficient algorithms, the time is analyzed in relation to growing input length and usually described using asymptotic notation [3]. A search heuristic A is said to be *efficient* for a function (class) f if the runtime grows as polynomial function of the instance size. On the other hand, if the runtime grows as an exponential function, then the heuristic is said to be *inefficient*. See Fig. 1 for an illustrative distinction.

Evolutionary Algorithms

A general framework of an evolutionary algorithm is the $(\mu+\lambda)$ EA defined in Algorithm 1. The algorithm evolves a population of μ candidate solutions, generally called the *parent population*. At each generation an *offspring population* of λ individuals is created by selecting individuals from the parent population uniformly at random and by applying a mutation operator to them. The generation is concluded by selecting the μ fittest individuals out of the $\mu + \lambda$ parents and offspring. Algorithm 1 presents a formal definition.

Algorithm 1: $(\mu+\lambda)$ EA

1: **Initialisation**:
Initialise $P_0 = \{x^{(1)}, \ldots, x^{(\mu)}\}$ with μ individuals chosen uniformly a random from $\{0,1\}^n$;
$t \leftarrow 0$;
2: **for** $i = 1, \ldots, \lambda$ **do**
3: **Selection for Reproduction**: Choose $x \in P_t$ uniformly at random;
4: **Variation**: Create $y^{(i)}$ by flipping each bit in x with probability p_m;
5: **end for**
6: **Selection for Replacement**
Create the new population P_{t+1} by choosing the best μ individuals out of $\{x^{(1)}, \ldots, x^{(\mu)}, y^{(1)}, \ldots, y^{(\lambda)}\}$;
7: $t \leftarrow t + 1$; Continue at 2;

In order to apply the algorithm for the optimization of a fitness function $f : \{0,1\}^n \to \mathbb{R}$, some parameters need to be set: the population size μ, the offspring population size λ, and the mutation rate p_m. Generally $p_m = 1/n$ is considered a good setting for the mutation rate. Also, in practical applications a stopping criterion has to be defined since the algorithm does not halt. A fixed number of generations or a fixed number of fitness function evaluations are usually decided in advance. Since the objective of the analysis is to calculate the time required to reach the optimal (approximate) solution for the first time, no stopping condition is required, and one can assume that the algorithms are allowed to run forever. The + symbol in the algorithm's name indicates that *elitist truncation selection* is applied. This means that the whole population consisting of both parents and offspring are sorted according to fitness, and the best μ are retained for the next generation. Some criterion needs to be decided in case the best μ individuals are not uniquely defined. Ties between solutions of equal fitness may be broken uniformly at random. Often offspring are preferred over parents of equal fitness. In the latter case, if $\mu = \lambda = 1$ are set, then the standard (1+1) EA is obtained, a very simple and well-studied evolutionary algorithm. On the other hand, if some stochastic selection mechanism was used instead of the elitist mechanism and a crossover operator was added as variation mechanism, then Algorithm 1 would become a genetic algorithm (GA) [14]. Given the importance of the (1+1) EA in this chapter, a formal definition is given in Algorithm 2.

Algorithm 2: (1+1) EA

1: **Initialisation**:
   Initialise $x^{(0)}$ uniformly a random from $\{0,1\}^n$;
   $t \leftarrow 0$;
2: **Variation**:
   Create y by flipping each bit in $x^{(t)}$ with probability $p_m = 1/n$;
3: **Selection for Replacement**
4: **if** $f(y) \geq f(x^{(t)})$ **then**
5: $\quad x^{(t+1)} \leftarrow y$
6: **end if**
7: $t \leftarrow t + 1$; Continue at 2;

The algorithm is initialized with a random bitstring. At each generation a new candidate solution is obtained by flipping each bit with probability $p_m = 1/n$. The number of bits that flip can be represented by a binomial random variable $X \sim \text{Bin}(n, p)$ where n is the number of bits (i.e., the number of trials) and $p = 1/n$ is the probability of a success (i.e., a bit actually flips), while $1 - p = 1 - 1/n$ is the probability of a failure (i.e., the bit does not flip). Then, the expected number of bits that flip in one generation is given by the expectation of the binomial random variable, $\mathbb{E}[X] = n \cdot p = n \cdot 1/n = 1$.

The algorithm behaves in a very different way compared to the random local search (RLS) algorithm that flips *exactly* one bit per iteration. Although the (1+1) EA flips exactly one bit in expectation per iteration, many more bits may flip or even none at all. In particular, the (1+1) EA is a *global optimizer* because there is a positive probability that any point in the search space is reached in each generation. As a consequence, the algorithm will find the global optimum in finite time. On the other hand, RLS is a local optimizer since it gets stuck once it reaches a local optimum because it only flips one bit per iteration.

The probability that a binomial random variable $X \sim \text{Bin}(n, p)$ takes value j (i.e., j bits flip) is

$$\Pr(X = j) = \binom{n}{j} p^j (1-p)^{n-j}.$$

Hence, the probability that the (1+1) EA flips exactly one bit is

$$\Pr(X = 1) = \binom{n}{1} \cdot \left(\frac{1}{n}\right) \cdot \left(1 - \frac{1}{n}\right)^{n-1} = \left(1 - \frac{1}{n}\right)^{n-1} \geq 1/e \approx 0.37$$

So the outcome of one generation of the (1+1) EA is similar to that of RLS only approximately 1/3 of the generations. The probability that two bits flip is exactly half the probability that one flips:

$$\begin{aligned}
\Pr(X = 2) &= \binom{n}{2} \left(\frac{1}{n}\right)^2 \left(1 - \frac{1}{n}\right)^{n-2} \\
&= \frac{n(n-1)}{2} \left(\frac{1}{n}\right)^2 \left(1 - \frac{1}{n}\right)^{n-2} \\
&= \frac{1}{2} \left(1 - \frac{1}{n}\right)^{n-1} \approx 1/(2e)
\end{aligned}$$

On the other hand, the probability no bits flip at all is

$$\Pr(X = 0) = \binom{n}{0} (1/n)^0 \cdot (1 - 1/n)^n \approx 1/e$$

The latter result implies that in more than 1/3 of the iterations, no bits flip. This should be taken into account when evaluating the fitness of the offspring, especially for expensive fitness functions.

In general, the probability that i bits flip decreases exponentially with i:

$$\Pr(X = i) = \binom{n}{i} \cdot \frac{1}{n^i} \cdot \left(1 - \frac{1}{n}\right)^{n-i} = \frac{1}{i!} \cdot \left(1 - \frac{1}{n}\right)^{n-i} \approx \frac{1}{i! \cdot e}$$

In the worst case, all the bits may need to flip to reach the optimum in one step. This event has probability $1/n^n$. Since this is always a lower bound on the probability of reaching the optimum in each generation, by a simple waiting time argument, an upper bound of $O(n^n)$ may be derived for the expected runtime of the (1+1) EA on any pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$. It is simple to design an example trap function for which the algorithm actually requires $\Theta(n^n)$ expected steps to reach the optimum [13]. This simple result further motivates why it is fundamental to gain a foundational understanding of how the runtime of stochastic search heuristics depends on the parameters of the problem and on the parameters of the algorithms.

Test Functions

Test functions are artificially designed to analyze the performance of stochastic search algorithms when they face optimization problems with particular characteristics. These functions are used to highlight characteristics of function classes which may make the optimization process easy or hard for a given algorithm. For this reason they are often referred to as *toy problems*. The analysis on test functions of simple and well-understood structure has allowed the development of several general techniques for the analysis. Afterward these techniques have allowed to analyze the same algorithms for more complicated problems with practical applications such as classical combinatorial optimization problems. Furthermore, in recent years several standard techniques originally developed for simple algorithms have been extended to allow the analyses of more realistic algorithms. In this section the test functions that will be used as example functions throughout the chapter are introduced.

The most popular test function is ONEMAX (x) := $\sum_{i=1}^{n} x_i$ which simply counts the number of one-bits in the bitstring. The global optimum is a bitstring of only one-bits. ONEMAX is the easiest function with unique global optimum for the (1+1) EA [7].

A particularly difficult test function for stochastic search algorithms is the *needle-in-a-haystack* function. NEEDLE(x) := $\prod_{i=1}^{n} x_i$ consists of a huge plateau of fitness value zero apart from only one optimal point of fitness value one represented by the bitstring of only one-bits. This function is hard for search heuristics because all the search points apart from the optimum have the same fitness. As a consequence, the algorithms cannot gather any information about where the needle is by sampling search points.

Both ONEMAX and NEEDLE (as defined above) have the property that the function values only depend on the number of ones in the bitstring. The class of functions with this property is called *functions of unitation*

$$\text{UNITATION}(x) := f\left(\sum_{i=1}^{n} x_i\right)$$

Throughout this chapter, functions of unitation will be used as a general example class to demonstrate the use of the techniques that will be introduced. For simplicity of the analysis, the optimum is assumed to be the bitstring of only one-bits.

For the analysis the function of unitation will be divided into three different kinds of subblocks: linear blocks, gap blocks, and plateau blocks. Each block will be defined by its length parameter m (i.e., the number of bits in the block) and by its position parameter k (i.e., each block starts at bitstrings with $m + k$ zeroes and ends at bitstrings with k zeroes). Given a unitation function, it is divided into subblocks proceeding from left to right from the all-zeroes bitstring toward the all-ones bitstring. If the fitness increases with the number of ones, then a *linear block* is created. The linear block ends when the function value stops increasing with the number of ones.

$$\text{LINEAR}(|x|) = \begin{cases} a|x| + b & \text{if } k < n - |x| \leq k + m \\ 0 & \text{otherwise.} \end{cases}$$

See Fig. 2 for an illustration.

If the fitness function decreases with the number of ones, then a *gap block* is created. The gap block ends when the fitness value reaches for the first time a higher value than the value at the beginning of the block.

$$\text{GAP}(|x|) = \begin{cases} a & \text{if } n - |x| = k + m \\ 0 & \text{otherwise.} \end{cases}$$

See Fig. 3 for an illustration.

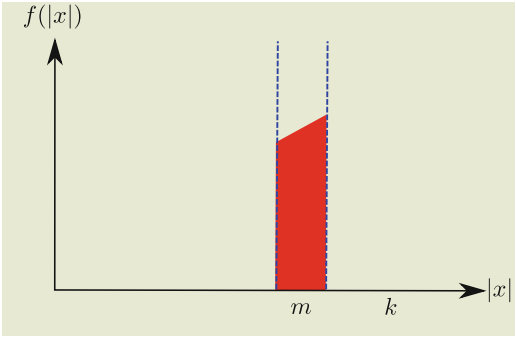


Fig. 2 A linear unitation block of length m starting at position $m + k$ and ending at position k. A linear unitation block of length n is the ONEMAX function

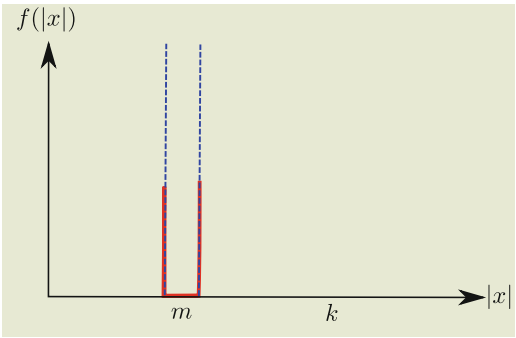


Fig. 3 A gap unitation block of length m starting at position $m + k$ and ending at position k. A gap unitation block of length $n - 1$ is the NEEDLE function

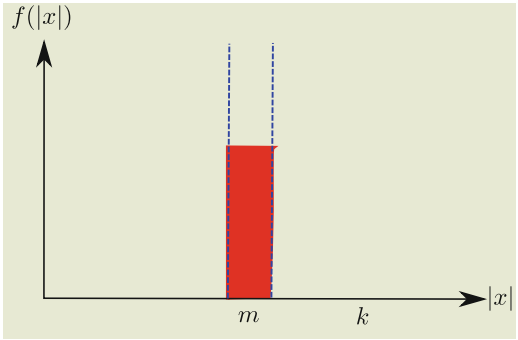


Fig. 4 A plateau unitation block of length m starting at position $m + k$ and ending at position k

If the fitness remains the same as the number of ones in the bitstrings increases, then a *plateau block* is created. The block ends at the first point where the fitness value changes.

$$\textsc{Plateau}(|x|) = \begin{cases} a & \text{if } k < n - |x| \leq k + m \\ 0 & \text{otherwise.} \end{cases}$$

See Fig. 4 for an illustration.

By proceeding from left to right, the whole search space is subdivided into blocks. See Fig. 5 for an illustration. Let the unitation function be subdivided into r sub-functions $f_1, f_2, \ldots f_r$, and let T_i be the runtime for an elitist search heuristic to optimize each sub-function f_i. Then by linearity of expectation, an upper bound on the expected runtime of an elitist stochastic search heuristic for the unitation function is

$$\mathbb{E}[T] \leq \mathbb{E}\left[\sum_{i=1}^{r} T_i\right] = \sum_{i=1}^{r} \mathbb{E}[T_i].$$

Hence, an upper bound on the total runtime for the unitation function may be achieved by calculating upper bounds on the runtime for each block separately. Once these are obtained, summing all the bounds yields an upper bound on the total runtime. Attention needs to be put when calculating upper bounds on the runtime

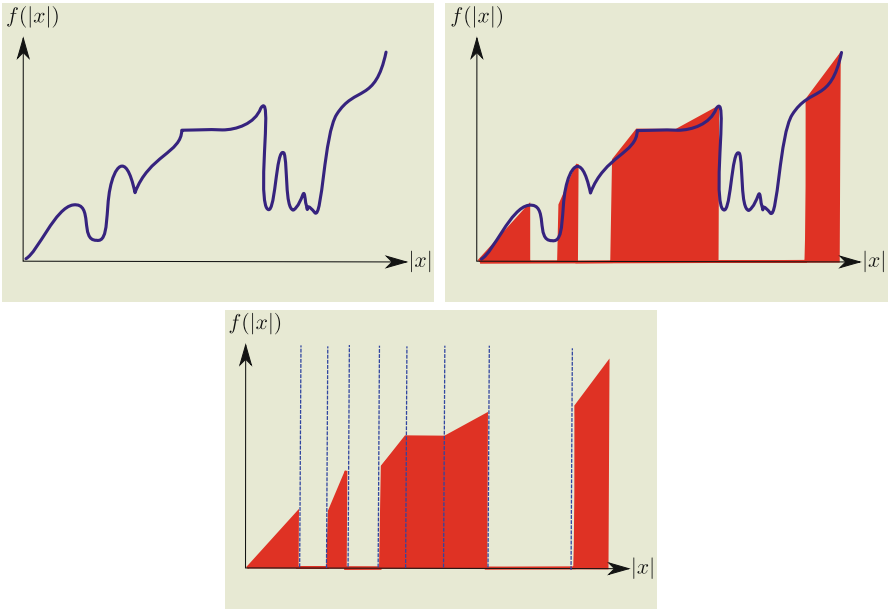


Fig. 5 An illustration of how the search space of a unitation function may be subdivided into blocks of the three kinds

to overcome a plateau block when this is followed by a gap block because points straight after the end of the plateau will have lower fitness values and hence will not be accepted. In these special cases, the upper bound for the PLATEAU block needs to be multiplied by the upper bound for the GAP block to achieve a correct upper bound on the runtime to overcome both blocks. In the remainder of the chapter, upper and lower bounds for each type of block will be derived as example applications of the presented runtime analysis techniques. The reader will then be able to calculate the runtime of the (1+1) EA and other evolutionary algorithms for any such unitation function.

By simply using waiting time arguments, it is possible to derive upper and lower bounds on the runtime of the (1+1) EA for the GAP block. Assuming that the algorithm is at the beginning of the gap block, then to reach the end, it is *sufficient* to flip m zero-bits into one-bits and leave the other bits unchanged. On the other hand, it is a *necessary* condition to flip at least m zero-bits because all search points achieved by flipping less than m zero-bits have a fitness value of zero and would not be accepted by selection. Given that there are $m + k$ zero-bits available at the beginning of the block, the following upper and lower bounds on the probability of reaching the end of the block follow

$$\left(\frac{m+k}{nm}\right)^m \frac{1}{e} \leq \binom{m+k}{m}\left(\frac{1}{n}\right)^m \frac{1}{e} \leq p \leq \binom{m+k}{m}\left(\frac{1}{n}\right)^m \leq \left(\frac{(m+k)e}{nm}\right)^m.$$

Here the outer inequalities are achieved by using $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ for $k \geq 1$. Then by simple waiting-time arguments, the expected time for the (1+1) EA to optimize a GAP block of length m and position k is upper and lower bounded by

$$\left(\frac{nm}{(m+k)e}\right)^m \leq \binom{m+k}{m}^{-1} n^m \leq \mathbb{E}[T] \leq en^m \binom{m+k}{m}^{-1} \leq e\left(\frac{nm}{m+k}\right)^m.$$

Tail Inequalities

The runtime of a stochastic search algorithm A for a function (class) f is a random variable $T_{A,f}$, and the main goal of a runtime analysis is to calculate its expectation $\mathbb{E}[T_{A,f}]$. Sometimes the expected runtime may be particularly large, but there may also be a high probability that the actual optimization time is significantly lower. In these cases a result about the *success probability* within t steps helps considerably in understanding the algorithm's performance. In other occasions it may be interesting to simply gain knowledge about the probability that the actual optimization time deviates from the expected runtime. In such circumstances *tail inequalities* turn out to be very useful tools by allowing to obtain bounds on the runtime that hold with high probability. An example of the expectation of a random variable and its probability distribution are given in Fig. 6.

Given the expectation of a random variable, which often may be estimated easily, tail inequalities give bounds on the probability that the actual random variable deviates from its expectation [30, 31]. The most simple tail inequality is Markov's inequality. Many strong tail inequalities are derived from Markov's inequality.

Theorem 1 (Markov's Inequality). *Let X be a random variable assuming only nonnegative values. Then for all $t \in \mathbb{R}^+$,*

$$\Pr(X \geq t) \leq \frac{\mathbb{E}[X]}{t}.$$

The power of the inequality is that no knowledge about the random variable is required apart from it being nonnegative.

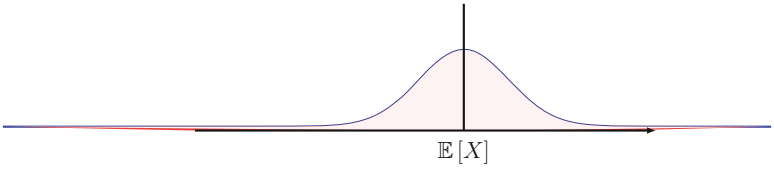


Fig. 6 The expectation of a random variable and its probability distribution. The tails are highlighted in red

Let X be a random variable indicating the number of bits flipped in one iteration of the (1+1) EA. As seen in the previous section, one bit is flipped per iteration in expectation, i.e., $\mathbb{E}[X] = 1$. One may wonder what is the probability that more than one bit is flipped in one time step. A straightforward application of Markov's inequality reveals that in at least half of the iterations, either one bit is flipped or none:

$$\Pr(X \geq 2) \leq \frac{\mathbb{E}[X]}{2} = \frac{1}{2}$$

Similarly, one may want to gain some information on how many *ones* are contained in the bitstring at initialization, given that in expectation there are $\mathbb{E}[X] = n/2$ (here X is a binomial random variable with parameters n and $p = 1/2$). An application of Markov's inequality yields that the probability of having more than $(2/3)n$ ones at initialization is bounded by

$$\Pr(X \geq (2/3)n) \leq \frac{\mathbb{E}[X]}{(2/3)n} = \frac{n/2}{(2/3)n} = 3/4 \tag{1}$$

Since X is binomially distributed, it is reasonable to expect that, for large enough n, the actual number of obtained ones at initialization would be more concentrated around the expected value. In particular, while the bound is obviously correct, the probability that the initial bitstring has more than $(2/3)n$ ones is much smaller than $3/4$. However, to achieve such a result, more information about the random variable should be required by the tail inequality (i.e., that it is binomially distributed). An important class of tail inequalities used in the analysis of stochastic search heuristics are Chernoff bounds.

Theorem 2 (Chernoff Bounds). *Let $X_1, X_2, \ldots X_n$ be independent random variables taking values in $\{0, 1\}$. Define $X = \sum_{i=1}^{n} X_i$, which has expectation $E(X) = \sum_{i=1}^{n} \Pr(X_i = 1)$.*

(a) $\Pr(X \leq (1-\delta)\mathbb{E}[X]) \leq e^{\frac{-\mathbb{E}[X]\delta^2}{2}}$ *for* $0 \leq \delta \leq 1$.
(b) $\Pr(X > (1+\delta)\mathbb{E}[X]) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathbb{E}[X]}$ *for* $\delta > 0$.

An application of Chernoff bounds reveals that the probability that the initial bitstring has more than $(2/3)n$ one-bits is exponentially small in the length of the bitstring. Let $X = \sum_{i=1}^{n} X_i$ be the random variable summing up the random values X_i of each of the n bits. Since each bit is initialized with probability $1/2$, it holds that $\Pr(X_i = 1) = 1/2$ and $\mathbb{E}[X] = n/2$. By fixing $\delta = 1/3$, it follows that $(1+\delta)\mathbb{E}[X] = (2/3)n$, and finally by applying inequality (b),

$$\Pr(X > (2/3)n) \leq \left(\frac{e^{1/3}}{(4/3)^{4/3}}\right)^{n/2} < \left(\frac{29}{30}\right)^{n/2}$$

In fact an exponentially small probability of deviating from $n/2$ by a constant factor of the search space c/n for any constant $c > 0$ may easily be obtained by Chernoff bounds.

Artificial Fitness Levels (AFL) and the Level-Based Method

The artificial fitness-level technique is a very simple method to achieve upper bounds on the runtime of elitist stochastic optimization algorithms. Albeit its simplicity, it often achieves very good bounds on the runtime.

The idea behind the method is to divide the search space of size 2^n into m disjoint fitness-based partitions $A_1, \dots A_m$ of increasing fitness such that $f(A_i) < f(A_j)$ $\forall i < j$. The union of these partitions should cover the whole search space, and the level of highest fitness A_m should contain the global optimum (or all global optima if there is more than one) (Fig. 7).

Definition 1. A tuple $(A_1, A_2, \dots, A_m)$ is an f**-based partition** of $f : \mathcal{X} \to \mathbb{R}$ if

1. $A_1 \cup A_2 \cup \cdots \cup A_m = \mathcal{X}$
2. $A_i \cap A_j = \emptyset$ for $i \neq j$
3. $f(A_1) < f(A_2) < \cdots < f(A_m)$
4. $f(A_m) = \max_x f(x)$

For functions of unitation, a natural way of defining a fitness-based partition is to divide the search space into $n + 1$ levels, each defined by the number of ones in the bitstring. For the ONEMAX function, where fitness increases with the number of ones in the bitstring, the fitness levels would be naturally defined as $A_i := \{x \in \{0,1\}^n \mid \text{ONEMAX}(x) = i\}$.

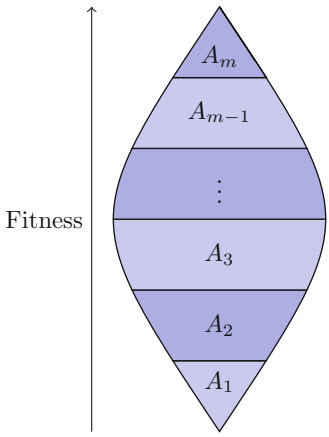


Fig. 7 A partition of the search space satisfying the conditions of an f-based partition

AFL: Upper Bounds

Given a fitness-based partition of the search space, it is obvious that an elitist algorithm using only one individual will only accept points of the search space that belong to levels of higher or equal fitness to the current level. Once a new fitness level has been reached, the algorithm will never return to previous levels. This implies that each fitness level has to be left at most once by the algorithm. Since in the worst case all fitness levels are visited, the sum of the expected times to leave all levels is an upper bound on the expected time to reach the global optimum. The artificial fitness-level method simplifies this idea by only requiring a lower-bound s_i on the probability of leaving each level A_i rather than asking for the exact probabilities to leave each level.

Theorem 3 (Artificial Fitness Levels). *Let $f : \mathcal{X} \to \mathbb{R}$ be a fitness function, $A_1 \dots A_m$ be a fitness-based partition of f, and $s_1 \dots s_{m-1}$ be lower bounds on the corresponding probabilities of leaving the respective fitness levels for a level of better fitness. Then the expected runtime of an elitist algorithm using a single individual is $\mathbb{E}\left[T_{A,f}\right] \leq \sum_{i=1}^{m-1} 1/s_i$.*

The artificial fitness-level method will now be applied to derive an upper bound on the expected runtime of (1+1) EA for the ONEMAX function. Afterward, the bound will be generalized to general linear blocks of unitation.

Theorem 4. *The expected runtime of the (1+1) EA on* ONEMAX *is $O(n \ln n)$.*

Proof. The artificial fitness-level method will be applied to the $n + 1$ partitions defined by the number of ones in the bitstring, i.e., $A_i := \{x \in \{0,1\}^n \mid \text{ONEMAX}(x) = i\}$. This means that all bitstrings with i ones and $n - i$ zeroes belong to fitness-level A_i. For each level A_i, the method requires a lower bound on the probability of reaching any level A_j where $j > i$. To reach a level of higher fitness, it is necessary to increase the number of ones in the bitstring. However, it is sufficient to flip a *zero* into a *one* and leave the remaining bits unchanged. Since the probability of flipping a bit is $1/n$ and there are $n - i$ zeroes that may be flipped, a lower bound on the probability to reach a level of higher fitness from level A_i is

$$s_i \geq (n-i) \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n-i}{en}$$

where $(1 - 1/n)^{n-1}$ is the probability of leaving $n - 1$ bits unchanged and the inequality follows because $(1 - 1/n)^{n-1} \geq 1/e$ for all $n \in \mathbb{N}$.

Then by the artificial fitness-level method (Theorem 3),

$$\mathbb{E}\left[T_{(1+1)\ EA,\text{ONEMAX}}\right] \leq \sum_{i=0}^{m-1} 1/s_i \leq \sum_{i=0}^{n-1} \frac{en}{n-i} = en \sum_{i=1}^{n} \frac{1}{i} = O(n \ln n).$$

□

Theorem 5. *The expected runtime of the (1+1)-EA for a* linear block *of length m ending at position k is $O(n \ln((m+k)/k))$.*

Proof. Apply the artificial fitness-level method where each partition A_i consists of the bitstrings in the block with i zeroes. Then the probability of leaving a fitness level is bounded by $s_i \geq i/n \cdot (1-1/n)^{n-1} \geq i/en$. Given that at most m fitness levels need to be left and that the block starts at position $m+k$ and ends at position k, by Theorem 3 the expected runtime is

$$\mathbb{E}[T] \leq \sum_{i=k+1}^{k+m} \frac{en}{i} \leq en \sum_{i=k+1}^{k+m} \frac{1}{i} \leq en \left(\sum_{i=1}^{k+m} \frac{1}{i} - \sum_{i=1}^{k} \frac{1}{i} \right) \leq en \ln \left(\frac{m+k}{k} \right)$$

□

AFL: Lower Bounds

Recently Sudholt introduced an artificial fitness-level method to obtain lower bounds on the runtime of stochastic search algorithms [42]. Since lower bounds are aimed for, apart from the probabilities of leaving each fitness level, the method needs to also take into account the probability that some levels may be skipped by the algorithm.

Theorem 6. *Consider a fitness function $f : \mathcal{X} \to \mathbb{R}$ and $A_1 \ldots A_m$ a fitness-based partition of f. Let u_i be the probability of starting in level A_i, s_i be an upper bound on the probability of leaving A_i, and $p_{i,j}$ be an upper bound on the probability of jumping from level A_i to level A_j. If there exists some $0 < \chi \leq 1$ such that for all $j > i$*

$$p_{i,j} \geq \chi \cdot \sum_{k=j}^{m-1} p_{i,k},$$

then the expected runtime of an elitist algorithm using a single individual is

$$\mathbb{E}\left[T_{A,f}\right] \geq \chi \cdot \sum_{i=1}^{m-1} u_i \sum_{j=i}^{m-1} \frac{1}{s_j}$$

The method will first be illustrated for the (1+1) EA on the ONEMAX function. Afterward, the result will be generalized to general linear blocks of unitation.

Theorem 7. *The expected runtime of the (1+1) EA on* ONEMAX *is* $\Omega(n \ln n)$.

Proof. Apply the artificial fitness-level method on the $n+1$ partitions defined by the number of ones in the bitstring, i.e., $A_i := \{x \in \{0,1\}^n \mid \text{ONEMAX}(x) = i\}$. This means that all bitstrings with i ones and $n-i$ zeroes belong to fitness level A_i. To apply the artificial fitness-level method, bounds on s_i and χ need to be derived. An upper bound on the probability of leaving fitness-level A_i is simply $s_i \leq (n-i)/n$ because it is a necessary condition that at least one zero flips to reach a better fitness level. The bound follows because each bit flips with probability $1/n$ and there are $n-i$ zeroes available to be flipped. In order to obtain an upper bound on χ, the method requires a lower bound on $p_{i,j}$ and an upper bound on $\sum_{k=j}^{m-1} p_{i,k}$. For the lower bound on $p_{i,j}$, notice that in order to reach level A_j, it is sufficient to flip $j-i$ zeroes out of the $n-i$ zeroes available and leave all the other bits unchanged. Hence the following bound is obtained:

$$p_{ij} \geq \binom{n-i}{j-i} \left(\frac{1}{n}\right)^{j-i} \left(1-\frac{1}{n}\right)^{n-(j-i)}$$

For an upper bound on the sum, notice that to reach any level A_k $k \geq j$ from level A_i, it is necessary to flip at least $j-i$ zeroes out of the $n-i$ available zeroes. So,

$$\sum_{k=j}^{n-1} p_{i,k} \leq \binom{n-i}{j-i} \left(\frac{1}{n}\right)^{j-i}$$

and for $\chi := 1/e$ the condition of Theorem 6 is satisfied as follows:

$$p_{i,j} \geq \left(1-\frac{1}{n}\right)^{n-(j-i)} \cdot \sum_{k=j}^{n-1} p_{i,k} \geq \chi \cdot \sum_{k=j}^{n-1} p_{i,k}$$

By Eq. (1), the probability that the initial search point has less than $(2/3)n$ one-bits is at least

$$\sum_{i=1}^{(2/3)n} u_i \geq 1 - \frac{3}{4}$$

The statement of Theorem 6 now yields

$$\begin{aligned}
\mathbb{E}\left[T_{A,f}\right] &\geq \left(\frac{1}{e}\right) \cdot \sum_{i=1}^{n-1} u_i \sum_{j=i}^{n-1} \frac{1}{s_j} \\
&> \left(\frac{1}{e}\right) \cdot \left(\sum_{i=1}^{(2/3)n} u_i\right) \sum_{j=(2/3)n}^{n-1} \frac{1}{s_j} \\
&\geq \left(\frac{1}{e}\right) \cdot \left(1 - \frac{3}{4}\right) \sum_{j=(2/3)n}^{n-1} \frac{n}{n-j} \\
&\geq \left(\frac{n}{4e}\right) \cdot \sum_{j=1}^{n/3} \frac{1}{j}.
\end{aligned}$$

It now follows that $\mathbb{E}\left[T_{A,f}\right] = \Omega(n \log n)$. □

Similarly the following result may also be proved for linear blocks of unitation functions by defining the fitness partitions as $A_i := \{x : n - |x| = k + m - i\}$ for $0 \leq i \leq m$.

Theorem 8. *The expected runtime of the (1+1)-EA for a* linear block *of length m ending at position k is* $\Omega(n \ln((m+k)/k))$.

Level-Based Analysis of Non-elitist Populations

A weakness with the classical artificial fitness-level technique is that it is limited to search heuristics that only keep one solution, such as the (1+1) EA, and it heavily relies on the selection mechanism to use elitism. Corus et al. [6] recently introduced the so-called level-based analysis, a generalization of fitness-level theorems for non-elitist evolutionary algorithms which is also applicable to search heuristics with populations, using higher arity operators such as crossover.

Their theorem applies to any algorithm that can be expressed in the form of Algorithm 3, such as genetic algorithms [6] and estimation of distribution algorithms UMDA [9]. The main component of the algorithm is a random operator D which given the current population $P_t \in \mathcal{X}^\lambda$ returns a probability distribution $D(P_t)$ over the search space $\mathcal{X}$. The next population P_{t+1} is obtained by sampling individuals independently from this distribution.

In contrast to classical fitness-level theorems, the level-based theorem (Theorem 9) only assumes a partition $(A_1, \ldots, A_{m+1})$ of the search space $\mathcal{X}$ and not an f-based partition (see Definition 1). Hence, the level-based method is not an artificial fitness-level method. Each of the sets $A_j, j \in [m+1]$ is called a *level*, and the symbol $A_j^+ := \bigcup_{i=j+1}^{m+1} A_i$ denotes the set of search points above level A_j. Given a constant $\gamma_0 \in (0,1)$, a population $P \in \mathcal{X}^\lambda$ is considered to be at level A_j

Algorithm 3: Population-based algorithm with independent sampling

1: **Initialisation**:
$t \leftarrow 0$; Initialise P_t uniformly at random from $\mathcal{X}^\lambda$.
2: **Variation and Selection**:
3: **for** $i = 1 \dots \lambda$ **do**
Sample $P_{t+1}(i) \sim D(P_t)$
4: **end for**
5: $t \leftarrow t + 1$; Continue at 2

with respect to γ_0 if $|P \cap A_{j-1}^+| \geq \gamma_0 \lambda$ and $|P \cap A_j^+| < \gamma_0 \lambda$ meaning that at least a γ_0 fraction of the population is in level A_j or higher.

Theorem 9 ([6]). *Given any partition of a finite set $\mathcal{X}$ into m nonoverlapping subsets $(A_1, \dots, A_{m+1})$, define $T := \min\{t\lambda \mid |P_t \cap A_{m+1}| > 0\}$ to be the first point in time that elements of A_{m+1} appear in P_t of Algorithm 3. If there exist parameters $z_1, \dots, z_m, z_* \in (0, 1]$, $\delta > 0$, and a constant $\gamma_0 \in (0, 1)$ such that for all $j \in [m]$, $P \in \mathcal{X}^\lambda$, $y \sim D(P)$, and $\gamma \in (0, \gamma_0]$, it holds*

(C1) $\Pr\left(y \in A_j^+ \mid |P \cap A_{j-1}^+| \geq \gamma_0 \lambda\right) \geq z_j \geq z_*$

(C2) $\Pr\left(y \in A_j^+ \mid |P \cap A_{j-1}^+| \geq \gamma_0 \lambda \text{ and } |P \cap A_j^+| \geq \gamma\lambda\right) \geq (1 + \delta)\gamma$, *and*

(C3) $\lambda \geq \frac{2}{a} \ln\left(\frac{16m}{ac\varepsilon z_*}\right)$ *with* $a = \frac{\delta^2 \gamma_0}{2(1+\delta)}$, $\varepsilon = \min\{\delta/2, 1/2\}$ *and* $c = \varepsilon^4/24$

then

$$\mathbb{E}[T] \leq \frac{2}{c\varepsilon}\left(m\lambda(1 + \ln(1 + c\lambda)) + \sum_{j=1}^{m} \frac{1}{z_j}\right).$$

The theorem provides an upper bound on the expected optimization time of Algorithm 3 if it is possible to find a partition $(A_1, \dots, A_{m+1})$ of the search space $\mathcal{X}$ and accompanying parameters $\gamma_0, \delta, z_1, \dots, z_m, z_*$ such that conditions (C1), (C2), and (C3) are satisfied. Condition (C1) requires a nonzero probability z_j of creating an individual in level A_{j+1} or higher if there are already at least $\gamma_0\lambda$ individuals in level A_j or higher. In typical applications, this imposes some conditions on the variation operator. The condition is analogous to the probability s_j in the artificial fitness-level technique. Condition (C2) requires that if in addition there are $\gamma\lambda$ individuals at level A_{j+1} or better, then the probability of producing an individual in level A_{j+1} or better should be larger than γ by a multiplicative factor $1 + \delta$. In typical applications, this imposes some conditions on the strength of the selective pressure in the algorithm. Finally, condition (C3) imposes minimal requirements on the population size in terms of the parameters above.

As an example application of the level-based theorem, the (μ, λ) EA is analyzed, which is the non-elitist variant of the $(\mu + \lambda)$ EA shown in Algorithm 1. The two algorithms differ in the selection step (line 6), where the new population P_{t+1} in (μ, λ) EA is chosen as the best μ individuals out of $\{y_1, \ldots, y_\lambda\}$ and breaking ties uniformly at random. While the $(\mu + \lambda)$ EA always retains the best μ individuals in the population (hence the name elitist), the (μ, λ) EA always discards the old individuals $x^{(1)}, \ldots, x^{(\mu)}$.

At first sight, it may appear as if the (μ, λ) EA cannot be expressed in the form of Algorithm 3. The μ individuals $x^{(1)}, \ldots, x^{(\mu)}$ that are kept in each generation are not independent due to the inherent sorting of the offspring. However, taking a different perspective, the population of the algorithm at time t could also be interpreted as the λ offspring $y^{(1)}, \ldots, y^{(\lambda)}$. In this alternative interpretation, the new population is now created by sampling uniformly at random among the μ best individuals in the population and applying the mutation operator. The operator D in Algorithm 3 can now be defined as in Algorithm 4.

Algorithm 4: Operator D corresponding to (μ, λ) EA

1: **Selection**:
Sort the population $P_t = (y^{(1)}, \ldots, y^{(\lambda)})$ such that $f(y^{(1)}) \geq f(y^{(2)}) \geq \ldots \geq f(y^{(\lambda)})$.
Select x uniformly at random among $\{y^{(1)}, \ldots, y^{(\mu)}\}$.
2: **Variation (mutation)**:
Create x' by flipping each bit in x with probability χ/n.
3: **return** x'.

The following lemma will be useful when estimating the probability that the mutation operator does not flip any bit positions.

Lemma 1. *For any $\delta \in (0, 1)$ and $\chi > 0$, if $n \geq (\chi + \delta)(\chi/\delta)$, then*

$$\left(1 - \frac{\chi}{n}\right)^n \geq (1 - \delta)e^{-\chi}.$$

Proof. Note first that $\ln(1 - \delta) < -\delta$; hence,

$$\left(\frac{n}{\chi} - 1\right)(\chi - \ln(1 - \delta)) \geq n + \frac{n\delta}{\chi} - (\chi + \delta) \geq n.$$

By making use of the fact that $(1 - 1/x)^{x-1} \geq 1/e$ and simplifying the exponent n as above,

$$\left(1 - \frac{\chi}{n}\right)^n \geq \left[\left(1 - \frac{\chi}{n}\right)^{(n/\chi)-1}\right]^{\chi - \ln(1-\delta)} \geq (1 - \delta)e^{-\chi}.$$

□

The expected optimization time of the (μ, λ) EA on ONEMAX can now be expressed in terms of the mutation rate χ/n and the problem size n assuming some constraints on the population sizes μ and λ. The theorem is valid for a wide range of mutation rates χ/n. In the classical setting of $\chi = 1$, the expected optimization time reduces to $O(n\lambda \ln \lambda)$.

Theorem 10. *The expected optimization time of the (μ, λ) EA with bitwise mutation rate χ/n where $\chi \in (0, n/2)$ and population sizes μ and λ satisfying for any constant $\delta \in (0, 1)$*

$$\frac{\lambda}{\mu} \geq \left(\frac{1+\delta}{1-\delta}\right) e^{\chi}, \quad \text{and} \quad \lambda \geq \frac{4}{\delta^2 e^{\chi}} \ln\left(\frac{24576n(n+1)}{\delta^7 \chi}\right)$$

on ONEMAX *is for any $n \geq (\chi + \delta)/(\chi/\delta)$ no more than*

$$\frac{1536n}{\delta^5}\left(\lambda \ln(\lambda) + \frac{e^{\chi}\ln(n+2)}{\chi(1-\delta)}\right) + O(n\lambda).$$

Proof. Apply the level-based theorem with the same $m := n + 1$ partitions as in the proof of Theorem 7. Since the parameter δ is assumed to be some constant $\delta \in (0, 1)$, it also holds that the parameters a, ε, and c are positive constants. The parameters $\gamma_0, z_1, \ldots, z_m$, and z_* will be chosen later.

To verify that conditions (C1) and (C2) hold for any $j \in [m]$, it is necessary to estimate the probability that operator D produces a search point x' with $j + 1$ one-bits when applied to a population P containing at least $\gamma_0\lambda$ individuals, each having at least j one-bits (formally $|P \cap A^+_{j-1}| \geq \gamma_0\lambda$). Such an event is called a *successful sample*.

Condition (C1) asks for bounds z_j for each $j \in [m]$ on the probability that the search point x' returned by Algorithm 4 contains $j + 1$ one-bits. First chose the parameter setting $\gamma_0 := \mu/\lambda$. This parameter setting is convenient, because the selection step in Algorithm 4 always picks an individual x among the best $\mu = \gamma_0\lambda$ individuals in the population. By the assumption that $|P \cap A^+_{j-1}| \geq \gamma_0\lambda$, the algorithm will always select an individual x containing at least $j + i$ one-bits for some nonnegative integer $i \geq 0$.

Assume without loss of generality that the first j bit positions in the selected individual x are one-bits, and let $k, j < k \leq n$, be any of the other bit positions. If there is a zero-bit in position k or if $i \geq 2$, then a successful sample occurs if the mutation operator flips only bit position k. If there is a one-bit in position k, and if $i = 1$, then the step is still successful if the mutation operator flips none of the bit positions. Since the probability of not flipping a position is higher than the probability of flipping a position, i.e., $1-\chi/n \geq \chi/n$, the probability of a successful sample is therefore in both cases at least

$$(n - j)(\chi/n)(1 - \chi/n)^{n-1}. \tag{2}$$

By Lemma 1, the probability above is at least $z_j := (n-j)(\chi/n)e^{-\chi}(1-\delta)$. The parameter z_* is chosen to be the minimal among these probabilities, i.e., $z_* := (\chi/n)e^{-\chi}(1-\delta)$.

Condition (C2) assumes in addition that $\gamma\lambda < \mu$ individuals have fitness $j+1$ or higher. In this case, it suffices that the selection mechanism picks one of the best $\gamma\lambda$ individuals among the μ individuals and that none of the bits are mutated in the selected individual. The probability of this event is at least

$$\frac{\gamma\lambda}{\mu}(1-\chi/n)^n \geq \frac{\gamma\lambda}{\mu}e^{-\chi}(1-\delta)$$

Hence, to satisfy condition (C2), it suffices to require that

$$\frac{\gamma\lambda}{\mu}\exp(-\chi)(1-\delta) \geq \gamma(1+\delta),$$

which is true whenever

$$\frac{\lambda}{\mu} \geq \left(\frac{1+\delta}{1-\delta}\right)e^{\chi}.$$

To check condition (C3), notice that $\varepsilon = \delta/2$, and $c = \delta^4/384$; hence,

$$a = \frac{\delta^2(\lambda/\mu)}{2(1+\delta)} \geq \frac{\delta^2 e^{\chi}}{2(1-\delta)}, \text{ and}$$

$$ac\varepsilon z_* \geq \frac{\delta^2\chi}{2n}c\varepsilon = \frac{\delta^7\chi}{1536n}$$

Condition (C3) is now satisfied, because the population size λ is required to fulfil

$$\frac{2}{a}\ln\left(\frac{16m}{ac\varepsilon z_*}\right) \leq \frac{4(1-\delta)}{\delta^2 e^{\chi}}\ln\left(\frac{24576mn}{\delta^7\chi}\right) \leq \lambda$$

All conditions are satisfied, and the theorem follows. □

The artificial fitness-level method was first described by Wegener [44]. The original method was designed for the achievement of upper bounds on the runtime of stochastic search heuristics using only one individual such as the (1+1) EA. Since then, several extensions of the method have been devised for the analysis of more sophisticated algorithms. Sudholt introduced the method presented in section "AFL: Lower Bounds" for the obtainment of lower bounds on the runtime [42]. In an early study, Witt [45] used a potential function that generalizes the fitness level

argument of [44] to analyze the (μ+1) EA. His analysis achieved tight upper bounds on the runtime of the (μ+1) EA on LEADINGONES and ONEMAX by waiting for a sufficient amount of individuals of the population to *take over* a given fitness-level A_i before calculating the probability to reach a fitness level of higher fitness. Chen et al. [2] extended the analysis to offspring populations by analyzing the (N+N) EA, also taking into account the take-over process. Recently, Corus and Oliveto [4] proved that steady state (μ+1) GAs using crossover optimise ONEMAX faster than mutation-only EAs by coupling each fitness level with a Markov chain to analyse the diversity of the population on each level. Lehre [23] introduced a general fitness-level method for arbitrary population-based EAs with non-elitist selection mechanisms and unary variation operators. This technique was later generalized further into the level-based method presented in section "Level-Based Analysis of Non-elitist Populations" [6]. The method allows the analysis of sophisticated non-elitist heuristics such as genetic algorithms equipped with mutation, crossover, and stochastic selection mechanisms, both for classical and noisy and uncertain optimization [8].

Drift Analysis

Drift analysis is a very flexible and powerful tool that is widely used in the analysis of stochastic search algorithms. The high-level idea is to predict the long-term behavior of a stochastic process by measuring the expected progress toward a target in a single step. Naturally, a measure of progress needs to be introduced, which is generally called a *distance function*. Given a random variable X_k representing the current state of the process at step k, over a finite set of states S, a distance function $d : S \to \mathbb{R}_0^+$ is defined such that $d(X_k) = 0$ if and only if X_k is a target point (e.g., the global optimum). Drift analysis aims at deriving the expected time to reach the target by analyzing the decrease in distance in each step, i.e., $d(X_{k+1}) - d(X_k)$. The expected value of this decrease in distance, $\Delta_k = \mathbb{E}[d(X_{k+1}) - d(X_k) \mid X_k]$, is called the *drift*. See Fig. 8 for an illustration. If the initial distance from the target is $d(X_0)$ and a bound on the drift Δ (i.e., the expected improvement in each step) is known, then bounds on the expected runtime to reach the target may be derived.

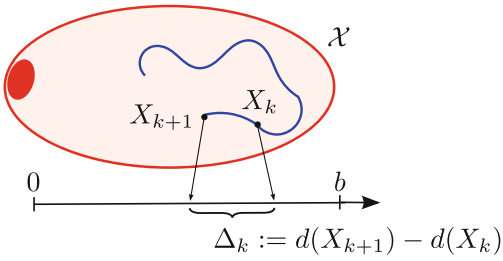


Fig. 8 An illustration of the drift Δ at time step k of a process represented by the random variable X and a distance function d

Additive Drift Theorem

The additive drift theorem was introduced to the field of evolutionary computation by He and Yao [15]. The theorem allows to derive both upper and lower bounds on the runtime of stochastic search algorithms. Consider a distance function $Y_k = d(X_k)$ indicating the current distance, at time k, of the stochastic process from the optimum. The theorem simply states that if at each time step k the drift is *at least* some value $-\varepsilon$ (i.e., the process has moved closer to the target), then the expected number of steps to reach the target is *at most* Y_0/ε. Conversely if the drift in each step is *at most* some value $-\varepsilon$, then the expected number of steps to reach the target is *at least* Y_0/ε (Fig. 9).

Theorem 11 (Additive Drift Theorem). *Given a stochastic process $X_1, X_2, \ldots$ over an interval $[0, b] \subset \mathbb{R}$ and a distance function $d : S \to \mathbb{R}_0^+$ such that $d(X_k) = 0$ if and only if X contains the target. Let $Y_k = d(X_k)$ for all k, define $T := \min\{k \geq 0 \mid Y_k = 0\}$, and assume $\mathbb{E}[T] < \infty$.*
Verify the following conditions:

$$(C1+) \quad \forall k \quad \mathbb{E}[Y_{k+1} - Y_k \mid Y_k > 0] \leq -\varepsilon$$
$$(C1-) \quad \forall k \quad \mathbb{E}[Y_{k+1} - Y_k \mid Y_k > 0] \geq -\varepsilon$$

Then,

1. *If (C1+) holds for an $\varepsilon > 0$, then $\mathbb{E}[T \mid Y_0] \leq b/\varepsilon$.*
2. *If (C1−) holds for an $\varepsilon > 0$, then $\mathbb{E}[T \mid Y_0] \geq Y_0/\varepsilon$.*

An example application of the additive drift theorem follows concerning the (1+1) EA for plateau blocks of functions of unitation of length m positioned such that $k > n/2 + \varepsilon n$.

Theorem 12. *The expected runtime of the (1+1)-EA for a* plateau block *of length m ending at position $k > n/2 + \varepsilon n$ is $\Theta(m)$.*

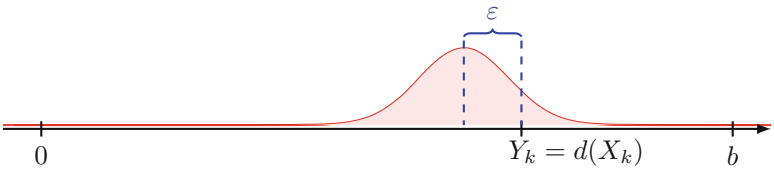


Fig. 9 An illustration of the condition of the additive drift theorem. If the expected distance to the optimum decreases of *at least* ε at each step (i.e., condition C1+), then an upper bound on the runtime is achieved. If the distance decreases of *at most* ε at each step (i.e., condition C1-), then a lower bound on the runtime is obtained

Proof. The additive drift theorem will be applied to derive both upper and lower bounds on the expected runtime. The starting point is a bitstring X_0 with $m + k$ zeroes, and the target point is a bitstring X_t with k zeroes. Choose to use the natural distance function $Y_t = d(X_t) := n - |X_t|$ that counts the number of zeroes in the bitstring. Subtract k from the distance such that target points with k zeroes have distance 0 and the initial point has distance m. As long as points on the plateau are generated, they will be accepted because all plateau points have equal fitness. Given that each bit flips with probability $1/n$, and at each step the current search point has Y_t zeroes and $n - Y_t$ ones, the drift is

$$\Delta_t := \mathbb{E}[Y_t - Y_{t+1} \mid Y_t > 0] = \frac{Y_t}{n} - \frac{n - Y_t}{n} = \frac{2 \cdot Y_t}{n} - 1$$

A lower bound on the drift is obtained by considering that as long as the end of the plateau has not been reached, there are always at least k zeroes that may be flipped (i.e., $Y_t \geq k$). Accordingly for an upper bound, at most $m + k$ zeroes may be available to be flipped (i.e., $Y_t \leq m + k$). Hence,

$$\frac{2k}{n} - 1 \leq \Delta_t \leq \frac{2(m+k)}{n} - 1$$

Then by additive drift analysis (Theorem 11),

$$\mathbb{E}[T \mid Y_0] \leq \frac{m}{(2k)/n - 1} = \frac{mn}{2k - n} = O(m)$$

and

$$\mathbb{E}[T \mid Y_0] \geq \frac{m}{2(m+k)/n - 1} = \frac{mn}{2(m+k) - n} = \Omega(m)$$

where the last equalities hold as long as $k > n/2 + \varepsilon n$. □

Note again that if the plateau block is followed by a gap block, then an upper bound on the expected time to optimize both blocks is achieved by multiplying the upper bounds obtained for each block. This is necessary because points in the gap will not be accepted by the (1+1) EA.

Multiplicative Drift Theorem

In the additive drift theorem, the worst-case decrease in distance is considered. If the expected decrease in distance changes considerably in different areas of the search space, then the estimate on the drift may be too pessimistic for the obtainment of tight bounds on the expected runtime.

Drift analysis of the (1+1) EA for the classical ONEMAX function will serve as an example of this problem. Since the global optimum is the all-ones bitstring and

the fitness increases with the number of ones, a natural distance function is $Y_t = d(X_t) = n - \textsc{OneMax}(X_t)$ which simply counts the number of zeroes in the current search point. Then the distance will be zero once the optimum is found. Points with less one-bits than the current search point will not be accepted by the algorithm because of their lower fitness. So the drift is always positive, i.e., $\Delta_t \geq 0$, and the amount of progress is the expected number of ones gained in each step. In order to find an upper bound on the runtime, a lower bound on the drift is needed (i.e., the worst case improvement). Such worst case occurs when the current search point is optimal except for one zero-bit. In this case the maximum decrease in distance that may be achieved in a step is $Y_t - Y_{t+1} = 1$, and to achieve such progress, it is necessary that the algorithm flips the zero into a one and leaves the other bits unchanged. Hence, the drift is

$$\Delta_t \geq 1 \cdot \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en} := \varepsilon$$

Since the expected initial distance is $\mathbb{E}[Y_0] = n/2$ due to random initialization, the drift theorem yields

$$\mathbb{E}[T \mid Y_0] \leq \frac{\mathbb{E}[Y_0]}{\varepsilon} = \frac{n/2}{1/(en)} = e/2 \cdot n^2 = O(n^2)$$

In section "Artificial Fitness Levels (AFL) and the Level-Based Method," it was proven that the runtime of the (1+1) EA for $\textsc{OneMax}$ is $\Theta(n \ln n)$; hence, a bound of $O(n^2)$ is not tight. The reason is that on functions such as $\textsc{OneMax}$, the amount of progress made by the algorithm depends crucially on the distance from the optimum. For $\textsc{OneMax}$ in particular, larger progress per step is achieved when the current search point has many zeroes that may be flipped. As the algorithm approaches the optimal solution, the amount of expected progress in each step becomes smaller because search points have increasingly more one-bits than zero-bits in the bitstring. In such cases a distance function that takes into account these properties of the objective function needs to be used. For $\textsc{OneMax}$ a correct bound is achieved by using a distance function that is logarithmic in the number of zeroes i, i.e., $Y_t = d(X_t) := \ln(i + 1)$, where a 1 is added to i in the argument of the logarithm such that the global optimum has distance zero (i.e., $\ln(1) = 0$). With such distance measure, the decrease in distance when flipping a zero and leaving the rest of the bitstring unchanged is

$$\ln(i + 1) - \ln(i) = \ln\left(1 + \frac{1}{i}\right) \geq \frac{1}{2i}$$

where the last inequality holds for all $i \geq 1$. Since it is sufficient to flip a zero and leave everything else unchanged to obtain an improvement, the drift is

$$\Delta_t \geq \frac{i}{en} \cdot \frac{1}{2i} = \frac{1}{2en} := \varepsilon$$

Given that the maximum possible distance is $Y_0 \geq \ln(n+1)$, the drift theorem yields

$$\mathbb{E}[T] \leq \frac{Y_0}{1/(2en)} = 2en \cdot \ln(n+1) = O(n \ln n).$$

The multiplicative drift theorem was introduced as a handy tool to deal with situations as the one described above where the amount of progress depends on the distance from the target.

Theorem 13 (Multiplicative Drift Theorem [12]). *Let $\{X_t\}_{t \in \mathbb{N}_0}$ be random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}$. Let T be the random variable that denotes the earliest point in time $t \in \mathbb{N}_0$ such that $X_t = 0$. If there exist $\delta, c_{\min}, c_{\max} > 0$ such that for all $t < T$,*

1. $\mathbb{E}[X_t - X_{t+1} \mid X_t] \geq \delta X_t$ *and*
2. $c_{\min} \leq X_t \leq c_{\max}$,

then

$$\mathbb{E}[T] \leq \frac{2}{\delta} \cdot \ln\left(1 + \frac{c_{\max}}{c_{\min}}\right)$$

The following derivation of an upper bound on the runtime of the (1+1) EA for linear blocks illustrates the multiplicative drift theorem.

Theorem 14. *The expected time for the (1+1)-EA to optimize a linear unitation block of length m ending at position k is $O(n \ln((m+k)/k))$*

Proof. Let X_t be the number of zero-bits in the bitstring at time step t, representing the distance from the end of the linear block. By remembering that increases in distance are not accepted due to elitism, the expected decrease in distance at time step can be bounded by

$$\mathbb{E}[X_{t+1} | X_t] \leq X_t - 1 \cdot \frac{X_t}{en} = X_t \left(1 - \frac{1}{en}\right)$$

simply by considering that if a zero-bit is flipped and nothing else, then the distance decreases by 1. Then the drift is

$$\mathbb{E}[X_t - X_{t+1} | X_t] \geq X_t - X_t \left(1 - \frac{1}{en}\right) = \frac{1}{en} X_t := \delta X_t$$

By fixing $k = c_{\min} \leq X_t \leq c_{\max} = m + k$, the multiplicative drift theorem yields

$$\mathbb{E}[T] \leq \frac{2}{\delta} \cdot \ln\left(1 + \frac{c_{\max}}{c_{\min}}\right) = 2en \ln(1 + (m+k)/k) = O(n \ln((m+k)/k))$$

□

By fixing $1 = c_{\min} \leq X_t \leq c_{\max} = n$, an $O(n \ln n)$ bound on the expected runtime of the (1+1) EA for ONEMAX is achieved.

Variable Drift Theorem

The multiplicative drift theorem is applicable when the drift of a stochastic process is linear with respect to the current position. However, in some stochastic processes, the drift is nonlinear in the current position, i.e.,

$$\mathbb{E}[X_t - X_{t+1} \mid X_t \geq x_{\min}] \geq h(X_t) \tag{3}$$

for some function h. The following variable drift theorem provides bounds on the expectation and the tails of the hitting time distribution of such processes, given some assumptions about the function h.

Theorem 15 (Corollary 1 in [25]). *Let $(X_t)_{t\geq 0}$ be a stochastic process over some state space $S \subseteq \{0\} \cup [x_{\min}, x_{\max}]$, where $x_{\min} \geq 0$. Let $h\colon [x_{\min}, x_{\max}] \to \mathbb{R}^+$ be a differentiable function. Then the following statements hold for the first hitting time $T := \min\{t \mid X_t = 0\}$.*

(i) If $\mathbb{E}[X_t - X_{t+1} \mid X_t \geq x_{\min}] \geq h(X_t)$ and $h'(x) \geq 0$, then

$$\mathbb{E}[T \mid X_0] \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(y)}\,\mathrm{d}y.$$

(ii) If $\mathbb{E}[X_t - X_{t+1} \mid X_t \geq x_{\min}] \leq h(X_t)$ and $h'(x) \leq 0$, then

$$\mathbb{E}[T \mid X_0] \geq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(y)}\,\mathrm{d}y.$$

(iii) If $\mathbb{E}[X_t - X_{t+1} \mid X_t \geq x_{\min}] \geq h(X_t)$ and $h'(x) \geq \lambda$ for some $\lambda > 0$, then

$$\Pr(T \geq t \mid X_0) < \exp\left(-\lambda\left(t - \frac{x_{\min}}{h(x_{\min})} - \int_{x_{\min}}^{X_0} \frac{1}{h(y)}\,\mathrm{d}y\right)\right).$$

(iv) If $\mathbb{E}[X_t - X_{t+1} \mid X_t \geq x_{\min}] \leq h(X_t)$ and $h'(x) \leq -\lambda$ for some $\lambda > 0$, then

$$\Pr(T < t \mid X_0 > 0) < \frac{e^{\lambda t} - e^{\lambda}}{e^{\lambda} - 1} \exp\left(-\frac{\lambda x_{\min}}{h(x_{\min})} - \int_{x_{\min}}^{X_0} \frac{\lambda}{h(y)}\,\mathrm{d}y\right).$$

To illustrate the variable drift theorem, an upper bound on the optimization time of the (1+1) EA on the class of linear functions with bounded coefficients will be derived. More formally, this class of functions contains any function of the form

$$f(x) := \sum_{i=1}^{n} w_i x_i,$$

with bounded, positive coefficients $w_1, \ldots, w_n \in (w_{\min}, w_{\max})$ where $0 < w_{\min} < w_{\max}$.

The drift function h in this example turns out to be linear; hence, the multiplicative drift theorem could have been applied instead.

Theorem 16. *The expected optimization time of the (1+1) EA on linear functions is less than* $t(n) := en(\ln(n) + \ln(w_{\max}/w_{\min}) + 1)$*, and the probability that the optimization time exceeds* $t(n) + ren$ *for any* $r \geq 0$ *is no more than* e^{-r}.

Proof. Define the distance X_t at time k to be the function value that "remains" at time k, i.e.,

$$X_t := \left(\sum_{i=1}^{n} w_i\right) - \left(\sum_{i=1}^{n} w_i x_i^{(t)}\right) = \sum_{i=1}^{n} w_i \left(1 - x_i^{(t)}\right),$$

where $x_i^{(t)}$ is the i-th bit in the current search point at time t. For any $i \in [n]$, assume that the mutation operator flipped only bit position i and no other bit positions, an event denoted by the symbol $\mathcal{E}_i$. If $x_i^{(t)} = 0$, then bit position i flipped from 0 to 1 and the distance reduced by w_i. Otherwise, if $x_i^{(t)} = 1$, then bit position i flipped from 1 to 0, the new search point was not accepted, and the distance reduced by 0. Hence, the distance always reduces by $w_i(1 - x_i^{(t)})$ when event $\mathcal{E}_i$ occurs. Using the law of total probability, and noting that the distance can never increase, one obtains

$$\begin{aligned}\mathbb{E}[X_t - X_{t+1} \mid X_t = r] &\geq \sum_{i=1}^{n} \Pr(\mathcal{E}_i \mid X_t = r)\, \mathbb{E}[X_t - X_{t+1} \mid \mathcal{E}_i \wedge X_t = r] \\ &\geq \left(\frac{1}{n}\right)\left(1 - \frac{1}{n}\right)^{n-1} \sum_{i=1}^{n} w_i \left(1 - x_i^{(t)}\right) \geq \frac{r}{en}.\end{aligned}$$

Therefore $\mathbb{E}[X_t - X_{t+1} \mid X_t] \geq h(X_t)$ for the function $h(x) = x/en$ which has derivative $h'(x) = 1/en$. By Theorem 15 part (i), it follows that

$$\mathbb{E}[T \mid X_0] \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(y)}\, \mathrm{d}y$$

$$
\begin{aligned}
&= \frac{w_{\min}}{h(w_{\min})} + \int_{w_{\min}}^{nw_{\max}} \frac{en}{y}\,\mathrm{d}y \\
&= en + en(\ln(nw_{\max}) - \ln(w_{\min})) \\
&= en(\ln(n) + \ln(w_{\max}/w_{\min}) + 1) =: t(n).
\end{aligned}
$$

Furthermore, Theorem 15 part (iii) with $\lambda := 1/en$ gives for any $r \geq 0$

$$\Pr(T \geq t(n) + enr) \leq e^{-r}.$$

□

Negative-Drift Theorem

The drift theorems presented in previous subsections are designed to prove polynomial bounds on the runtime of stochastic search algorithms. For this a positive drift toward the optimum is required. On the other hand, a negative drift indicates that the stochastic process moves away from the optimum in expectation at each step. In such a case, it is unlikely that the algorithm is efficient for the function it is attempting to optimize. Rather, an exponential lower bound on the runtime could probably be proved. The negative-drift theorem is the standard technique used for the purpose.

Apart from a negative drift, the theorem also requires a second condition showing that large jumps are unlikely. The intuitive reason for this second condition is that if large jumps were possible, then the algorithm could maybe be able to jump to the optimum even if in expectation it is drifting away. For technical reasons also large jumps heading away from the optimum need to be excluded (see [36]) (Fig. 10).

Theorem 17 (Negative-Drift Theorem). *Let X_t, $t \geq 0$ be the random variables describing a Markov process over the state space S, and denote the increase in distance in one step $D_t(i) := (X_{t+1} - X_t | X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exist an interval $[a, b]$ of the state space and three constants $\delta, \varepsilon, r > 0$ such that for all $t \geq 0$, the following conditions hold:*

1. $\Delta_t(i) = \mathbb{E}[D_t(i)] \geq \varepsilon$ *for* $a < i < b$
2. $\Pr(|D_t(i)| = j) \leq \frac{1}{(1+\delta)^{j-r}}$ *for* $i > a$ *and* $j \geq 1$

Then there exists a constant c^* such that for $T := \min\{t \geq 0 : X_t \leq a | X_0 \geq b\}$ it holds $Pr(T \leq 2^{c^*(b-a)}) = 2^{-\Omega(b-a)}$.

The following theorem is an example application showing exponential runtime of the (1+1) EA for the NEEDLE function.

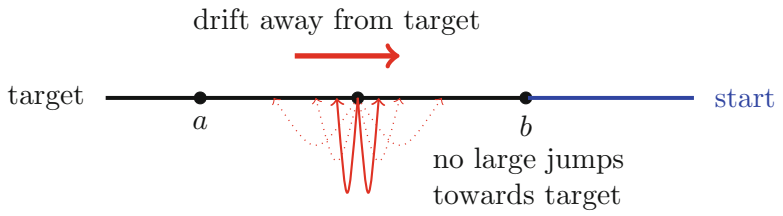


Fig. 10 An illustration of the two conditions of the negative-drift theorem

Theorem 18. *Let $\eta > 0$ be constant. Then there is a constant $c > 0$ such that with probability $1 - 2^{-\Omega(n)}$, the (1+1) EA on* NEEDLE *creates only search points with at most $n/2 + \eta n$ ones in 2^{cn} steps.*

Proof. Apply the negative-drift theorem and set the interval $[a, b]$ as $a := n/2-2\gamma n$ zeroes and $b := n/2 - \gamma n$ zeroes, with γ a positive constant. By Chernoff bounds the probability that the initial random search point has less than $n/2 - \gamma n$ zeroes is $e^{-\Omega(n)}$, implying that the algorithm starts outside the interval $[a, b]$ as desired. The remainder of the proof shows that the two conditions of the drift theorem hold.

Since the interval is a plateau, all created points have the same fitness and are accepted. Given that the current search point has i zeroes and that each bit is flipped with probability $1/n$, the drift is

$$\Delta_t(i) = \frac{n-i}{n} - \frac{i}{n} = \frac{n-2i}{n} \geq 2\gamma := \varepsilon$$

where the last inequality is achieved because in the interval there are always at least b zeroes, i.e., $i \geq n/2 - \gamma n$. Concerning the second condition for standard bit mutation, the probability of flipping j bits decreases exponentially with the number of bits j

$$\Pr\left(|\Delta_t(i)| \geq j\right) \leq \binom{n}{j}\left(\frac{1}{n}\right)^j \leq \left(\frac{n^j}{j!}\right)\left(\frac{1}{n}\right)^j \leq \frac{1}{j!} \leq \left(\frac{1}{2}\right)^{j-1}$$

and the condition holds for $\delta = r = 1$. By the negative-drift theorem, the optimum is found within $2^{c*(b-a)} = 2^{cn}$ steps with probability at most $2^{-\Omega(b-a)} = 2^{-\Omega(n)}$. □

The proof can be generalized to plateau blocks of functions of unitation positioned such that $k + m < (1/2 - \varepsilon)n$.

Theorem 19. *The time for the (1+1) EA to optimize a* plateau block *of length m at position such that $k + m < (1/2 - \varepsilon)n$ is at least $2^{\Omega(m)}$ with probability at least $1 - 2^{-\Omega(m)}$.*

Proof. The proof follows the same arguments as the proof of Theorem 18 by setting $b := m + k$ zeroes and $a := k$ zeroes. □

Conclusions

Drift analysis dates back to 1892 when it was applied for the analysis of stability equilibria in ordinary differential equations [28]. The first use of drift techniques for the runtime analysis of stochastic search heuristics was performed by Sasaki and Hajek to analyze simulated annealing on instances of the maximum matching problem [41]. Drift analysis was applied in a considerable number of applications in evolutionary computation after He and Yao introduced the additive drift technique to the field [15]. Several extensions for the analysis of more sophisticated algorithms and processes have been since devised. The multiplicative drift method introduced in section "Multiplicative Drift Theorem" is due to Doerr et al. [12]. An improved version introduced by Doerr and Goldberg allows to derive bounds also on the probability to deviate from the expected runtime. Witt recently introduced a multiplicative drift theorem to achieve lower bounds on the runtime [46]. While the multiplicative drift theorem may only be used when the drift is linear, a variable drift theorem was introduced to deal with cases where the expected progress is nonlinear [21] with respect to the current position. A general variable drift theorem with tail bounds was introduced in [25]. This theorem subsumes most existing drift theorems, and a special case of this theorem was given in section "Variable Drift Theorem" (see Theorem 15).

Oliveto and Witt proposed the negative-drift theorem presented in section "Negative-Drift Theorem" to derive exponential lower bounds on the runtime of randomized search heuristics [35]. This theorem was the main tool used in the first analysis of the standard *simple genetic algorithm (SGA)* [37, 38]. Finally, Lehre extended the negative-drift theorem to allow a systematic analysis of population-based heuristics using non-elitist selection mechanisms [22].

Final Overview

This chapter has presented the most commonly used techniques for the time complexity analysis of stochastic search heuristics. Example applications have been shown concerning simple evolutionary algorithms on classical test problems. The same techniques have allowed the obtainment of time complexity results on more sophisticated function classes, such as standard combinatorial optimization problems. The reader is referred to [32] for an overview of such advanced results concerning evolutionary algorithms and ant colony optimization. Several other techniques have been used to analyze stochastic search heuristics, such as typical runs, family trees, martingales, probability generating functions, and branching processes. For an introduction to these tools for the analysis of evolutionary algorithms, the reader is referred to a recent extensive monograph [17] which

covers a more extensive set of techniques than is possible in this book chapter. Apart from EAs and ACO for discrete single-objective optimization, several other aspects of stochastic search optimization have been investigated theoretically, such as simulated annealing, evolution strategies for continuous optimization, particle swarm optimization (▸ Chap. 40, "Particle Swarm Optimization for the Vehicle Routing Problem: A Survey and a Comparative Analysis"), memetic algorithms (▸ Chap. 20, "Memetic Algorithms"), and multi-objective optimization techniques (▸ Chap. 7, "Multi-objective Optimization"). These topics are overviewed in [1]. Significant results have been achieved recently that have not yet been covered in monographs and edited book collections. Among these, certainly worth mentioning are the recent considerable advances in black box optimization [11, 24], which may be regarded as a complementary line of research to runtime analysis. Rather than determining the time required by a given heuristic for a problem class, black box optimization focuses on determining the best possible performance achievable by any search heuristic for that class of problems. A recent interesting variation to classical runtime analysis is to determine the expected solution quality achieved by a stochastic search heuristic if it is only allowed a predefined budget [19]. Fixed budget computation analyses are driven by the consideration that in practical applications only a predefined amount of resources are available and the hope is to use such resources at best to achieve solutions of the highest quality. Recent years have witnessed considerable advances in the theory of artificial immune systems (AISs). Results concerning standard AISs, such as the B-cell algorithm, for classical combinatorial optimization problems have appeared [18, 19] and the Opt-IA [7] together with analyses of sophisticated AIS operators such as stochastic aging mechanisms [34]. Complexity analyses of standard steady-state GAs [4, 10], parallel evolutionary algorithms [43], hyper-heuristics [27], and genetic programming [26, 29, 33] have also recently appeared. Finally, systematic work has been carried out in unifying theories of evolutionary algorithms and population genetics [39].

Cross-References

- ▸ Ant Colony Optimization: A Component-Wise Overview
- ▸ Evolutionary Algorithms
- ▸ Genetic Algorithms
- ▸ Memetic Algorithms
- ▸ Multi-objective Optimization
- ▸ Particle Swarm Methods
- ▸ Theory of Local Search

Acknowledgments The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE) and by the EPSRC under grant agreement no EP/M004252/1 (RIGOROUS).

References

1. Auger A, Doerr B (eds) (2011) Theory of randomized search heuristics. World scientific
2. Chen T, He J, Sun G, Chen G, Yao X (2009) A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. IEEE Trans Syst Man Cybern B 39(5):1092–1106
3. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, London
4. Corus D, Oliveto PS (2017, in press) Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2017.2745715
5. Corus D, He J, Jansen T, Oliveto PS, Sudholt D, Zarges C (2017) On easiest functions for mutation operators in bio-inspired optimisation. Algorithmica 78(2):714–740
6. Corus D, Dang DC, Eremeev AV, Lehre PK (2017) Level-based analysis of genetic algorithms and other search processes. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2017.2753538
7. Corus D, Oliveto PS, Yazdani D (2017) On the runtime analysis of the Opt-IA artificial immune system. In: GECCO'17: proceedings of the 2017 annual conference on Genetic and evolutionary computation, pp 83–90
8. Dang DC, Lehre PK (2015) Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms. In: Proceedings of the 2015 ACM conference on foundations of genetic algorithms XIII – FOGA'15. ACM Press, New York, pp 62–68
9. Dang DC, Lehre PK (2015) Simplified runtime analysis of estimation of distribution algorithms. In: Proceedings of the 2015 on genetic and evolutionary computation conference – GECCO'15. ACM, New York, pp 513–518
10. Dang D, Friedrich T, Kötzing T, Krejca MS, Lehre PK, Oliveto PS, Sudholt D, Sutton AM (2016, In press) Escaping local optima using crossover with emergent diversity. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2017.2724201
11. Doerr B, Winzen C (2014) Reducing the arity in unbiased black-box complexity. Theor Comput Sci 545:108–121
12. Doerr B, Johannsen D, Winzen C (2010) Drift analysis and linear functions revisited. In: IEEE congress on evolutionary computation (CEC'10), pp 1967–1974
13. Droste S, Jansen T, Wegener I (2002) On the analysis of the (1+1) evolutionary algorithm. Theor Comput Sci 276:51–81
14. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
15. He J, Yao X (2001) Drift analysis and average time complexity of evolutionary algorithms. Artif Intell 127(1):57–85
16. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
17. Jansen T (2013) Analyzing evolutionary algorithms. The computer science perspective. Springer, Berlin/Heidelberg
18. Jansen T, Zarges C (2012) Computing longest common subsequences with the b-cell algorithm. In: Coello CAC, Greensmith J, Krasnogor N, Liò P, Nicosia G, Pavone M (eds) Artificial immune systems – proceedings of the 11th international conference, ICARIS 2012, Taormina, 28–31 Aug 2012. Lecture notes in computer science, vol 7597. Springer, pp 111–124
19. Jansen T, Zarges C (2014) Reevaluating immune-inspired hypermutations using the fixed budget perspective. IEEE Trans Evol Comput 18(5):674–688
20. Jansen T, Oliveto PS, Zarges C (2011) On the analysis of the immune-inspired b-cell algorithm for the vertex cover problem. In: Liò P, Nicosia G, Stibor T (eds) Artificial immune systems – proceedings of the 10th international conference, ICARIS 2011, Cambridge, 18–21 July 2011. Lecture notes in computer science, vol 6825. Springer, pp 117–131

21. Johannsen D (2012) Random combinatorial structures and randomized search heuristics. PhD thesis, Univesity of Saarland
22. Lehre PK (2010) Negative drift in populations. In: Parallel problem solving from nature – PPSN XI, proceedings of the 11th international conference, Kraków, 11–15 Sept 2010, part I, pp 244–253
23. Lehre PK (2011) Fitness-levels for non-elitist populations. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation – GECCO'11. ACM Press, New York, p 2075
24. Lehre PK, Witt C (2012) Black-box search by unbiased variation. Algorithmica 64(4): 623–642
25. Lehre PK, Witt C (2014) Concentrated hitting times of randomized search heuristics with variable drift. In: Ahn H, Shin C (eds) Algorithms and computation – proceedings of the 25th international symposium, ISAAC 2014, Jeonju, 15–17 Dec 2014. Lecture notes in computer science, vol 8889. Springer, pp 686–697
26. Lissovoi A, Oliveto PS (2018, in press) On the time and space complexity of genetic programming for evolving Boolean conjunctions. In: AAAI-18: proceedings of the thirty-second AAAI conference on artificial intelligence
27. Lissovoi A, Oliveto PS, Warwicker JA (2017) On the runtime analysis of generalised hyper-heuristics for pseudo-boolean optimisation. In: GECCO'17: proceedings of the 2017 annual conference on genetic and evolutionary computation, pp 849–856
28. Lyapunov AM (1892) The general problem of the stability if motion (in Russian). Doctoral dissertation, University of Kharkov, Kharkov Mathematical Society, 250 p
29. Mambrini A, Oliveto PS (2016) On the analysis of simple genetic programming for evolving boolean functions. In: Proceedings of the 2016 European conference on genetic programming – EuroGP'16. Lecture notes in computer science. Springer, pp 99–114
30. Mitzenmacher M, Upfal E (2005) Probability and computing: randomized algorithms and probabilistic analysis. Cambridge University Press, Cambridge
31. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge
32. Neumann F, Witt C (2010) Bioinspired computation in combinatorial optimization – algorithms and their computational complexity. Springer, Berlin/Heidelberg
33. Neumann F, O'Reilly UM, Wagner M (2011) Computational complexity analysis of genetic programming – initial results and future directions. In: Riolo R, Vladislavleva E, Moore JH (eds) Genetic programming theory and practice IX. Springer, New York, pp 113–128
34. Oliveto PS, Sudholt D (2010) On the runtime analysis of stochastic ageing mechanisms. In: GECCO'14: proceedings of the 2014 annual conference on Genetic and evolutionary computation, pp 113–120
35. Oliveto PS, Witt C (2011) Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica 59(3):369–386
36. Oliveto PS, Witt C (2012) Erratum: simplified drift analysis for proving lower bounds in evolutionary computation. Technical report. arXiv:1211.7184, arXiv preprint
37. Oliveto PS, Witt C (2014) On the runtime analysis of the simple genetic algorithm. Theor Comput Sci 545:2–19
38. Oliveto PS, Witt C (2015) Improved time complexity analysis of the simple genetic algorithm. Theor Comput Sci 605:21–41
39. Paixao T, Badkobeh G, Barton N, Corus D, Dang DC, Friedrich T, Lehre PK, Sudholt D, Sutton AM, Trubenová B (2015) Toward a unifying framework for evolutionary processes. J Theor Biol 383:28–43
40. Rudolph G (1998) Finite Markov chain results in evolutionary computation: a tour d'Horizon. Fundamenta Informaticae 35(1):67–89
41. Sasaki GH, Hajek B (1988) The time complexity of maximum matching by simulated annealing. J Assoc Comput Mach 35(2):387–403

42. Sudholt D (2010) General lower bounds for the running time of evolutionary algorithms. In: Proceedings of the 11th international conference on parallel problem solving from nature (PPSN XI). Lecture notes in computer science, vol 6238. Springer, pp 124–133
43. Sudholt D (2015) Parallel evolutionary algorithms. In: Kacprzyk J, Pedrycz W (eds) Handbook of computational intelligence. Springer, Berlin/Heidelberg, pp 929–959
44. Wegener I (2002) Methods for the analysis of evolutionary algorithms. In: Sarker R, Yao X (eds) Evolutionary optimization. Kluwer Academic, New York, pp 125–141
45. Witt C (2006) Runtime analysis of the (μ+1) ea on simple pseudo-boolean functions. Evol Comput 14(1):65–86
46. Witt C (2012) Optimizing linear functions with randomized search heuristics – the robustness of mutation. In: Dürr C, Wilke T (eds) 29th international symposium on theoretical aspects of computer science (STACS 2012). Leibniz international proceedings in informatics (LIPIcs), vol 14, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, pp 420–431

Part V
Applications

City Logistics

30

Jaume Barceló, Hanna Grzybowska, and Jesús Arturo Orozco

Contents

Introductory Remarks on City Logistics: The Nature of the Problem . . . 888
Logistics and City Logistics . . . 888
The Relevance of City Logistics . . . 889
A Systems Approach to City Logistics . . . 890
Organizational Aspects and Decision-Making in City Logistics . . . 892
Fleet Management and ICT Applications . . . 895
Strategic Decisions in City Logistics . . . 899
Two-Echelon Single-Source Location Problems . . . 899
Two-Echelon Location Routing . . . 905
Operational Decisions: Routing Problems . . . 905
Pickup and Delivery Vehicle Routing Problem with Time Windows . . . 906
Real-Time Management . . . 915
Computation of Feasibility . . . 917
Heuristic Approach . . . 919
Extensions . . . 925
Concluding Remarks . . . 927
Cross-References . . . 928
References . . . 928

J. Barceló (✉)
Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, Barcelona, Spain
e-mail: jaume.barcelo@upc.edu

H. Grzybowska
School of Civil and Environmental Engineering, Research Centre for Integrated Transport Innovation, University of New South Wales, Sydney, NSW, Australia
e-mail: h.grzybowska@unsw.edu.au

J. A. Orozco
Department of Operations Management, IPADE Business School, Mexico City, Mexico
e-mail: jaorozco@ipade.mx

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_55

Abstract

This chapter provides an introductory overview of city logistics systems, highlighting the specific characteristics that make them different from general logistics problems. It analyzes the types of decisions involved in managing city logistics applications, from strategic, tactical, and operational, and identifies the key models to address them. This analysis identifies types of problems, location, location routing, and variants of routing problems with time windows, all those with ad hoc formulations, derived from the constraints imposed by policy and operational regulations, technological conditions, or other specificities of urban scenarios, which result in variants of the classical models that, for its size and complexity, become a fertile field for metaheuristic approaches to define algorithms to solve the problems. Some of the more relevant cases are studied in this chapter, and guidelines for further and deeper insights on other cases are provided to the reader through a rich set of bibliographical references.

Keywords

City Logistics · Decision Support Systems · Location Models · Metaheuristics · Vehicle Routing with Time Windows

Introductory Remarks on City Logistics: The Nature of the Problem

City logistics systems exhibit intrinsic characteristics that differentiate them from the general logistics systems, as part of the supply chain, in terms of the specificities of the scenarios where they occur (urban and large metropolitan areas), as well as for the economic relevance that they have as part of the freight distribution system. This section is aimed at explaining and highlighting these two crucial aspects.

Logistics and City Logistics

Logistics, as defined by the Council of Logistics Management CLM [13], is "that part of the supply chain process that plans, implements, and controls the efficient, effective flow and storage of goods, services, and related information from the point of origin to the point of consumption in order to meet customers' requirements." However, when logistics activities take place in urban areas, they show unique characteristics making them different from the general logistics activities. Thus, in order to differentiate the two phenomena and to highlight the special characteristics, the transport in urban areas, and specifically the freight flows associated with the supply of city centers with goods, is usually referred to as "city logistics," "urban freight distribution," or "last mile logistics."

Taniguchi et al. [51] define city logistics as "the process of totally optimizing the logistics and transport activities by private companies in urban areas while considering the traffic environment, traffic congestion and energy consumption

within the framework of a market economy." This definition can be updated by adding to the concept of the energy consumption the contemporary concerns on the environmental impacts such as emissions, urban noise, vibration, etc., generated by logistic vehicles in urban areas. These impacts affect not only the economy but also the quality of life of the city residents.

An analysis of city logistics activities, and how they are performed, results with an immediate identification of their specific intrinsic characteristics. It allows for establishing substantial differences between logistics and city logistics justifying the entity of city logistics as an individual field with proper identity. Some of the main city logistics characteristics include:

- Spatial restrictions:
  - Urban microstructure determined by the urban network (e.g., one-way streets, dead-end streets, etc.) causes the paths between customers, or between the depot and the customers, to differ depending on the order of the visits.
  - Limited vehicle access. Most cities mandate limited access for some areas of the city for specific types of vehicles during specific time windows.
  - Small-quantity deliveries.
  - High density of delivery points.
- Traffic infrastructure: traffic schemes banning specific turnings, traffic lights and their associated traffic control plans, etc.
- Environmental concerns and sensitivities:
  - Growing role of small specialized urban vehicles (i.e., environmentally friendly vehicles commonly referred to as green vehicles) as a consequence of sustainable urban policies implying energy savings, reduction of emissions, noise, etc.
  - Low automation and critical human role when manual deliveries are necessary.
  - High operational and environmental costs, as a consequence of the human involvement among other reasons.

The Relevance of City Logistics

Urban logistics operations consist of the set of activities related with the distribution of goods and provision of services within an urban area. There is a wide range of examples of urban logistics that includes parcel delivery, material collection, goods storage, waste collection, home delivery services, or electrical appliance reparation services, among others. Current urban activities are far away from what they were a couple of decades ago. New challenges have emerged, new technologies have been developed, and urban population dynamics have changed.

The road system is the main transportation modality in most countries. In Europe, 68% of goods transportation is made by roads (BESTUFS report [6]). Countries with less developed rail networks tend to have higher road utilization. Moreover, a high proportion of all goods are delivered within the cities. Cities as London and

Dublin estimate this proportion in slightly more than 40% (BESTUFS report [6]), while almost 70% of the deliveries are concentrated in Tokyo (Taniguchi et al., [51]). In summary, road transportation is the **main** freight transportation mode, and **high proportion** of the **total** goods transportation happens in **cities**.

The significant impact of commercial vehicles in daily traffic is evident. According to the BESTUFS report [6], about a fifth of a city's traffic flow is made up of commercial trucks. In London, freight transport accounts for 20%–26% of the total traffic, while in Italy this proportion is estimated to be 18%. In the same study, it is reported that urban freight in French cities represents between 13% and 20% of the total traffic. Overall, the BESTUFS [6] estimates that between 15% and 20% of the average flow in cities during rush hours correspond to logistics fleets (i.e., light goods vehicles, heavy goods vehicles, etc.). Consequently, logistics fleets are a net contributor to traffic congestion in urban and metropolitan areas and have a relevant impact on energy consumption and on the quality of life in cities.

Although urban freight distribution represents a small fraction of the total transportation length, the Council of Supply Chain Management Professionals, Goodman [29] estimates that the last mile cost accounts for about 28% of the total transportation cost. In a study made by the European Logistics Association and AT Kearney [21], it is shown that transportation activities account for approximately 43% of the total logistics costs. Assuming that the cost of delivering within urban areas (i.e., the last mile cost) is 28%, we can then estimate that the cost of urban freight activities represents about 12% of the total logistics costs.

A Systems Approach to City Logistics

The complexity of city logistics systems should be addresses from a systemic perspective accounting for all of its components and, what is more important, their interactions determining the dynamics of the system and the way it behaves. Namely, it should be addressed in terms of the conditions, operational but also technological, which are imposed by the relationships among stakeholders, which are synthetically described in this section.

Any systemic approach to city logistics must look at the system as a whole, identify its components, and account for their mutual interactions. For example, consider the impacts that city logistics activities have on traffic congestion, and at the same time, consider how in return the city logistics activities are impacted by traffic congestion and operational constraints of the urban areas.

Thus, the *city logistics models must account for the two-way interaction*. They should include the effects of the city logistics deliveries and commercial activities on urban traffic congestion, and conversely they should include operational constraints in routing and logistics optimization models considering time-varying traffic congestion allowing defining how city logistics activities are impacted by traffic congestion.

Notwithstanding, a systemic approach should neither forget that urban freight distribution is mainly a private sector activity, designed to maximize the profits,

which takes place in a public scenario under rules and constraints determined by public authorities. In the words of Boudoin et al. [10], "today, the city as a geographic and economical space is the centre of attention of decision makers of the private and public sectors, considering that the performance of the logistics system depends also greatly on the decisions taken by these two categories of actors. The urban logistics spaces are in the core of the goods distribution device, as they are interfaces between interurban and urban, private and public, producer and consumer." A systemic approach must also take into account that city logistics is a scenario with various stakeholders with possibly different and conflicting interests.

Public stakeholders will usually be interested in achieving social, economic, environmental, or energetic objectives, while private stakeholders, namely, private shippers and freight carriers, aim to reduce their freight costs and to optimize their traffic flows in accordance with their specific needs, which are not conforming to the objectives of an overall optimization. Therefore, a systemic approach to city logistics must:

1. Be supported by a holistic view in modeling approaches integrating urban and logistic planning which will allow innovative approaches to urban logistics solution.
2. Suitable to be applied in practice in terms of a framework enabling the understanding between stakeholders, fostering new ways of stakeholder collaboration, and providing policy frameworks allowing sustainable business models.
3. Become operational in terms of decision support tools to efficiently implement the desirable policies.

In other words, the conceptual approach to such systemic view must be able to appropriately account for the roles of the main stakeholders and the relationships between them as highlighted in the conceptual diagram in Fig. 1.

Freight operators supply their services under supply conditions and contracts with customers imposing constraints on:

- Fleet routing and scheduling.
- Service times and delivery conditions that could be determined by the characteristics of the delivery (i.e., loading and unloading) point.
- Service time windows imposed either by the customer requirements or by the regulatory conditions determined by local authorities.

The supply regulations established by the local government strongly determine the way in which logistics fleets operate. The restrictions to access inner city, namely, in cities with historical centers, reinforced in most cases by access control systems, usually impose time constraints that determine the routings and scheduling, which can also be affected by the existence, or not, of loading/unloading points and regulations on their use. Furthermore, the regulations on vehicle sizes authorized to operate in urban areas, or green logistics policies imposing thresholds on the acceptable levels of emissions, could determine the fleet compositions.

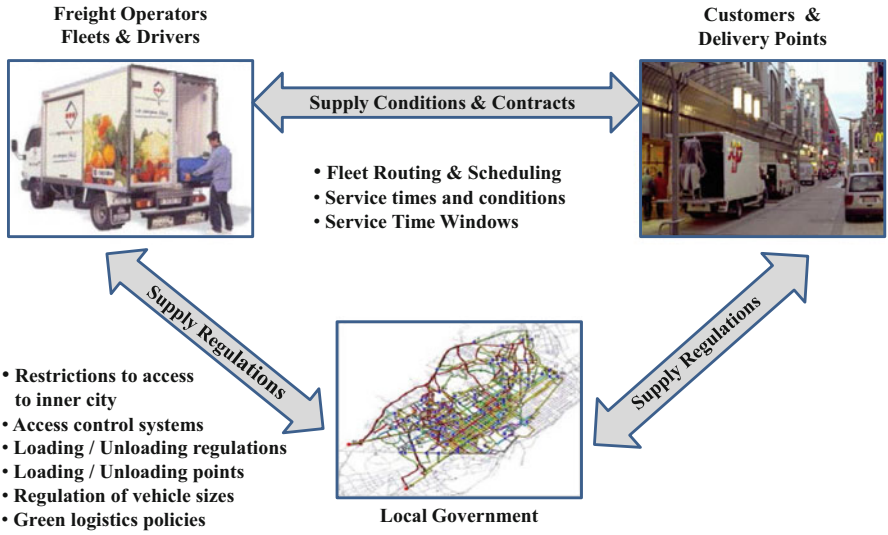


Fig. 1 Stakeholders' interaction in city logistics operations

Organizational Aspects and Decision-Making in City Logistics

The proposed systemic view must be translated in terms of a model, suitably representing the system, which, further than providing a deep understanding on how the system works, should provide support to any type of operational decisions on the system. Two key aspects in this process are, respectively, identify and understand the main logistics operations and identify and distinguish the decision levels in the management of the city logistics system. This view is neatly aligned with the key approach of Operations Research: acquire enough knowledge about a system to understand how the system works, and formulate the system's knowledge in terms of modeling hypothesis that can be subsequently translated in terms of a formal model (usually a mathematical model) which can be used to make decisions about the system. Therefore, understanding the nature of the decisions in city logistics, which of them depend on the nature of the system, and which are conditioned by the relationships among stakeholders is part of such knowledge acquisition to build models of city logistics. Decision levels for fleet management applications (Goel [28]) are primarily:

- **Strategic:**
  - Decisions that usually concern a large part of the organization, have major financial impact, and may have long-term effects: typically concern the design of the transportation system.
  - The **size and mix** of vehicle fleet and equipment.
  - The **type and mix** of transportation services offered.

  - The territory coverage including **terminal location**.
  - Strategic alliances and cooperation, including the integration of information systems.
- **Tactical:**
  - Concerns short- or medium-term activities: typically involve decisions about how to effectively and efficiently use the existing infrastructure and how to organize operations according to strategic objectives.
  - Equipment acquisition and replacement.
  - Capacity adjustments in response to demand forecasts.
  - Static pricing and pricing policies for contract and spot pricing.
  - Acquisition of regularly requested services, including pricing and provisional routing.
  - Long-term driver to vehicle assignments.
  - Cost and performance analysis.
- **Operational:**
  - Commercial vehicle operations underlay a variety of external influences which cannot be foreseen: real-time management decisions have to be made to appropriately react on discrepancies between planned and actual state of the transportation system.
  - Load acceptance.
  - Real-time dispatching considering the actual state of transportation system.
  - Instructing drivers about their tasks.
  - Monitoring the transportation processes, including tracking and tracing and arrival time estimations.
  - Incident management.
  - Observing the state of order processing.
- **Real Time:**
  - Concerns all activities needed to monitor, control, and plan transportation processes.
  - The required information that dispatchers must collect to manage the fleet include vehicle positions and traffic conditions.
  - The decisions that dispatchers must make to manage the fleet include diversion of vehicles from current route to new destinations and insertion of customers into predefined routes.
  - The most challenging task is the generation of dynamic schedules.
  - Determination of plans indicating which vehicle should visit, pick up, deliver, or service a customer and at what time.

As a consequence of the fact that the stakeholders can play different roles in the city logistics scenarios, they can adopt various organizational forms. These organizational forms can be individual with no self-coordination, individual but with different degrees of coordination in which the private stakeholders are the main actors, or super-coordinated where public organizations play the main role currently referred to as goods distribution centers. Each scenario implies different levels of decision from the long-term strategic decisions on the locations of warehouses or

distribution centers to the operational decisions concerning fleets and routing and scheduling of fleet vehicles. Moreover, the real-time decisions on rerouting and rescheduling vehicles as far as traffic and operational conditions change become each time more relevant due to technological evolution. Efficiency in these scenarios may be achieved through better fleet management practices, rationalization of distribution activities, traffic control, freight consolidation/coordination, and deployment of intermediate facilities [36].

This last measure plays a fundamental role in most of the city logistics projects based on distribution system, where transportation to and from an urban area is performed through platforms. These platforms are called city distribution centers (CDC), goods distribution centers, or city terminals (CT) [42]. They are located far from city limits. Freights, directed to a city and arriving by different transportation modes and vehicles, are consolidated at platforms on trucks in charge of final distribution. According to Boccia et al. [8], "these platforms have had a great impact on effectiveness of freight distribution in urban areas, but their use is showing some deficiencies because of two main reasons: their position (often far from final customers) and the constrained structure of urban areas."

To overcome the limits of such single-echelon system, distribution systems have been proposed, where another intermediate level of facilities is added between platforms and final customers. These facilities, referred to as satellites or transit points (Fig. 2), perform no storage activities and are devoted to transfer and consolidate freights coming from platforms on trucks into smaller vehicles, more suitable for distribution in city centers (Crainic et al. [16, 17]). This two-echelon system could determine an increase of costs due to additional operations at satellites. However, these costs should be compensated by freight consolidation, decrement of empty trips, economy of scale, reduction of traffic congestion, and environmental safety.

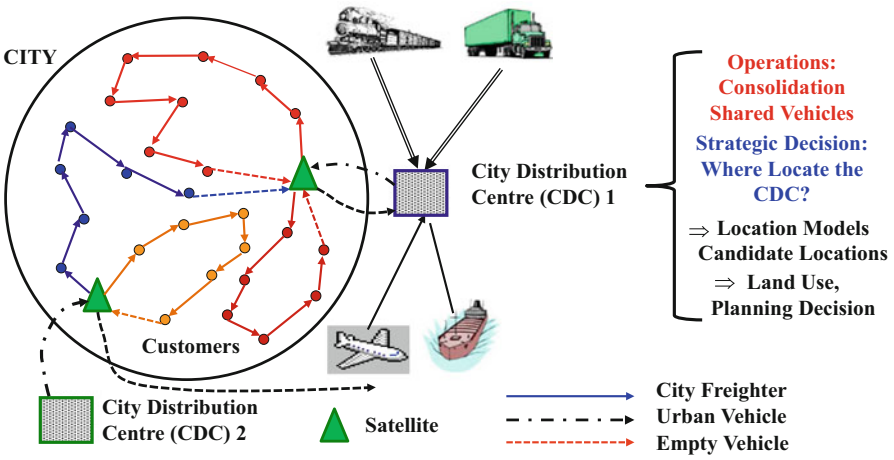


Fig. 2 Two-echelon freight distribution systems and implied decisions

The model simultaneously determines how many depots (CT or CDC) should be open, their locations at the first echelon, how many vehicles (urban vehicles) are needed to supply the satellites, how many satellites should be open at the second echelon, which vehicles (city freighters) should operate from which open satellites, and which customers each vehicle should service.

Most of these decisions must be supported appropriately and that implies the use of suitable models (Giani et al. [24]). Location models (Daskin and Owen [19]) become the key components of strategic decision-making concerning decisions on warehouse or CT locations. Vehicle routing models are at the core of operational and real-time fleet management decisions and processes. Therefore, they deserve a special attention and become the main operational tools to achieve the objectives of private stakeholders, shippers, and freight carriers. Routing problems in urban areas have specific characteristics that differentiate them from generic routing problems. Bodin et al. [9] have coined the term "street routing" to highlight these differences.

Fleet Management and ICT Applications

We have already highlighted the key role of decision-making in city logistics, and the role of models to support strategic decisions has been highlighted in the previous section. However, the tactical and operational decisions should not be forgotten, as they usually determine the quality of the service, which is highly appreciated by customers. Operational decisions in urban areas are changing very fast due to the impacts of the new telecommunication technologies. Further than enabling faster and more flexible operational modes, the new telecommunication technologies induce deep sociological changes in customers' uses, which in turn foster a quick evolution of management policies. The implications of the latter will be discussed in this section.

The growing importance of the real-time fleet management applications is mainly due to the recent significant evolution of pervasive automatic vehicle location (AVL) technologies (Goel [28]). Fleet management in urban areas has to explicitly account for the dynamics of traffic conditions leading to congestions and variability in travel times severely affecting the distribution of goods and the provision of services. An efficient management should be based on decisions accounting for all factors conditioning the problem: customers' demands and service conditions (i.e., time windows, service times, and others), fleet operational conditions (e.g., positions, states and availabilities of vehicles, etc.), and traffic conditions.

Instead of making decisions based on trial and error and operator's experience, a sounder procedure would be to base the decisions on the information provided by a decision support system (DSS). Regan et al. [47, 48] provide a conceptual framework for the evaluation of real-time fleet management systems which considers dynamic rerouting and scheduling decisions implied by operations with real-time information as new orders or updated traffic conditions. The recent advances in information and communications technologies (ICT) have prompted the research on dynamic routing and scheduling problems. At present, easy and fast acquisition and

processing of the real-time information is feasible and affordable. The information, which becomes the input to dynamic models, captures the dynamic nature of the addressed problems allowing for more efficient dynamic fleet management decisions.

Barceló et al. [4, 5] propose and computationally explore a methodological proposal for a decision support system to assist in the decision-making concerning the real-time management of a city logistics fleet in dynamic environments when real-time information is available. Figure 3 depicts the architecture of such a DSS based on a dynamic router and scheduler. The feasibility of the proposed solutions depends on:

- The availability of the real-time information, which we assume will be provided by ICT applications, combined with the knowledge of the scheduled plan with the current fleet and customer status.
- The quality of the vehicle routing models and algorithms to efficiently tackle the available information to provided solutions.

Unlike the classic approach, where routes are planned with the known demand and they are unlikely to be changed throughout the planning period, the real-time fleet management approach assumes that real-time information is constantly

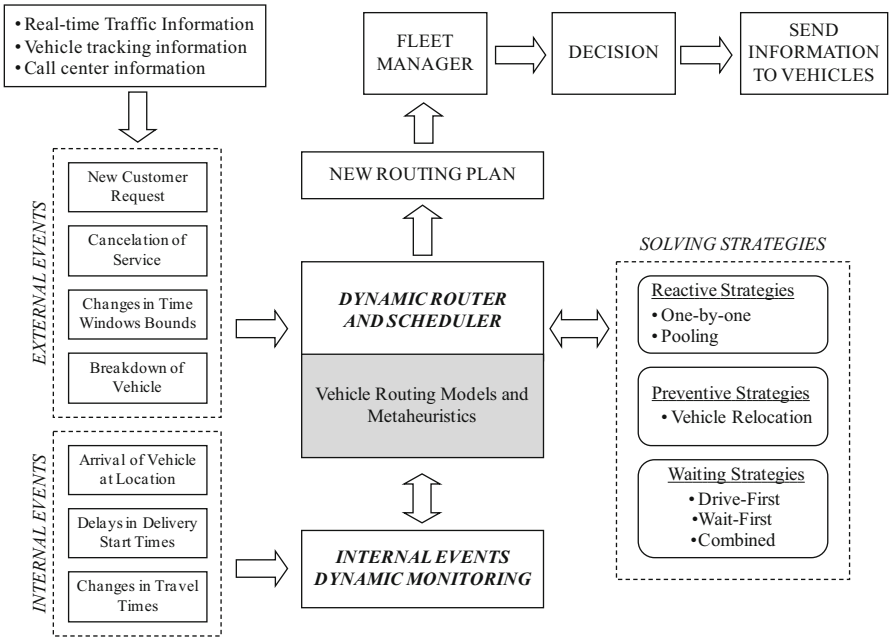


Fig. 3 Logical architecture of a decision support system for real-time fleet management based on a dynamic router and scheduler

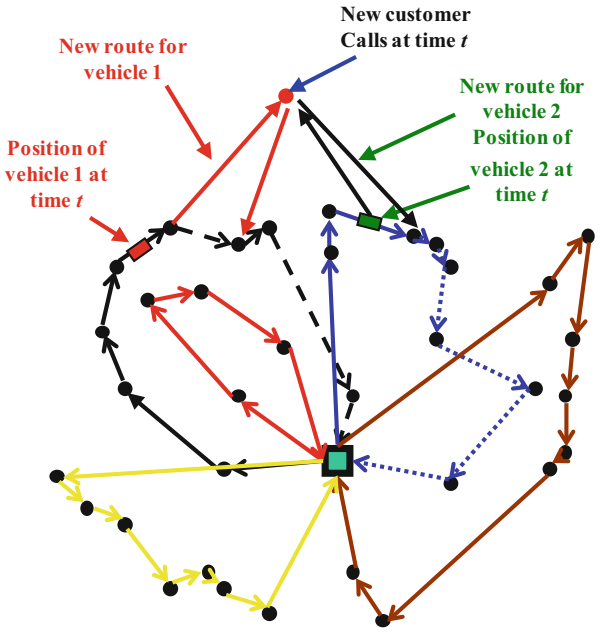


Fig. 4 Dynamic vehicle rerouting in a real-time fleet management system

revealed to the fleet manager who has to decide whether the current routing plan should be modified or not. The process assumes partial knowledge of the demand. At the beginning of the considered time period (i.e., one work day), an initial schedule for the available fleet to serve the known demand is proposed. This initial operational plan can be modified later on, when the operations are ongoing and new real-time information is available. The new information can concern new demands, unsatisfied demands, changes in the routes due to traffic conditions, changes in the fleet availability (i.e., vehicle breakdowns), etc. It constitutes an input to a dynamic router and scheduler (DRS) which provides a proposal of a new dynamic operational plan prepared on the basis of real-time information.

Figure 4 depicts an example of how the conceptual process described in Fig. 3 can be handled by a dynamic routing and scheduling system. The routes, initially assigned to a set of five vehicles, are identified with various colors. The arrows indicate the order in which customers are to be served according to the initial schedule. We assume that vehicles can be tracked in real time. At time t, after the fleet has started to perform its initial operational plan, a new customer calls requiring a service which has not been scheduled. If real-time information, such as positions and states of the vehicles and current and forecasted traffic conditions, is available to the fleet manager, he or she can use it to make a better decision on which vehicle to assign to the new customer and whether the new assignment results in a direct diversion from a route (vehicle 2) or a later scheduling (vehicle 1).

Vehicle routing problem (VRP) techniques constitute a fundament for the transportation, distribution, and city logistics systems modeling. The static version of the VRP has been widely studied in the literature. Among others, an extensive survey on VRP was provided by Fisher [22] and by Toth and Vigo [54]. The static approaches reckon with all the required information to be known a priori and constant throughout the time. However, in most of the real-life cases, a large part of the data is revealed to the decision-maker when the operations are already in progress. Thus, the dynamic VRP assumes partial knowledge of the demand and that new real-time information is revealed to the fleet manager throughout the operational period. A comprehensive review of the dynamic VRP can be found in Ghiani et al. [24]. Psaraftis [45, 46] and Powell et al. [44] contrast the two variants of the problem and clearly distinguish the dynamic VRP from its static version.

A wide variety of vehicle routing problems with time windows (VRPTW) becomes the engine of real-time decision-making processes to address situations where real-time information is revealed to the fleet manager, and he or she has to make decisions in order to modify an initial routing plan with respect to the new needs (Barceló and Orozco [1]). Real-time routing problems are mainly driven by events which are the cause of such modifications. Events take place in time, and their nature may differ according to the type of service provided by a motor carrier. The most common type of event is the arrival of a new order. When a new order is received from a customer, the fleet manager must decide which vehicle to assign to the new customer and what is the new schedule that this vehicle must follow.

A standard practice of introducing the dynamism into the definition of a routing problem is to determine specific features as *time-dependency*. The VRP with time-dependent travel times acknowledges the influence of traffic conditions on the routing planning. An introduction to the problem and a made heuristic development is provided in Malandraki and Daskin [37]. Further algorithm developments for the static VRP with time-dependent travel times can be found in Ichoua et al. [32], Fleishman et al. [23], and Tang [50]. The dynamic VRP with time-dependent information has also been addressed by Chen et al. [12] and Potvin et al. [43].

Thomas and White [53] addressed the problem by assuming that stochastic information was represented by the time of arrival of a new request. The objective of their approach was to find the best policy for selecting the next node to visit that minimizes the total travel time. They called this problem the *anticipatory routing problem* as vehicles anticipate the arrival of a new customer by changing their path if the request is received while they are in transit. The authors assumed that a new service may be delivered only if the reward or benefit is sufficiently high. The problem was modeled as a finite-horizon Markov decision process, and the authors used standard stochastic dynamic programming methods to solve each one of the proposed instances. Thomas [52] extended the results by incorporating waiting strategies with the objective of maximizing the expected number of new customers served. They also modeled the problem using a finite-horizon Markov decision process. The authors proposed heuristic algorithms for real-time decision-making.

In the approach proposed by Barcelo et al. [2, 3], time-dependent information is generated by means of traffic simulation of a real-world urban network, in order to emulate the role of a real-time traffic information system providing reliable real-time information on traffic conditions and short-term forecasts.

In order to test the management strategies and algorithms assuming the availability of the real-time information, the dynamic traffic simulation model emulates the current travel times estimated as a function of the prevailing traffic conditions and the short-term forecast of the expected evolution of travel times that an advanced traffic information system would provide. This is a basis for making more realistic decisions on the feasibility of providing the requested services within the specified time windows. Furthermore, simulation can also emulate real-time vehicle tracking giving access to positions and availabilities of the fleet vehicles, which is the information required by a DRS.

This example was inspired by the modeling framework proposed by Taniguchi et al. [51], including models needed by the authorities to support long-term planning decisions accounting for the already mentioned interactions between city logistics activities and urban congestion.

Strategic Decisions in City Logistics

The current evolution of city logistics systems has raised the interest in the variants of location models to support the concerned strategic decisions. In many cases they are induced by regulations imposing conditions on the location of the CTs or CDCs and restrictions on the vehicle types allowed to operate in the inner city, leading to organizational restructuring of the services from intermediate satellites. The potential scenarios illustrated in Fig. 2 can be generically addressed by two types of location models, described in this section, depending on whether the routings of the service vehicles are explicitly included in the model or not.

Two-Echelon Single-Source Location Problems

The increasing importance of the e-commerce results with interesting organizational decisions. For example, more and more people purchase products online, but are not usually at home at daytime and cannot accept the deliveries, return the unaccepted deliveries, or deliver the parcels to be sent to other customers. As a consequence, there are required alternative solutions for home deliveries.

An example of such alternative solution was described in BESTUFS II [7]: the Packstations used by Deutsche Post and DHL in Germany. Similar solutions were also deployed in other countries. Usually, they are locker facilities installed in specific locations, which provide automated booths, or locker boxes for self-service collection and delivery of parcels.

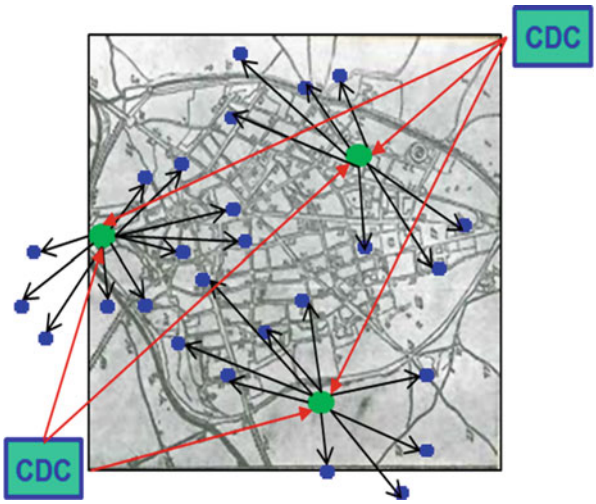


Fig. 5 Two-echelon single-source location problem

In this case, the logistics system can be considered as a particular case of a *two-echelon single-source location problem*, in which special fleet of vehicles service the satellite-automated booths (marked in green in Fig. 5); the special fleet can consist of urban vehicles, such as vans or trucks, fulfilling specific urban regulations regarding sizes and technologies, for example, special vehicles with less than 3.5 tonnes weight, with electrical engines, or propelled by biofuel or low emission fuels. Customers (marked in blue in Fig. 5) travel to the automated locker boxes for service.

In the pilots reported in BESTUFS II [7] the locker boxes have been installed in public spaces (e.g., main station, market place, petrol stations, etc.), and also at parking places of big companies. However, the search for systematic, general solutions can be formulated in terms of two-echelon single-source models. This is the problem arising in a two-stage distribution process, with deliveries being made from first-echelon facilities (e.g., city terminals) to second-echelon facilities (e.g., satellites) and from there to customers. The two-echelon, single-source, and capacitated facility location problem can be considered as an extension of the single-source capacitated facility location problem, dealing with the problem of simultaneously locating facilities in the first and second echelons where:

- Each facility in the second echelon has limited capacity and can be supplied by only one facility in the first echelon,
- Each customer is served by only one facility in the second echelon.

The model simultaneously determines how many depots (CTs or CDCs) should be open, what are their locations at the first echelon, how many vehicles (urban vehicles) are needed to supply the satellites, how many satellites should be open at

the second echelon, which vehicles (city freighters) should operate from which open satellites, and which customers each vehicle should service.

Notation:

$I = \{1, 2, \ldots, m\}$: Set of potential facilities (satellites)
$J = \{1, 2, \ldots, n\}$: Set of customers
$K = \{1, 2, \ldots, nn\}$: Set of potential depots (CDCs)
a_j: demand of customer $j, \forall j \in J$
b_i: capacity of facility (satellite) $i, \forall i \in I$
f_{ik}: cost of assigning facility i to depot $k, \forall i \in I, \forall k \in K$
c_{ijk}: cost of facility i from depot k servicing customer $j, \forall i \in I, \forall j \in J, \forall k \in K$
g_k: cost of setting a depot at location $k, \forall k \in K$

Decision variables:

$$Y_{ik} = \begin{cases} 1 & \textit{if facility i is open and served from depot k}, \ \forall i \in I, \ \forall k \in K \\ 0 & otherwise \end{cases}$$

$$X_{ijk} = \begin{cases} 1 & \textit{if facility i served by depot k services customer j}, \forall i \in I, \forall j \in J, \forall k \in K \\ 0 & otherwise \end{cases}$$

$$Z_k = \begin{cases} 1 & \textit{if a depot is set a location k}, \ \forall k \in K \\ 0 & otherwise \end{cases}$$

The model:

$$p : \text{Min} \sum_{i \in I} \sum_{k \in K} \sum_{j \in J} c_{ijk} x_{ijk} + \sum_{i \in I} \sum_{k \in K} f_{ik} \ y_{ik} + \sum_{k \in k} g_k z_k \tag{1}$$

subject to

$$\sum_{j \in J} a_j x_{ijk} \leq b_i \quad \forall i \in I, \forall k \in K \tag{2}$$

$$\sum_{i \in I} \sum_{k \in K} x_{ijk} = 1 \quad \forall j \in J \tag{3}$$

$$\sum_{k \in k} y_{ik} \leq 1 \ \forall i \in I \tag{4}$$

$$x_{ijk} \leq y_{ik} \ \forall i \in I, \forall j \in J, k \in K \tag{5}$$

$$y_{ik} \leq Z_k \quad \forall i \in I, k \in K \tag{6}$$

$$x_{ijk}, y_{ik}, z_k \in \{0, 1\} \quad \forall i \in I, \forall j \in J, k \in K \tag{7}$$

The objective function (1) includes the total cost of assigning customers to facilities, the cost of establishing facilities, and the cost of opening depots. The side constraint (2) ensures that the customer demand serviced by a facility does not exceed its capacity. The equation (3) ensures that each customer is assigned to exactly one facility. Each facility can be serviced by only one depot, as defined by constraint (4). The inequality (5) states that the assignments are made only to open facilities (i.e., customers are allocated only to open facilities). Lastly, the formulation (6) ensures that facilities are allocated only to open depots.

Tragantalerngsak et al. [55] propose a variety of Lagrangian heuristics for this problem. Taking into account the computational results achieved, one of the most performing Lagrangian decompositions analyzed is based on the possibility of strengthening the resulting subproblems as a consequence of the observation that at least one depot must be open. Therefore, without any loss of generality, there can be used the constraint:

$$\sum_{k \in K} z_k \geq 1 \tag{8}$$

Also, there may be included a constraint which forces to open sufficient facilities to supply all customer demands:

$$\sum_{i \in I} \sum_{k \in K} y_{ik} b_i \geq \sum_{j \in J} a_j \tag{9}$$

This will improve the lower bound provided by the relaxation.

By relaxing the constraints (3) and (6), with Lagrangian multipliers λ and ω, respectively, the problem can be separated into the two following Lagrangian subproblems LRxy and LRz:

$$LR_z : \min \sum_{k \in K} \left(g_k - \sum_{i \in I} \omega_{ik} \right) z_k \tag{10}$$

subject to

$$\sum_{k \in K} z_k \geq 1 \tag{11}$$

$$z_k \in \{0, 1\} \quad \forall k \in K \tag{12}$$

and

$$LR_{xy} : \min \sum_{i \in I} \sum_{k \in K} \sum_{j \in J} (c_{ijk} - \lambda_j) x_{ijk} + \sum_{i \in I} \sum_{k \in K} (f_{ik} + \omega_{ik}) \tag{13}$$

subject to

$$\sum_{j \in J} a_j x_{ijk} \leq b_i \quad \forall i \in I, \forall k \in K \tag{14}$$

$$\sum_{k \in K} y_{ik} \leq 1 \forall i \in I \tag{15}$$

$$\sum_{i \in I} \sum_{k \in K} b_i y_{ik} \geq \sum_{j \in J} a_j \tag{16}$$

$$x_{ijk} \leq y_{ik} \quad \forall i \in I, \forall j \in J, k \in K \tag{17}$$

$$x_{ijk}, y_{ik} \in \{0, 1\} \quad \forall i \in I, \forall j \in J, k \in K \tag{18}$$

Let $g1_k = g_k - \sum_{i \in I} \omega_{ik}$; then problem LR_{xy} can be solved by inspection:

$$z_k = \begin{cases} 1 \text{ if } g1_k \leq 0 \\ 0 \text{ Otherwise} \end{cases} \tag{19}$$

In the case where all $g1_k > 0$, set $z_{k'} = 1$, where $g1_{k'} = \min_k\{g1_k\}$ and $z_k = 0,\ k \neq k'$.

The problem can be reformulated as

$$\min \sum_{i \in I} \sum_{k \in K} v_{ik} y_{ik} \tag{20}$$

subject to

$$\sum_{k \in K} y_{ik} \leq 1 \quad \forall i \in I \tag{21}$$

$$\sum_{i \in I} \sum_{k \in K} b_i y_{ik} \geq \sum_{j \in J} a_j \tag{22}$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in I, k \in K \tag{23}$$

where $v_i k$ is the optimal solution of

$$\min \sum_{j \in J} (c_{ijk} - \lambda_j) x_{ijk} + (f_{ik} + \omega_{ik}) \tag{24}$$

subject to

$$\sum_{j \in J} a_j x_{ijk} \leq b_i \quad \forall i \in I, \forall k \in K \tag{25}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in I, \forall j \in J, k \in K \tag{26}$$

The problem of finding v_{ik} is now a 0-1 *knapsack problem* involving all facilities, but the fact that each facility can be served only from one depot enables to rewrite the problem as follows:

$$\text{Let } v_i = \min_k\{v_i k\} \tag{27}$$

Solve

$$\min \sum_{i \in I} v_i u_i \tag{28}$$

subject to

$$\sum_{i \in I} b_i u_i \geq \sum_{j \in J} a_j \tag{29}$$

$$u_i \in \{0, 1\} \quad \forall i \in I \tag{30}$$

This is a 0-1 knapsack problem with n variables. The lower bound for problem P is then given by $LBD = v(LR_{xy}) + v(LR_z)$. The upper bound UBD can be found from the solution of the problem LR_z as follows:

Let z_k^* be the solution to LR_z and $\tilde{K} = \{k | z_k^* = 1\}$. Solve LR_{xy} again but over the depots in $\tilde{K}$. The solution from this restricted set is then used to find a feasible solution. Let $\mathbf{u}^*$ be the solution to this problem, and denote $\tilde{I} = \{i | u_i^* = 1\}$.

Then

$$y_{ik}^* = \begin{cases} 1 \text{ if } i \in \tilde{I} \text{ and } k \in \tilde{K} \\ 0 \text{ otherwise} \end{cases} \tag{31}$$

The corresponding solution $\boldsymbol{x}^*$ is obtained as the solution giving the v_{ik} coefficients if $i \in \tilde{I},\ k \in \tilde{K}, y_{ik}^* = 1$. Otherwise, $x_{ijk}^* = 0$.

Using these solutions no capacity constraints are violated, but there may be some customers j that were assigned to multiple open facilities or not assigned to any facilities. A *generalized assignment problem* is constructed from these customers, open facilities, and the remaining capacity of open facilities. The solution to these problems reassigns these customers and provides the expected upper bound.

Once the lower bound LBD and upper bound UBD of the optimal objective function have been calculated, the Lagrangian multipliers λ and ω are updated until a convergence criterion is met.

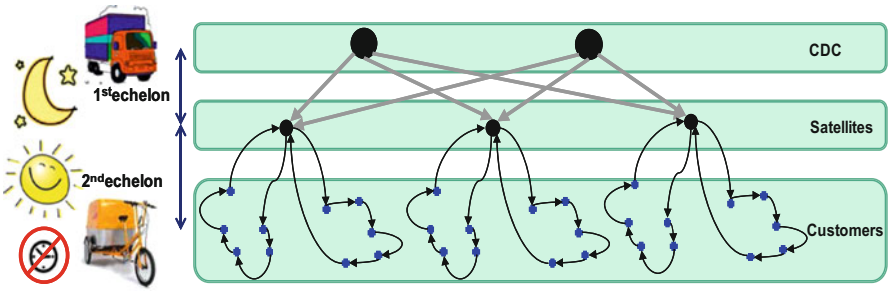


Fig. 6 An example of two-echelon location routing accounting for special urban regulations

Two-Echelon Location Routing

There are more variants of the location problems arising from a combination of organizational decisions and urban regulations on the characteristics and conditions of the logistics fleets, as discussed in section "Organizational Aspects and Decision-Making in City Logistics" and illustrated in Fig. 2. A more appealing version of the addressed location problem is the *two-echelon location routing*. In this case, in the first echelon, city freighters service satellites from city distribution centers. In the second echelon, special small urban vehicles serve customers from satellites in downtown regulated zones.

A special case of the two-echelon location routing, also prompted by the growth of e-commerce in urban areas, is the one depicted in Fig. 6. Here, the services to the satellites from the CDCs are provided by special fleets during the night. During the day, special city freighters (e.g., electrical vehicles, tricycles, etc.) serve the customers from the satellites.

Nagy and Salgy [38] elaborated a rather complete state-of-the-art report on this problem. They proposed a classification scheme and analyzed a number of variants as well as the exact and heuristic algorithmic approaches. Drexl and Scheinder [20] updated this state-of-the-art survey, providing an excellent panoramic overview of the current approaches. Metaheuristics dominate the panorama since they look more appropriate to deal with real instances of the problem. Perhaps one of the most appealing is that of Nguyen et al. [39] based on a GRASP approach combined with learning processes and path relinking. Another interesting heuristic, an approach to a relevant variant of the problem based on a Tabu Search accounting for time dependencies, can be found in Nguyen et al. [40]. The interested reader is directed to these references given that space limitations do not allow including them in this chapter.

Operational Decisions: Routing Problems

According to Taniguchi et al. [51], vehicle routing and scheduling models provide the core techniques for modeling city logistics operations. Once the facilities, or the city logistics centers, have been located, the next step is to decide on the efficient

use of the fleet of vehicles that must service the customers or make a number of stops to pick up and/or deliver passengers or products. Vehicle routing problems constitute a whole world given the many operational variants using them. The book of Toth and Vigo [54] provides a comprehensive and exhaustive overview of routing problems. To illustrate their role in city logistics, we have selected the relevant case of pickup and delivery models with time dependencies, which play a key role in courier services, e-commerce, and other related applications.

Pickup and Delivery Vehicle Routing Problem with Time Windows

Pickup and delivery vehicle routing problem with time windows (PDVRPTW) is a suitable approach for modeling routes and service schedules for optimizing the performance of freight companies in the city logistics context (e.g., the couriers). It is also a good example to demonstrate the operational decisions in the routing problems.

In this problem, each individual request includes pickup and a corresponding delivery of specific demand. The relationships between customers are defined by pairing (also known as coupling) and precedence constraints. The first constraint links two particular customers in a pickup-delivery pair, while the second one specifies that each pickup must be performed before the corresponding delivery.

Therefore, the main objective of the PDVRPTW is to determine, for the smallest number of vehicles from a fleet, a set of routes with a corresponding schedule, to serve a collection of customers with determined pickup and delivery requests, in such a way that the total cost of all the trips is minimal and all side constraints are satisfied. In other words, it consists of determining a set of vehicle routes with assigned schedules such that:

- Each route starts and ends at a depot (a vehicle leaves and returns empty to the depot),
- Each customer is visited exactly once by exactly one vehicle,
- The capacity of each vehicle is never exceeded,
- A pair of associated pickup-delivery customers is served by the same vehicle (pairing constraint),
- Cargo sender (pickup) is always visited before its recipient (delivery) (precedence constraint),
- Service takes place within customers' time window intervals (time windows constraint),
- The entire routing cost is minimized.

In order to describe mathematically the demonstrated PDVRPTW, we define for each vehicle k a complete graph $G_k \subseteq G$, where $G_k = (N_k, A_k)$. The set N_k

contains the nodes representing the depot and the customers, which will be visited by the vehicle k. The set $A_k = \{(i, j) : i, j \in N_k, i \neq j\}$ comprises all the feasible arcs between them. Thus, the problem formulation takes the form

$$\min \sum_{k \in K} \sum_{(i,j) \in A_k} c_{ijk} x_{ijk} \tag{32}$$

subject to

$$\sum_{k \in K} \sum_{j \in N_k \cup \{n+1\}} x_{ijk} = 1 \qquad \forall i \in N^+, \tag{33}$$

$$\sum_{i \in N_k^+} \sum_{j \in N_k} x_{ijk} - \sum_{j \in N_k} \sum_{i \in N_k^-} x_{jik} = 0 \qquad \forall k \in K, \tag{34}$$

$$\sum_{i} \sum_{j \neq i} x_{jik} \leq |S| - 1 \qquad \forall S \subseteq N : |S| \geq 2, \forall k \in K \tag{35}$$

$$\sum_{j \in N_k^+ \cup \{n+1\}} x_{0jk} = 1 \qquad \forall k \in K, \tag{36}$$

$$\sum_{i \in N_k \cup \{0\}} x_{ijk} - \sum_{i \in N_k \cup \{n+1\}} x_{jik} = 0 \qquad \forall k \in K,\ j \in N_k, \tag{37}$$

$$\sum_{i \in N_k^- \cup \{0\}} x_{i,n+1,k} = 1 \qquad \forall k \in K, \tag{38}$$

$$x_{ijk}(z_{ik} + s_i + c_{ijk} - z_{jk}) \leq 0 \qquad \forall k \in K,\ (i, j) \in A_k, \tag{39}$$

$$e_i \leq z_{ik} \leq l_i \qquad \forall k \in K, \quad i \in N_k \cup \{0\}, \tag{40}$$

$$z_{i,k} + c_{i,p(i),k} - z_{p(i),k} \leq 0 \qquad \forall k \in K, i \in N_k^+ \tag{41}$$

$$x_{ijk}(q_{ik} + d_j - q_{jk}) = 0 \qquad \forall k \in K, (i, j) \in A_k, \tag{42}$$

$$d_i \leq q_{i,k} \leq Q \qquad \forall k \in K, i \in N_k^+ \tag{43}$$

$$0 \leq q_{p(i),k} \leq Q - d_i \qquad \forall k \in K, i \in N_k^+ \tag{44}$$

$$_{q0k} = 0 \qquad \forall k \in K, \tag{45}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall k \in K, (i, j) \in A_k, \tag{46}$$

where

N: set of customers, where $N = N^+ \cup N^-, \left|N^+\right| = |N^-|$,
N^+: set of all customers that notify pickup request,
N^-: set of all customers that notify delivery request,
c_{ijk}: nonnegative cost of a direct travel between nodes i and j performed by vehicle k, assuming that $c_{ijk} = c_{jik} \forall i, j \in V$,
i: customer, where $i \in N_k^+$,
$p(i)$: pair partner of customer i, where $p(i) \in N_k^-$,
$_{di}$: customer's demand that will be picked up/delivered at $i/p(i)$, respectively, where $d_i = d_{p(i)}$,
$_{qik}$: vehicle's k capacity occupancy after visiting customer i,
$_{ai}$: arrival time at customer i,
$_{wi}$: waiting time at the customer i, where $w_i = max\{0, e_i - a_i\}$,
$_{zik}$: start of service at customer i by vehicle k, where $z_i = a_i + w_i$.

The nonlinear formulation of the objective function (32) minimizes the total travel cost of the solution that assures its feasibility with respect to the specified constraints. Equation (33) assigns each customer to exactly one route, while formulation (34) is a pairing constraint, which ensures that the visit of each pickup-delivery pair of customers $(i^+, p(i^+))$ is performed by the same vehicle k. The inequality (35) eliminates the possibility of construction of potential sub-tours. The three following constraints secure the commodity flow. Equality (36) defines the depot as every route's source and states that the first visited customer is the one with a pickup request. Likewise, formulation (38) determines the depot as every route's sink, and the last visited customer is the one that demands a delivery service. The degree constraint (37) specifies that the vehicle may visit each customer only once. The schedule concordance is maintained by equations (39) and (40) according to which, in case that a vehicle arrives to a customer early, it is permitted to wait and start the service within the time window interval only. The precedence constraint (41) assures that for each pair of customers the pickup i is always visited before its delivery partner $p(i)$. The next three restrictions express the dependencies between the customers' demands and the vehicles' restrained current and total capacities. Equation (42) indicates that after visiting the customer j, the current occupancy of the carriage loading space of the vehicle k is equal to the sum of the load carried after visiting the preceding customer i and the demand collected at customer j. According to inequality (43), the dimension of current occupancy of the total capacity of the vehicle k after visiting a pickup customer i shall be neither smaller than its demand d_i nor bigger than the entire vehicle's capacity. Similarly, following formulation (44), the current capacity of the vehicle k after visiting a delivery customer $p(i)$ shall never be smaller than zero and bigger than the difference between the total vehicle capacity and the size of its delivery request $d_{p(i)}$. The capacity constraint that considers the depot (45) states that the vehicle does not provide it with any service. The last formulation (46) expresses the binary and nonnegative nature of the problem-involved variable.

The exact algorithms are able to solve to optimality only the VRP problems with small number of customers (Cordeau et al. [15]). The heuristics do not guarantee optimality, but since they are capable of providing, for large-sized problems, a feasible solution in a relatively short amount of time, they strongly dominate among all the methods. What is more, the heuristics are proven to be quite flexible in adaptation to different VRP problem variations, which is of special importance when considering the real-world applications.

Tabu Search (TS) has been applied by many researchers to solve VRP. It is known to be a very effective method providing good, near-optimal solutions to the difficult combinatorial problems. The TS term was firstly introduced by Glover [27], and the concepts on which TS is based on had been previously analyzed by the same author Glover [26]. The main intention for TS creation was the necessity to overcome the barriers, stopping the local search heuristics from reaching better solutions than the local optima and explore intelligently a wider space of the possible outcomes. In this context, TS might be seen as extension of the local search methods or as a combination of them and the specific memory structures. The adaptive memory is the main component of this approach. It permits to flexibly and efficiently search the neighborhood of the solution.

The input to TS constitutes an initial solution created beforehand by a different algorithm (Fig. 7). An initial solution construction heuristic determines tours according to certain, previously established rules, but does not have to improve them. Its characteristic feature is that a route is built successively and the parts already constructed remain unchanged throughout the process of the execution of the algorithm.

Sweep algorithm is a good example of a VRP initial solution construction heuristic. It was introduced by Gillett and Miller [25]. However, its beginnings might be noticed in earlier published work of other authors, e.g., Wren [57] and Wren and Holliday [58]. The name of the algorithm describes its basic idea very well. A route is created in the process of gradually adding customers to a route. The selection of customers to add resembles the process of sweeping by a virtual ray that takes its beginning in the depot. When the route length, capacity, or the other previously set constraints are met, the route is closed, and the construction of a new route is started. The whole procedure repeats until all the customers are "swept" in the routes. A graphic representation of the sweep algorithm is provided in Fig. 8.

The solution provided by the classic sweep algorithm for PDVRPTW most likely will violate precedence and pairing constraints. The initial solution does not have to be feasible since it will be improved later on by TS in the optimization step. However, a feasible initial solution for PDVRPTW can be provided by a modified sweep algorithm accounting for side constraints. As a result, when a customer is met by the sweeping ray, it is added to the currently built route together with its corresponding partner, respecting the precedence constraint. The details on sweep algorithm adapted to provide initial solution for PDVRPTW are presented in Algorithm 1.

Unified Tabu Search heuristic proposed by Cordeau et al. [14] can be used to optimize the initial solution. Originally, it was designed to solve a VRPTW.

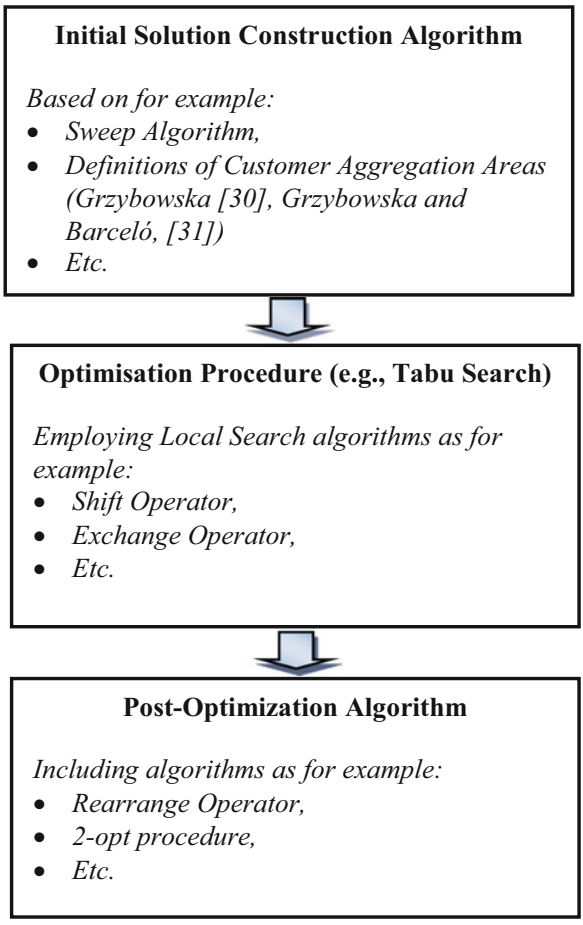


Fig. 7 Composite approach to solve a VRP

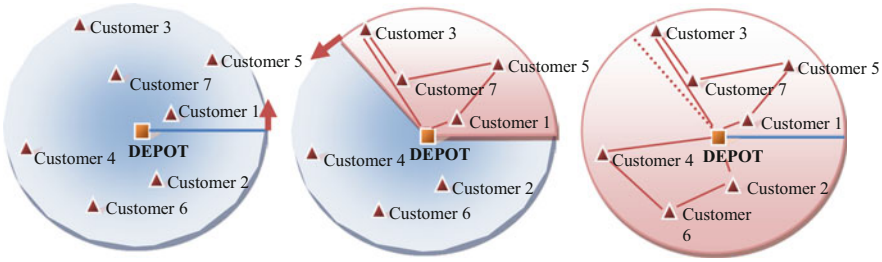


Fig. 8 Sweep algorithm

However, it can be adjusted to solve PDVRPTW. The main change regards the fact that each modification of a route concerns a pickup-delivery pair of customers instead of an individual customer. This affects the architecture and functioning of TS structures (e.g., adaptive memory). In addition, the local search algorithms used in TS need to consider the constraints of pairing and precedence.

TS starts with establishing the initial solution as best ($s^* = s_{\text{ini}}$). In each iteration, the employed local search algorithm defines a *neighborhood* $N(s^*)$ of the current best solution s^* by performing a collection of a priori designed *moves* modifying the original solution. The new solution s that upgrades the original solution, and characterizes with the best result of the objective function, is set as the best ($s* = s$). This repetitive routine (i.e., *intensification phase*) lasts until no further improvement can be found, which is interpreted as reaching the optimum. It uses the information stored in the short-term memory (i.e., *"recency" memory*) recording a number of consecutive iterations in which certain components of the best solution have been present. The short-time memory eliminates the possibility of cycling by prohibiting the moves leading back to the already known results, during certain number of iterations. The moves are labeled as *tabu* and placed at the last position in the *tabu list* – a short-term memory structure of length defined by a parameter called *tabu tenure*. The value of the tabu tenure might be fixed or regularly updated according to the preestablished rules, e.g., recurrently reinitialized at random from the interval limited by specific minimal and maximal value (Taillard [49]). The higher the value of the tabu tenure, the larger the search space to explore is.

Algorithm 1: Sweep algorithm for PDVRPTW

1. Let L be the list of all the customers $L = N \backslash \{0\}$
2. Sort all the customers by increasing angle $\angle A0S$, where S is the current customer, 0 is the depot, and A according to the chosen variant is either randomly chosen or fixed reference point.
3. Divide L into k sub-lists such that each sub-list l satisfies:

$$\angle A0S \in \left(\frac{2l-2-k}{k}\pi, \frac{2l-k}{k}\pi \right], \forall l \in K = \{1, \ldots, k\}$$

4. Sort all the customers in each sub-list l in decreasing order according to the travel cost between the depot and the customer.
5. **If** the sub-list l is not empty, **then** select the first customer, search for its partner, **and** insert both customers in a route in the least cost incrementing position. Respect the precedence constraint.
6. Delete the inserted customers from the sub-lists and go to step 5.
7. Repeat steps 5 and 6 for all the sub-lists.

The consequent choice of the next best solution is determined by the current neighborhood and defines the general direction of the search. The lack of broader perspective in most of the cases leads to finding a solution, which represents a local optimum instead of a global. It is also due to the decision on when to finish the whole procedure improvement (i.e., *stopping criterion*), which might be determined by a designated time limit for the complete performance or by a previously established number of repetitions, which do not bring any further improvement. In order to overcome this handicap, TS stops the local search algorithm and redirects the search in order to explore intelligently a wider space of the possible solutions (i.e., *diversification phase*). This includes permitting operations, which result in deterioration of the currently best solution. This phase requires access to the information accumulated during the whole search process and stored in the long-term memory structure (i.e., *frequency memory*). The success of the complete method depends on the balance established between these two phases, which complement each other instead of competing.

The functioning of the algorithm based on tabu bans is very efficient. However, oftentimes it may result in losing improvement opportunities by not accepting highly attractive moves, if they are prohibited. In such cases, the *aspiration criteria* should be activated. It is an algorithmic mechanism, which consists in canceling the tabu restrictions and permitting the move, if it results in construction of the new solution with the best yet value of the objective function. The implementation of TS is presented by Algorithm 2.

Algorithm 2: Tabu Search

1. Let η be the maximal number of permitted TS iterations.
2. Let α and β be the parameters of preestablished value equal to one.
3. Let Δ be the value of the objective function.
4. Let s be the initial solution.
5. Let s^* be the best solution and $s* = s$
6. Let $c(s^*)$ be the cost of the best solution.
7. **If** the solution s is feasible, **then** set $c(s^*) = c(s)$; **else** set $c(s^*) = \infty$
8. Let $B(s)$ be adaptive memory structure, whose arguments are initially set equal to zero.
9. **For** all the iterations $\kappa < \eta$, execute local search operator and create neighborhood $N(s)$
10. Select a solution from the neighborhood that minimizes the Δ function.
11. **If** solution s is feasible **and** $c(s) < c(s*)$, **then** set $s^* = s$ **and** $c(s^*) = c(s)$
12. Update $B(s)$ according to the performed move.
13. Update α and β

To define the neighborhood of a current solution, TS uses a local search algorithm. Local search algorithms are commonly used as intermediate routines performed during the main search process of more complex heuristics. However,

they constitute individual algorithms. Some of them are referred to as *operators* with specified features (e.g., *shift operator*, *exchange operator*, *rearrange operator*, etc.), while the others possess their own denomination (e.g., the local optimization methods involving neighborhoods, which apply to the original solution a number of modifications equal to k, are known as the *k-opt* heuristics).

Algorithm 3: Pickup-delivery customer pair shift operator

Let s be the current solution containing a set of routes R
Calculate $c(s)$, $q(s)$, and $w(s)$
Let $s^* = s$ be the best solution with cost $c(s^*) = c(s)$
Let $f(s^*) = \infty$ be the value of the objective function of the best solution s^*
Let $\Delta^* = \infty$ be the best value of the objective function
Let $B(s)$ be the empty adaptive memory matrix
For each route $r_1 \in R$ **and for** each route $r_2 \in R$, **such that** $r_1 \neq r_2$
For each pickup-delivery customer pair $m \in r_1$
Remove pair m from route r_1
Set bool TABU = FALSE
If the value of the tabu status in $B(s)$ for r_2 **and** pair m is $\neq 0$, **then** TABU = TRUE
Find the best insertion of pair m in r_2 and obtain new solution s'
Calculate $c(s')$, $q(s')$, and $w(s')$
Let $f(s')$ be the value of the objective function of the new solution s'
Let $p(s')$ be the value of the penalty function of the new solution s'
Let $\Delta = 0$ be the value of the objective function
If $f(s') < f(s^*)$, **then** $\Delta = f(s')$; **else** $\Delta = f(s') + p(s')$
Check the feasibility of solution s'
Set bool AspirationCriteria = FALSE
If $c(s') < c(s^*)$ **and** solution s' is feasible, **then** AspirationCriteria = TRUE
If $\Delta < \Delta^*$ **and** (Tabu = TRUE **or** AspirationCriteria = TRUE), **then** $\Delta^* = \Delta$ **and** the best move was found

Shift operator is a good example of a local search algorithm in TS. Its objective is to remove a customer from its original route and feasibly insert it in another route of the current solution, in such a way that its total cost is minimized. When solving PDVRPTW the shift move includes a pickup-delivery customer pair instead of an individual customer. During the entire search process, all the pickup-delivery pairs are successively moved, and all the possible reinsertion locations in the existing routes are checked. Only the moves which are in accordance with all the side constraints shall be accepted. Algorithm 3 presents the functioning of shift operator in TS for solving PDVRPTW.

The algorithm uses the values of violations of the constraints for the calculation of the objective function, as well as for determining the rate of the penalties, which need to be imposed for not complying with initial restrictions. These values are calculated according to the following formulas:

$$\Delta = f(s) + p(s) \tag{47}$$

$$f(s) = c(s) + \alpha \cdot q(s) + \beta \cdot w(s) \tag{48}$$

$$p(s) = \lambda \cdot c(s) \cdot \sqrt{n \cdot k} \cdot \varphi(s) \tag{49}$$

$$\varphi(s) = \sum_{(m,r) \in B(s)} \rho_{m,r} \tag{50}$$

where

Δ: objective function,
$f(s)$: cost evaluating function for the solution s,
$p(s)$: penalties evaluating function for the solution s,
$c(s)$: cost of solution s,
$q(s)$: vehicle capacity constraint violation function for solution s,
w(s) : time windows constraint violation function for solution s,
α: parameter related to the violation of the vehicle capacity constraint,
β: parameter related to the violation of the time windows constraint,
λ: scaling factor,
n: total number of customers,
k: total number of vehicles
φ(s): parameter controlling the addition frequency,
$B(s)$: adaptive memory matrix,
$\rho_{m,r}$: number of times the pickup-delivery customer pair m has been introduced into route r.

It is a usual practice to complement the TS heuristic with a *post-optimization* step. Its objective is to further improve the solution provided by TS. It is important to note that the final result has to be obligatorily feasible. Local search heuristics are often used for post-optimization purposes and *2-opt heuristic* is a good example. The 2-opt procedure is the most common representative of the family of *k-opt heuristics*. It has been introduced by Croes [18] for solving a Travelling Salesman Problem. The main idea of the 2-opt method is valid for the other k-opt heuristics. In the main, it consists of removing two nonconsecutive arcs connecting the route in a whole and substituting them by another two arcs reconnecting the circuit in such a way that a new solution which fulfills the predefined objectives is obtained. This move is commonly called a *swap* since it consists of swapping two customers in the original sequence. The swap can be performed in accordance with one of the following strategies: (i) search until the first possible improvement is found, and perform the swap, and (ii) search through the entire tour and all possible improvements, and perform only the swap resulting in the best improvement. A graphic representation of 2-opt procedure is provided in Fig. 9.

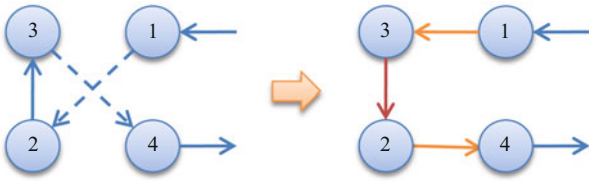


Fig. 9 2-opt algorithm

As in the case of the previously presented algorithms, the classic formulation of the 2-opt algorithm needs to be adapted to solve the PDVRPTW. The final solution needs to be strictly feasible; thus all the side constraints, including pairing and precedence, need to be respected. The PDVRPTW adapted 2-opt algorithm is presented by Algorithm 4.

Algorithm 4: 2-opt algorithm adapted to PDVRPTW

1. Let R be the set of routes defining the current solution s
2. **For** each route $r \in R$
3. Let $r*$ be the best route and $r^* = r$
4. Calculate $c(s^*)$, $q(s*)$, and $w(s*)$
5. **While** best move is not found
6. Let $pos(i)$ be the position of customer i in the sequence of customers of route r
7. Let $pos(j)$ be the position of customer j in the sequence of customers of route r
8. **For** each customer i and j in route r **such that** $pos(j) = pos(i) + 2$
9. Create new empty route r'
10. **For** each customer h in route r
11. **If** $pos\ (h) \leq pos$(i) **or if** $pos(h) \geq pos(j + 1)$, **then** append customer h to r'; **else** append customer h at position $pos(i) + pos(i) - pos(h) + 1$ to r'
12. Calculate $c(s')$, $q(s')$, and $w(s')$ and check the feasibility of route r'
13. **If** $c(r') < c(r*)$ **and** route r' is feasible, **then** $c(s^*) = c(s')$ **and** the best move was found

Real-Time Management

Under this category we cluster what are considered the most relevant city logistics services made available by the pervasive penetration of the ICT technologies. These technologies made it possible for the demand to occur anywhere and at any time. This entails a request for the system to provide the capability of suitably responding to the demand in real time, and in a way that at the time satisfies the quality requirements of the customers, and also provides the most

suitable benefit to the company. This section describes the main characteristics highlighting the dynamic aspects of the problem, which must be addressed by efficient computational time-dependent versions of the ad hoc algorithms. This is an area of applications characterized by the synergies between technologies, decision-making, and sophisticated routing algorithms. The domain of application, however, has recently been expanded to account also for emerging versions of public transport services, special cases of demand-responsive transport services, which can be formally formulated in similar terms (just replacing freight or parcels by persons). This application will be considered in section "Extensions".

Let us suppose that an initial operational plan has been defined with k routes and that there is an *advanced traffic information system* providing real-time information that allows us to calculate current travel times for every pair of clients. Since most of real-world fleets are usually equipped with *automatic vehicle location* (AVL) technologies, it also assumed that the fleet manager is capable of knowing the exact location of the vehicles at every time and has a two-way communication with the fleet.

In our proposed *rerouting algorithm* (RRA), we compute the feasibility of the remaining services while we keep track of the current state and location of the vehicles. In order to simplify the tracking process, we consider two approaches: either to compute feasibility once a vehicle finishes a service and informs about its current situation to the fleet manager or in a periodical manner after updating the fleet's situation. In the first approach, the trigger for the computations is only the departure of the vehicle, but there might be risks when vehicles take longer times to complete the service. The second approach is computationally more intensive, but it may prevent future failures in the service when unexpected service times are present.

In the case of the vehicle routing with time windows, there are two critical factors that define the feasibility of a route: capacity and time windows. Assuming that no variations in demand are expected (e.g., more units to pick up), capacity can be neglected as the corresponding route (static or dynamic) is built taking into account this constraint. In such case, time windows constraints become more critical as there are many uncontrollable parameters (e.g., unexpected traffic conditions, delayed service, etc.) that may affect the feasibility of a service and, therefore, the performance of the route.

Figure 10 describes the full dynamic monitoring process. On one hand, we have the advanced traffic information system which provides current and forecasted travel times of the road network to the fleet manager. On the other hand, we have the fleet management center that is able to communicate with the vehicles and receive their position along with other data such as current available capacity and status.

These two sources are inputs to a dynamic tracking system which through a desired strategy (periodic, after service or both) computes the current feasibility of the routing plan. If an unfeasible service is detected, the RRA is triggered to find a feasible current plan. If a customer cannot be allocated to a vehicle that is on the road, the RRA tries to assign an idle vehicle housed at the depot. If no vehicles are available, a penalty is applied.

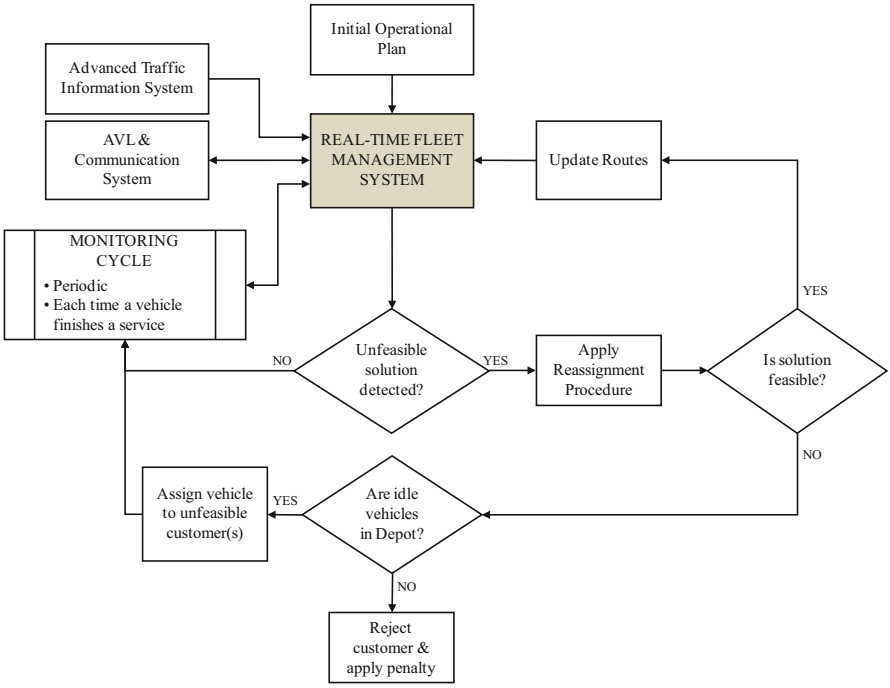


Fig. 10 Rerouting algorithm through dynamic tracking of the fleet

Computation of Feasibility

Computing the current feasibility of a route depends on the state of the vehicle at the time of evaluation. If a vehicle is traveling to the next scheduled customer or departing from a customer's location after service, the feasibility of the route is obtained by computing the expected arrival times to the next customers on the route. If at the time of evaluation, a vehicle has already started a service or waiting to start it, we must first estimate the departure time from the current location and then the arrival times in the rest of the route.

Under our approach, we assume that drivers send messages to the fleet management center when they arrive to and start or end a service with a customer. Therefore, at the time of evaluation, if a vehicle providing service has not finished under the estimated time lapse, the algorithm assumes that the remaining service time is the expected time used in the computation of the route. That is, if a service is assumed to take 10 min and, at the time of evaluation, the vehicle has 14 min without sending the end-of-service signal, the algorithm assumes, in a preventive way, that the vehicle needs another 10 min to complete the service.

Formally, the feasibility conditions can be computed as follows. Let $\hat{a}_i$ be the estimated arrival time of a vehicle at customer i, E_i the lower bound of the time window of customer i, e_i the estimated service start time at customer i, S_i the

estimated service time at customer i, and $T_{ij}(t)$ the travel time between customer i and j when vehicle departs at time t. Let V_k be a dummy node in the network representing vehicle k which route is given by $\{1, 2, \ldots, i, i+1, \ldots, n-1, n\}$.

If at time t_e, the vehicle has just finished a service at customer i or it is traveling to customer $i+1$, the estimated arrival and service start times at the next scheduled customers can be computed as follows:

$$\hat{a}_i = \begin{cases} t_e + T_{V_k,h}(t_e) & \text{if } h = i+1 \\ e_{h-1} + S_{h-1} + T_{h-1,h}(e_{h-1} + S_{h-1}) & \text{if } h = i+2, i+3, \ldots, n \end{cases} \tag{51}$$

$$e_h = \max\{E_h, \hat{a}_h\}, \quad \text{for } h = i+1, i+2, \ldots, n \tag{52}$$

On the other hand, if at time t_e, the vehicle is waiting to start service or providing service to customer i, we first compute d_i, the expected departure time from customer i. Let a_i be the real arrival time of a vehicle at customer i's location. Therefore, d_i is computed as follows:

$$d_i = \begin{cases} t_e + S_i & \text{if } \max\{\mathrm{E_i}, a_i\} + S_i < t_e \\ \max\{E_i, a_i\} + S_i & \text{otherwise} \end{cases} \tag{53}$$

In this case, the expected arrival times in subsequent customers $h = i+1$, $i+2, \ldots, n$ are given by:

$$\hat{a}_h = \begin{cases} d_i + T_{i,i+1}(d_i) & \text{if } h = i+1 \\ e_{h-1} + S_{h-1} + T_{h-1,h}(e_{h-1} + S_{h-1}) & \text{if } h = i+2, i+3, \ldots, n \end{cases} \tag{54}$$

where e_h is computed as in (2). The service at customer h becomes then *unfeasible* if $\hat{a}_h > L_h$, where L_h is the upper bound of the required time window in customer h. Figure 11 depicts a situation when one of the services becomes unfeasible.

The proposed RRA consists of computing feasibility conditions every time a vehicle has finished a service and is ready to depart from its current location. If one or more customers are detected to be in an unfeasible sequence, they are withdrawn and reinserted in the route using a greedy *dynamic insertion* heuristic (DINS). A re-optimization procedure is then applied using a *Tabu Search*-based metaheuristic (DTS). The greedy insertion heuristic is described in the following subsection.

If DINS algorithm does not find a feasible assignment to the vehicles on route, it allocates the withdrawn order to an idle vehicle in the depot. If no vehicles are available, the algorithm rejects the customer. The pseudo-code of the RRA is the following:

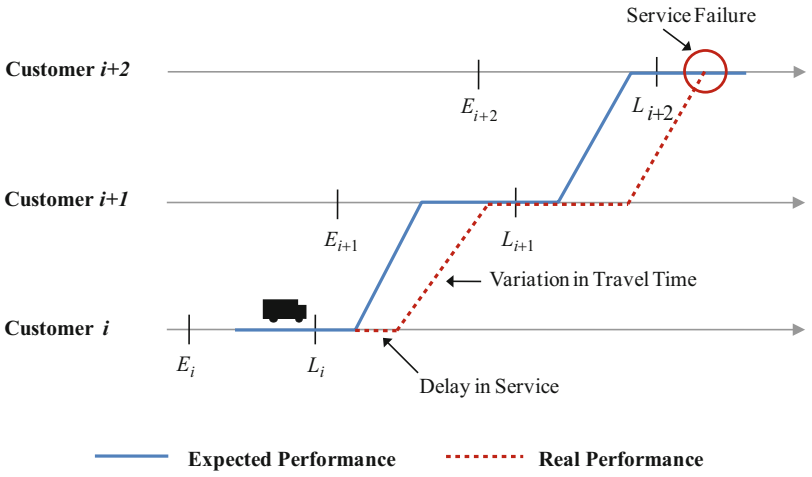


Fig. 11 Example of service failure due to delays [41]

Algorithm 5: Rerouting algorithm

After a vehicle k finishes a service:

1. Compute feasibility conditions for all unvisited customers in the route of vehicle k
2. Let U_k be the set of unvisited customers which are now unfeasible in current scheduling
3. **If** $U_k = \varnothing$, **then** STOP; **else** continue to step 4
4. **For** each customer $u \in U_k$, withdraw u from route applying operator DELETE(u)
5. **For** each customer $u \in U_k$, reinsert u in current routes applying DINS algorithm
6. **If** reinsertion is not possible for a customer, **then** create new route with available vehicle (if any)
7. Apply DTS algorithm to re-optimize routes.

The RRA assumes that all customers that are found to be unfeasible have the same importance, ranking, or priority. Therefore, if there is more than one unfeasible customer, step 5 selects randomly, with equal probability, the next customer to be inserted in the routes.

Heuristic Approach

The RRA is a two-phase methodology that first identifies customers that are likely to be not serviced on time and then reassigns them in order to optimize the service

level of the fleet. After the identification of customers, the algorithm withdraws those customers from the routes and uses them to build a set of customers U_k to be reassigned. The reassignment algorithm works by first inserting customers in U_k with a greedy insertion heuristic, and then, routes are re-optimized using a Tabu Search metaheuristic.

Dynamic Insertion Heuristic (DINS)

The heuristic used in the DRS is derived from the basic insertion heuristic proposed in Campbell and Savelsbergh [11], where every unrouted customer is evaluated at every insertion point. The evaluation of this movement consists in checking the feasibility and profitability of every insertion, which is the negative of the amount of additional travel time added to the route. The customer with a feasible insertion having the maximum profitability is then selected and inserted in the route. We have adapted this heuristic by introducing some new elements to reflect the dynamics of the operations of a fleet in an urban environment.

The objective of DINS heuristic is to insert a new client into the current routing plan once the vehicles have started the services. The general idea of the algorithm is simple. When a new client arrives to the system, the algorithm checks the current state of the vehicles. Then, routes with insufficient time to visit the new client within their schedule are rejected. Finally, the algorithm searches for the least cost feasible insertion in the candidate routes. If no feasible insertion is possible and there are idle vehicles at the depot, a new route is created including the new customer.

We assume that there are n routed customers in R different routes at the beginning of the time planning horizon. We also suppose that there is a single depot with a homogeneous fleet of vehicles with capacity Q. We define the following notation:

$T_{i,j}(t)$: travel time between customers i and j when vehicle departs at time t,
V_r : vehicle assigned to route r, $r = 1, 2, .., R$,
q_r : total demand assigned to route r,
d_i : demand of customer i,
E_i : time window lower bound of customer i,
L_i : time window upper bound of customer i,
e_i : earliest service start time at customer i,
l_i : latest service start time at customer i,
S_i : service time at customer i.

The time window of a customer i is denoted by (E_i, L_i). The values e_i and l_i refer to the earliest and latest time a service can take place at customer i, and they must satisfy the following condition: $E_i \leq e_i \leq l_i \leq L_i$. If we wish to insert customer w at time instant $t > 0$, the vehicles of the fleet may have one of the following three states:

1. The vehicle is in service at some customer i (SER).
2. The vehicle is moving to the next planned customer on the route or waiting at the customer location to start service within the time window (MOV).
3. The vehicle is idle at the depot, without a previously assigned route (IDL).

The state of a vehicle will let us know when a vehicle should be diverted from its current route, be assigned to a new one if it is idle, or keep the planned trip. Whenever a new customer arrives, the status of each vehicle must be known to compute travel times from their current positions to the new customer.

The first iteration of the algorithm consists of rejecting those routes that are definitely infeasible. To do this, it is necessary to compute the total available time of the routes, i.e., the sum of the available time (if any) of a vehicle on the assigned route. We define the available time of a vehicle as those periods of time where (i) a vehicle is waiting to provide service to a customer or (ii) a vehicle has finished the scheduled route and is returning to the depot. This is, in fact, the maximum slack time a vehicle has available to travel and service customer w.

The calculation of the available route time can be done by estimating the arrival times at each of the scheduled customer locations that has not been served at the time of insertion of the customer w. The arrival time a_i at customer i is computed from the current position of the vehicle, and we assume that the vehicle will start service as soon as it arrives, if the customer's time window is already open; otherwise, the vehicle waits. The calculation of arrival times can be made as follows.

If, at instant t, the vehicle is moving toward the next customer i, then the estimated arrival time is given by $a_i = t + T_{Vr,i}(t)$. The arrival times at subsequent customers on the route are computed as follows:

$$a_i = \max\{E_{i-1}, a_{i-1}\} + S_{i-1} + T_{i-1,i}(\max\{E_{i-1}, a_{i-1}\} + S_{i-1}) \tag{55}$$

Hence, the available time of a vehicle in route r at some node i (including the depot) is the difference between the associated lower limit of the time window and the arrival time of the vehicle to that location. That is, $W_i^r = \max\{E_i - a_i, 0\}$. Therefore, the total available time of a route r is given by $AT_{v_r} = \sum_i W_i^r$.

From each route, we choose the unvisited node that is the closest to the new client. If the travel time from that node to the new customer is greater than the total available time AT_{Vr}, then we reject the route. If every route is rejected, a new route must be created for this new customer. The routes obtained, as a result of this previous iteration, are *possibly feasible* in the sense that vehicles serving those routes have enough time to travel to the new customer. This is a necessary condition, but not sufficient, because we have to check for time windows feasibility. The second major iteration of the algorithm consists, therefore, in checking the feasibility and profitability of an insertion in every arc of the possible feasible routes. The feasible insertion with the highest profitability is then selected, and the corresponding route must be updated.

Feasibility. In order to evaluate the feasibility of the insertion of customer w between nodes i and $i+1$, we must compute e_w and l_w, the expected earliest and latest service start times, respectively. The earliest service start time at the inserted customer w is given by:

$$e_w = \begin{cases} \max\{E_w, t + T_{V_r,w}(t)\} & \text{if } V_r = i \\ \max\{E_w, e_i + S_i + T_{i,w}(e_i + S_i)\} & \text{otherwise} \end{cases} \tag{56}$$

The latest service start time is calculated through a backward procedure starting with the depot (the last node to be visited by the vehicle) as follows:

$$l_w = \min\left\{L_w, \tilde{t}_w - S_w\right\} \tag{57}$$

where $\tilde{t}_i = \arg\max_t \{t + T_{i,i+1}(t) - l_{i+1}\}$, $\forall t \leq l_{i+1} - T_{i,i+1}(t)$ is the latest possible departure time that allows the vehicle to arrive to the next scheduled customer $i+1$ at time l_{i+1}. If $e_w \leq l_w$ and $D_w < Q - q_r$, the insertion is feasible.

Profitability. The profit of an insertion is defined as the negative of the additional travel time incurred from inserting a customer in a route. If customer w is to be inserted between nodes i and $i+1$, the profitability of this insertion is given by:

$$\begin{aligned} \text{Profit} = -\lfloor T_{i,w}(\max\{E_i, a_i\} + S_i) + T_{w,i+1}(\max\{E_w, a_w\} + S_w) \\ - T_{i,i+1}(\max\{E_i, a_i\} + S_i)\rfloor \end{aligned} \tag{58}$$

If no insertion is possible and there are available vehicles at the depot, then a new route that includes customer w is created. If the whole fleet of vehicles is already occupied, then the call is rejected, and a penalty may be applied. Given a customer w to be inserted at time t into a set R of routes, the pseudo-code of the DINS heuristic is as follows:

The solution obtained by DINS heuristic can be further optimized by applying DTS.

Dynamic Tabu Search (DTS)

DTS constitutes an adaption of the *Unified Tabu Search* (UTS) heuristic proposed by Cordeau et al. [14]. The modifications of the original method were made in order to include the dynamic aspects of the addressed problem. One of the most important modifications regards the engagement of the operators, which perform the local search only on the unvisited customers of the scheduled routes.

Similarly as UTS, DTS is an iterative process searching for the best solution s^*. In each iteration, the algorithm searches for the best non-tabu solution s' in the neighborhood space $N(s)$ of current solution s, which minimizes the value of the

Algorithm 6: DINS heuristic

1. **For** each route r in R **do**:
Compute available time AT_{Vr}
Find travel time from w to closest neighbor
If available time is not enough, **then** reject route r
Else, add route r to R', the set of possible feasible routes
2. **For** each route r in R' **do**:
Check the status of the vehicle and set its position as the starting node of the route.
For each arc $(i, i+1)$ of the remaining route sequence **do**:
If insertion of w is feasible and profit is improved **then**:
Store current profit and insertion places
3. **If** insertion is possible **then:**
Insert w in least cost arc and update selected route.
4. **Else,** create a new route with an available idle vehicle (if any).

cost function $f_D(s)$ or satisfies the aspiration criteria. The solutions found in the search process are evaluated by the cost function:

$$\mathrm{f_D}(s) = c_D(s) + \alpha q(s) + \beta d_D(s) + \gamma w_D(s) \tag{59}$$

where

$c_D(s)$: estimated total travel time of the routes for the pending scheduled customers,
$q(s)$: total violation of the vehicles' capacity,
$d_D(s)$: total violation of the routes duration,
w_D(s): total violation of the time windows constraint of the customers to be visited.

The adaptive memory matrix $B(s)$ denotes the attribute set of a solution s and is defined as follows: $B(s) = \{(i,k)$, where customer i (or a pair of customers in the case of PDVRPTW) is visited by vehicle $k\}$.

The penalty function $p(s)$ was added to the objective function in order to diversify the search of the solutions toward new or not thoroughly explored areas. For each solution s' such that $f_D(s') \geq f_D(s)$, the penalty is calculated as follows:

$$\mathrm{p}(s') = \lambda c_D(s') \sqrt{n_u m} \sum_{(i,k) \in B(s')} \rho \tag{60}$$

where

n_u: number of customers that have not been visited (or pairs of customers in the case of PDVRPTW),
m: current number of routes,

λ : positive value parameter to control the intensity of diversification,
ρ_{ik}: addition frequency parameter equal to the number of times the attribute (i, k) has been added to a solution.

The value of the addition frequency parameter, which controls the intensity of diversification, is updated regularly and at the end of each iteration of DTS.

The aspiration criteria defined for the DTS accepts a solution if its cost improves the best solution found so far. Also, in order to achieve quick online solutions, we added a stopping criterion which takes into account the rate of improvement in the search process. When the κ number of iterations has been reached without observing an improvement larger than ε, then the search process is stopped.

Similarly as in UTS (Cordeau et al. [14]), in DTS we employ a relaxation mechanism facilitating the exploration of the solution space. The mechanism is particularly useful in the cases when tight constraints are defined (e.g., hard time windows, pairing, and precedence constraints). The relaxation is achieved by dynamically adjusting the values of the parameters: α, β, and γ. When the solution s' is feasible, the value of the best feasible solution is updated, and the current values of the three parameters are reduced by dividing them by the factor $(1 + \delta)$. In the contrary case, the current values of the parameters are increased by multiplying them by the same factor. δ is a parameter of fixed value.

DTS was designed in a way, so that it becomes a flexible framework in which various local search operators can be embedded. As a result, depending on the addressed problem, a different operator can be enabled. Algorithm 7 provides a general description of the DTS. The reader interested in details is directed to Barceló et al. [4,5].

Algorithm 7: Dynamic Tabu Search

1. **If** solution s is feasible **then**:
2. Set $s^* = s, \alpha = 1, \beta = 1, \gamma = 1$
3. Set $c(s^*) = c(s)$ **else** set $c(s^*) = \infty$
4. **For** $i = 1, \ldots, \eta$ **do**:
5. Select solution $s' \in N(s)$ that minimizes the objective function $\Delta = f(s') + p(s')$ such that s' is not tabu **or** satisfies aspiration criteria
6. **If** s' is feasible **and** $c(s') < c(s^*)$, **then** set $s^* = s$' **and** $c(s^*) = c(s')$
7. **If** $q(s') = 0$, **then** $\alpha = \alpha/(1 + \delta)$, **else** $\alpha = \alpha(1 + \delta)$
8. **If** $\mathrm{d}(s') = 0$, **then** $\beta = \beta/(1 + \delta)$, **else** $\beta = \beta(1 + \delta)$
9. **If** $\mathrm{w}(s') = 0$, **then** $\gamma = \gamma/(1 + \delta)$, **else** $\gamma = \gamma(1 + \delta)$
10. Set $s = s$'
11. Update α, β and γ
12. **If** number of iterations without improvement $= \kappa$, **and** last improvement $< \varepsilon$, **then** STOP

Extensions

The variants of pickup and delivery problems with time windows and time-dependent link travel times discussed in section "Operational Decisions: Routing Problems", as core components of the real-time fleet management applications analyzed in section "Real-Time Management" (as we have already mentioned), provide another area of application in urban scenarios which can be considered advanced implementations of demand-responsive transport systems. Examples of such could be the special shared taxi services provided by Uber [56] or Kutsuplus, uber-like minibus demand-responsive transit system experimented in Helsinki KUTSUPLUS [33].

A summary description of the Real-Time Multiple Passenger Ridesharing system and how it works could be the following. A customer asks for a service at a given time; the service could consist of either picking up a passenger (or various passengers) or a parcel, at a given location pickup location, at a given time (or within a time window) and delivering the passenger(s) or the parcel, at a delivery location at a given time (or within a time window). The service vehicles already in operation follow routes individually calculated depending on the customers' initial and final positions, the current traffic conditions, and the defined passengers' time windows.

Figure 12 illustrates conceptually how the demand-responsive mobility services work. Let's assume a fleet consisting of two vehicles V1 and V2, both with initially assigned plans which have been previously optimized. At a given time a customer C1 calls the system for a service, the customer's location is automatically recorded by the system. The customer tells the system which is his/her time expectancy of pickup, in other words the time window (e_1, l_1) of his/her waiting time expectancy (e.g., e_1 could be the time at which the call is made, and l_1 the latest time he/she expects to be picked up). The customer also informs the system of his/her destination that is delivery point, D1, and likely the time at which he/she would approximately like to arrive at the destination, let's say $(a_1 \pm \varepsilon_1)$, where ε_1 represents an acceptable slack time.

The system, which is aware of locations and status of each vehicle in the fleet (e.g., fleet vehicles are GPS tracked, and the ICT functions inform the system on the current level of occupancy of the vehicle, i.e., the number of passengers, their destinations, and time constraints), as well as the current network conditions (e.g., a real-time traffic information system keeps the decision support system updated about travel times, congestions, incidents, and so on), determines which of the vehicles is the most appropriate to provide the service to customer C1 both in terms of quality of the service and profitability. Let's assume that on the basis of the available information, the dispatching system assigns the service to vehicle V1 and the route R1 to pick up the customer and take him or her to his/her destination.

In a similar way, let's assume that customer C2, who is located at C2, has to be picked up within the time window (e_2, l_2) and wants to travel to destination D2. He

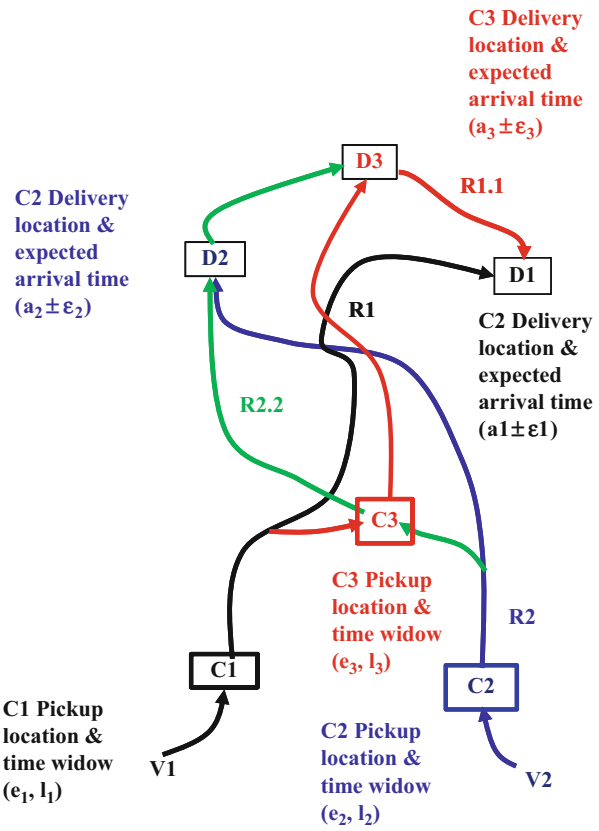


Fig. 12 Example of demand-responsive mobility services

or she expects to arrive around $(a_2 \pm \varepsilon_2)$ and is assigned to vehicle V2 that will follow route R2 to pick him/her up and travel to the destination D2.

The critical situation showing how the performance of the entire system depends on the quality of the decision support system that manages the fleet, and dispatches the services, arises when some time later, let's say at time t, a new customer C3, at location C3, asks for a pickup service within the time window (e_3, l_3) to travel to destination D3 where he/she expects to arrive around $(a_3 \pm \varepsilon_3)$.

In order to provide the new customer the requested service, the system faces many alternatives. First, it can open a new route assigning one of the empty vehicles of the fleet to the new client, but this action may imply a high cost. A better alternative would be to assign one of the en route vehicles that are closer to the client, if the time constraints of the customers that are already being served allow this. In this example, both proposed solutions accept a diversion policy, so, either of the vehicles could divert from their original routes to serve the new client and return to their originally assigned schedules.

One of the alternatives is to assign the new customer service to the vehicle V1, which exchanges its route R1 for the new route R1.1 (described in the figure with red arrows). This vehicle picks up customer C3 within the accepted time window, and then it drives to the D3 destination to deliver the customer. After this, vehicle V1 follows its new route and delivers customer C1 at the corresponding destination D1 taking into account the time constraints of all the served customers. The second possibility is that the new service of customer C3 is assigned to vehicle V2, which diverts its route to pick up the customer by following route R2.2 (described in the figure with green arrows). In this case, the vehicle delivers customer C2 at its destination D2 first, and then, it goes to the destination D3 of customer C3. Also in this case, we need to take into account the time constraints with respect to all involved customers. The decision will be made accounting for criteria that, while ensuring the quality of the service provided to the customers, achieve the maximum system profitability.

The management of the system is based on a decision support system similar to the one described in section "Real-Time Management". It allows making decisions on which vehicle to assign to each new coming customer. The dynamic rerouting is a particular case of the pickup and delivery problem with time windows, with time-dependent travel times, similar to the one described in section "Operational Decisions: Routing Problems".

Linares et al. [34, 35] provide details on a simulation study on the performance of these types of systems as a function of the expected demand for service relative to the total demand of conventional car trips in the selected urban scenario, the size of fleet providing the service (i.e., 500, 750, 1000... vehicles), and the capacity of the fleet vehicles, six or eight passengers.

Concluding Remarks

The main objective of this chapter is to introduce the main characteristics of city logistics problems that make them a special category – clearly differentiated from the general logistics problems. The analysis of these characteristics and their peculiarities regarding the type of decisions to make (i.e., strategic, tactic, and operational) allows to identify various classes of problems (i.e., location, location routing, routing with time windows, etc.) each one becoming a fertile domain for heuristics, given the complexity and size of the problem to solve in real life. A selected set of examples of such heuristics has been described in this chapter, with special attention paid to those particularly relevant to urban scenarios due to the imposed constraints such as the regulatory policies imposed by the authorities, time constraints imposed by the conditions of servicing customer in these scenarios, and the time dependencies implied by the variability of travel times in urban congestion. For the interested reader the chapter is complemented with a wide set of references.

Cross-References

- GRASP
- Matheuristics
- Supply Chain Management
- Tabu Search

References

1. Barceló J, Orozco A (2010) A decision support system to assist the design and evaluation of city logistics applications with real-time information. TRB10-3526, presented at the 89th TRB annual meeting, Washington, DC
2. Barceló J, Grzybowska H, Pardo S (2005) Combining vehicle routing models and microscopic traffic simulation to model and evaluating city logistics applications. In: The proceedings of the 16th mini-EURO conference and 10th meeting of EWGT, Italy
3. Barceló J, Grzybowska H, Pardo S (2007) Vehicle routing and scheduling models, simulation and city logistics. In: Dynamic fleet management. Springer, New York, pp 163–195
4. Barcelo J, Grzybowska H, Orozco JA (2008) A simulation based decision support system for city logistics applications. In: 15th World congress on intelligent transport systems and ITS America's 2008 annual meeting, New York
5. Barceló J, Orozco JA, Grzybowska H (2013) Making real-time fleet management decisions under time dependent conditions in urban freight distribution. In: Ben-Akiva M, Meersman H, van de Voorde E (eds) Freight transport modelling. Emerald Group Publishing Limited, Bingley
6. BESTUFS (2006). http://www.bestufs.net/
7. BESTUFS II, D5.2 quantifications of urban freight transport effects II. http://www.bestufs.net
8. Boccia M, Crainic TG, Sforza A, Sterle C (2011) Location-routing models for designing a two-echelon freight distribution system, CIRRELT-2011-06, Montréal
9. Bodin L, Maniezzo V, Mingozzi A (1999) Street routing and scheduling problems, Chap. 12, In: Hall RW (ed) Handbook of transportation science. Kluwer, Boston
10. Boudoin D, Morel C, Gardat M (2014) Supply chains and urban logistics platforms, In: González-Feliu J, Semet F, Routhier J-L (eds) Sustainable urban logistics: concepts, methods and information systems. Springer, Berlin/New York
11. Campbell AM, Savelsbergh M (2004) Efficient insertion heuristics for vehicle routing and scheduling problems. Transp Sci 38(3):369–378
12. Chen HK, Hsueh CF, Chang MS (2006) The real-time time-dependent vehicle routing problem. Transp Res E 42:383—408
13. CLM (2009) Definition for logistic, council of logistics management. http://cscmp.org/aboutcscmp/definitions.asp.
14. Cordeau JF, Laporte G, Mercier A (2001) A unified tabu search heuristic for vehicle routing problems with time windows. J Oper Res Soci 52:928–936
15. Cordeau JF, Gendreau M, Laporte G, Potvin J-Y, Semet F (2002) A guide to vehicle routing heuristic. J Oper Res Soci 53:512–522
16. Crainic TG, Ricciardi N, Storchi G (2004) Advanced freight transportation systems for congested urban areas. Transp Res C Emerg Tech 12(2):119–137
17. Crainic TG, Ricciardi N, Storchi G (2009) Models for evaluating and planning city logistic transportation systems. Transp Sci 43:432–454
18. Croes GA (1958) A method for solving travelling salesman problems. Oper Res 6:791–812

19. Daskin MS, Owen SH (1999) Location models in transportation, Chap. 10. In: Hall RW (ed) Handbook of transportation science. Kluwer, Boston
20. Drexl M, Schneider M (2013) A survey of location-routing problems. Technical report LM-2013-03, Gutenberg School of Management and Economics, Johannes Gutenberg University, Mainz
21. European Logistics Association (2004) A differentiation for performance: excellence in logistics. Dt. Verkehrs-Verl, Hamburg
22. Fisher M (1995) Vehicle routing, Chapter 1. In: Ball MO et al. (eds) Handbooks in OR and MS, vol 8. North Holland, Amsterdam
23. Fleischmann B, Gietz M, Gnutzmann S (2004) Time-varying travel times in vehicle routing. Transp Sci 38(2):160—173
24. Ghiani G, Laporte G, Musmanno R (2004) Introduction to logistics systems planning and control. Wiley, Chichester/Hoboken
25. Gillett B, Miller L (1974) A heuristic algorithm for the vehicle dispatch problem. Oper Res 22:340–349
26. Glover F (1977) Heuristics for integer programming using surrogate constraints. Dec Sci 8:156–166
27. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13:533–549
28. Goel A (2008) Fleet-telematics: real-time management and planning of commercial vehicle operations. Springer, New York
29. Goodman RW (2005) Whatever you call it, just don't think of last-mile logistics, last Glob Logist Supply Chain Strateg 9(12):46–51
30. Grzybowska H (2012) Combination of vehicle routing models and dynamic traffic simulation for city logistics applications. Doctoral dissertation. Ph.D. dissertation, Universitat Politècnica de Catalunya, Spain
31. Grzybowska H, Barceló J (2012) Decision support system for real-time urban freight management. Procedia-Soci Behav Sci 39:712–725
32. Ichoua S, Gendreau M, Potvin JY (2003) Vehicle dispatching with time-dependent travel times. Eur J Oper Res 144:379–396
33. KUTSUPLUS (2013). http://www.wired.com/2013/10/on-demand-public-transit/
34. Linares MP, Barceló J, Carmona C, Montero L (2016) Analysis and operational challenges of dynamic ride sharing demand responsive transportation models, 2016. In: International symposium of transport simulation (ISTS'16 Conference), transportation research procedia. Available online at www.sciencedirect.com
35. Linares MP, Montero L, Barceló J, Carmona C (2016) A simulation framework for real-time assessment of dynamic ride sharing demand responsive transportation models. In: Roeder TMK, Frazier PI, Szechtman R, Zhou E, Huschka T, Chick SE (eds) Proceedings of the 2016 winter simulation conference, Washington, DC
36. Lourenço HR (2005) Logistics management: an opportunity for metaheuristics. In: Rego C, Alidaee B (eds) Metaheuristics optimization via memory and evolution. Kluwer Academic, Boston, pp 329–356. ISBN:978-1-4020-8134-7
37. Malandraki C, Daskin MS (1992) Time dependent vehicle routing problems: formulations, properties and heuristics algorithms. Transp Sci 26(3):185—200
38. Nagy G, Salhi S (2007) Location-routing: issues, models and methods. Eur J Oper Res 177:649–672
39. Nguyen V-P, Prins C, Prodhon C (2012) Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. Eur J Oper Res 216:113–126
40. Nguyen PK, Crainic TG, Toulouse M (2013) A tabu search for time-dependent multi-zone multi-trip vehicle routing problem with time windows. Eur J Oper Res 231:43–56
41. Orozco J (2011) A microscopic traffic simulation based decision support system for real–time fleet management. Ph.D. disserattion, Universitat Politècnica de Catalunya, Spain
42. PORTAL (2003) Inner urban freight transport and city logistics. http://www.eu-portal.net

43. Potvin J-Y, Xu Y, Benyahia I (2006) Vehicle routing and scheduling with dynamic travel times. Comput Oper Res 33:1129–1137
44. Powell WB, Jaillet P, Odoni A (1995) Stochastic and Dynamic Networks and Routing. Chap. 3. In: Handbooks in operations research and management science, vol 8. Elsevier
45. Psaraftis HN (1988) Dynamic vehicle routing problems. In: Golden B, Assad A (eds) Vehicle routing: methods and studies. North-Holland, Amsterdam
46. Psaraftis HN (1995) Dynamic vehicle routing: status and prospects. Ann Oper Res 61:143–164
47. Regan AC, Mahmassani HS, Jaillet P (1997) Dynamic decision making for commercial fleet operations using real-time information. Transp Res Rec 1537:91–97
48. Regan AC, Mahmassani HS, Jaillet P (1998) Evaluation of dynamic fleet management systems: simulation framework. Transp Res Rec 1645:176–184
49. Taillard É (1991) Robust taboo search for the quadratic assignment problem. Paral Coput 17:443–455
50. Tang H (2008) Efficient implementation of improvement procedures for vehicle routing with time-dependent travel times. Transp Res Rec 2089:66–75
51. Taniguchi E, Thompson RG, Yamada T, Van Duin R (2001) City logistics: network modeling and intelligent transport systems. Pergamon, Amsterdam
52. Thomas B (2007) Waiting strategies for anticipating service requests from known customer locations. Transp Sci 41(3):319–331
53. Thomas B, White C III (2004) Anticipatory route selection. Transp Sci 38(4):473–487
54. Toth P, Vigo D (eds) (2014) Vehicle routing: problems, methods, and applications. MOS-SIAM series on optimization, 2nd edn. ISBN:978-1-611973-58-7
55. Tragantalerngsak S, Holt J, Rönnqvist M (1997) Lagrangian heuristics for the two-echelon, single-source, capacitated facility location problema. Eur J Oper Res 102:611–625
56. UBER (2015). http://www.uber.com
57. Wren A (1971) Computers in transport planning and operation. Ian Allan, London
58. Wren A, Holliday A (1972) Computer scheduling of vehicles from one or more depots to a number of delivery points. Oper Res Q 23:333–344

Cutting and Packing

31

Ramón Alvarez-Valdes, Maria Antónia Carravilla, and José Fernando Oliveira

Contents

Introduction . . . 932
Cutting and Packing Problems . . . 933
Dimensionality . . . 933
Geometry . . . 935
Problem Types . . . 937
Additional Characteristics and Constraints . . . 939
Rectangular Problems (2D and 3D) . . . 942
Constructive Algorithms . . . 942
Local Search Procedures . . . 946
Metaheuristic Algorithms . . . 948
Model-Based Algorithms . . . 956
Irregular Problems . . . 957
Constructive Heuristics . . . 957
Local Search Heuristics . . . 963
Mathematical Programming-Based Heuristics . . . 966
Metaheuristics . . . 970
Conclusion . . . 973
Cross-References . . . 973
References . . . 974

Abstract

Cutting and Packing (C&P) problems arise in many industrial and logistics applications, whenever a set of small items, with different shapes, has to be

R. Alvarez-Valdes (✉)
Universitat de València, Burjassot, Spain
e-mail: ramon.alvarez@uv.es

M. A. Carravilla · J. F. Oliveira
INESC TEC and Faculty of Engineering, University of Porto, Porto, Portugal
e-mail: mac@fe.up.pt; jfo@fe.up.pt

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_43

assigned to large objects with specific shapes so as to optimize some objective function. Besides some characteristics common to combinatorial optimization problems, the distinctive feature of this field is the existence of a geometric subproblem, to ensure that the items do not overlap and are completely contained in the large objects. The geometric tools required to deal with this subproblem depend on the shapes (rectangles, circles, irregular) and on the specific conditions of the problem being solved. In this chapter, after an introduction that describes and classifies Cutting and Packing problems, we review the basic strategies that have appeared in the literature for designing constructive algorithms, local search procedures, and metaheuristics for problems with regular and irregular shapes.

Keywords
Cutting stock · Bin packing · Strip packing · Container loading · Nesting

Introduction

Cutting and Packing (C&P) problems are hard combinatorial optimization problems (almost all are NP-hard) that arise in the context of several real-world applications, both in industry and in services, whenever one or more large object or container space has to be divided into smaller items, so that the waste is minimized. These problems include cutting paper rolls into narrower rolls in the paper industry, cutting large boards of wood into smaller rectangular panels in the furniture industry, or cutting the irregularly shaped components of garment from fabric rolls, but they include also packing boxes into containers or loading items on pallets, in logistics applications. All the problems have in common the existence of a geometric subproblem, originated by the natural item nonoverlapping constraints.

In the literature, these problems have received an increasing amount of attention. In the 1960s, the seminal papers were published and the number of publications has grown exponentially since then. As combinatorial optimization problems, C&P problems may be solved using all the optimization approaches and techniques available nowadays: linear programming techniques, branch and bound search algorithms, constraint logic programming, dedicated heuristics, metaheuristics, matheuristics, etc. Unfortunately, due to their combinatorial nature, exact techniques are not capable to efficiently tackle these problems for large instances; therefore, heuristic approaches must be used.

The remaining of this chapter is organized as follows. In the next section, C&P problems will be formally defined and categorized, according to their objective and to their quantitative and geometric parameters. Then, in the following two sections, heuristic approaches for rectangular problems and irregular problems, both two- and three-dimensional, are presented, ranging from simple constructive heuristics to mathematical programming model-based heuristics and including metaheuristics.

Table 1 Example of a two-dimensional rectangular cutting problem

Item type	Quantity	Length (meters)	Width (meters)
1	2	2.00	0.50
2	2	1.80	1.00
3	3	1.90	0.65
4	2	2.10	0.55
5	6	0.65	0.23

Cutting and Packing Problems

A carpenter wants to build a wardrobe for which he/she needs to cut a set of rectangular items in the quantities and with the dimensions presented in Table 1. To cut these items, the carpenter has at his/hers workshop two large wood panels of 3 by 2 m and three other large wood panels of 2.5 by 2.5 m. How should the carpenter cut the small items from the available large panels so that the waste (the trim of the cutting process) is minimized?

This toy example allows us to introduce the main concepts of Cutting and Packing problems. In general, in Cutting and Packing problems, "small" items have to be assigned to "large" objects under both geometric constraints (assuring that the items do not overlap and are contained inside the objects) and quantitative constraints (e.g., at least a given quantity of each item type has to be cut). The objective of the problem may be either related to the minimization of the value of the large objects that are used or to the maximization of the value of the small items that are cut. The solution of the problem is one or more cutting patterns, describing the geometric disposition of the small items in/on the large objects (Fig. 1). It should be emphasized that, in general, to describe the solution of a Cutting and Packing problem is not enough to say which small items are cut from which large objects (the assignment part of the problem). Part of the solution is describing how the small items are arranged on the large objects (the geometric problem).

These problems can be categorized according to a typology proposed by Wäscher et al. [82]. This typology takes into account the problem objective, the number of objects and items, and the geometry of the items. The different types of Cutting and Packing problems will be described by closely following Wäscher et al. typology.

Dimensionality

Cutting and Packing problems are usually classified according to their dimensionality as one-, two-, and three-dimensional problems. This classification is directly related to the number of physical dimensions relevant for the problem. If it is true that our world and all the objects in it are three-dimensional, when planning how to cut narrower paper rolls from a wider roll, only one dimension is relevant as the other two are kept equal in small items and large objects, i.e., the problem is one-dimensional (1D). Applying the same reasoning to the initial carpenter problem,

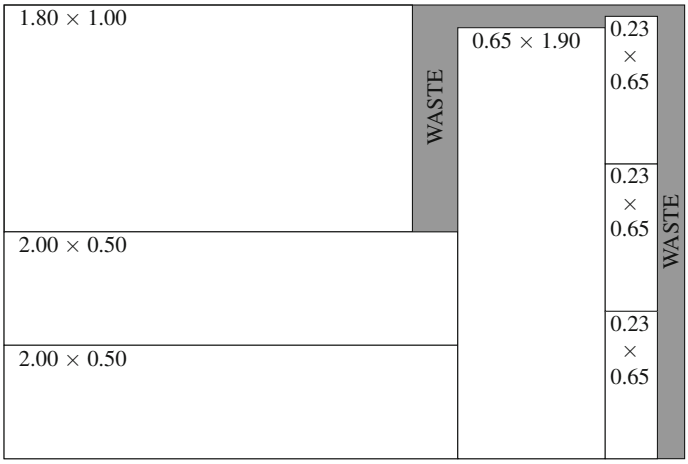


Fig. 1 A two-dimensional cutting pattern example

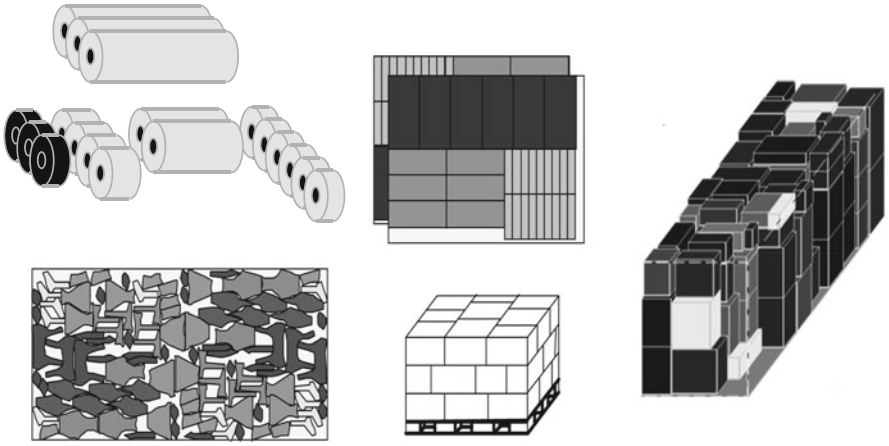


Fig. 2 Examples of one-, two-, and three-dimensional Cutting and Packing problems

it is the length and the width of the large objects that must be cut in order to match the length and the width of the small items, not changing the thickness of the wood boards; this is therefore a two-dimensional problem (2D). When packing parallelepiped boxes inside a truck or a container, the three dimensions are relevant for the problem, and it is therefore a three-dimensional problem (3D). Although four- and even higher-dimensional problems are theoretically possible, no relevant practical applications have been described until now, and therefore they will be disregarded in this text. In Fig. 2, examples of 1D, 2D and 3D problems arising in the paper, garment, furniture, and logistics industries are shown.

Geometry

The second most distinctive characteristic of the Cutting and Packing problems is the geometry associated with the dimensions that are relevant for the problem. In 1D problems, the geometric problem is trivial as, once the assignment problem is solved and it is known which small items are cut from each large object, the cutting order is not relevant as any sequence is feasible. Two- and three-dimensional problems are classified as rectangular, circular, or irregular problems. This division is related with the type of geometric tools needed to handle the geometric constraints of the problem. Geometric feasibility is guaranteed if the small items do not overlap each other and are completely contained inside the large object. For the sake of simplicity, in the following, only two dimensions will be considered, but the same reasoning and concepts hold for three-dimensional problems.

When dealing with rectangular shapes, analyzing if they do not overlap each other is just a question of coordinate comparison. Let rectangle i, with dimensions (l_i, w_i), have its left lower corner placed on coordinates (x_i, y_i) of the Cartesian plane, and let rectangle j, with dimensions (l_j, w_j), have its left lower corner placed on coordinates (x_j, y_j) of the Cartesian plane. Rectangle j does not overlap rectangle i if it is either above, below, or to the left or right of rectangle j, i.e.:

$$x_j + l_j \leq x_i \quad \vee \quad x_j \geq x_i + l_i \quad \vee \quad y_j \geq y_i + w_i \quad \vee \quad y_j + w_j \leq y_i$$

The containment conditions of rectangle i inside the larger rectangle (L, W) (assuming that its left lower corner is placed on coordinates $(0, 0)$) are:

$$x_i \geq 0 \quad \wedge \quad x_i + l_i \leq L \quad \wedge \quad y_i \geq 0 \quad \wedge \quad y_i + w_i \leq W$$

In circular Cutting and Packing problems, small items have circular shapes and the large object can be either another (larger) circle or a rectangle. The condition for the items not to overlap is now based on the computation of the distance between the centers of the circles. Let circle i, with radius r_i, have its center placed on coordinates (x_i, y_i) of the Cartesian plane, and let circle j, with radius r_j, have its center placed on coordinates (x_j, y_j) of the Cartesian plane. Circle j does not overlap circle i if the distance between the two centers is greater than or equal to the sum of their radii, i.e.:

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq r_i + r_j$$

It should be noticed that it is not necessary to compute the computationally expensive square root, and this condition can be replaced by the equivalent one having both sides squared.

The containment condition of circle i, with its center placed on coordinates (x_i, y_i), inside the larger circle of radius R and center on coordinates $(0, 0)$ is:

$$\sqrt{x_i^2 + y_i^2} \leq R - r_i$$

If the container is a rectangle (L, W), with its bottom-left corner on coordinates $(0, 0)$, the containment conditions are:

$$x_i - r_i \geq 0 \quad \wedge \quad x_i + r_i \leq L \quad \wedge \quad y_i - r_i \geq 0 \quad \wedge \quad y_i + r_i \leq W$$

All shapes other than rectangles and circles are considered irregular and require more complex geometric algorithms to check and enforce geometric feasibility. A complete and thorough tutorial on the geometry of irregular Cutting and Packing problems (aka as nesting problems) can be found in Bennell and Oliveira [17]. As the no-fit polygon is the most widely used geometric tool to analyze if two irregular shapes overlap, it will be described for the simpler case of convex polygons.

Let P_i be a polygon with n_i vertices with coordinates $(a_i^k, b_i^k), k = 1, \ldots, n_i$ relative to an arbitrary reference point. Let P_j be another polygon with n_j vertices with coordinates $(a_j^m, b_j^m), m = 1, \ldots, n_j$ also relative to an arbitrary reference point. Consider that the reference point of polygon P_i is placed on coordinates (x_i, y_i) and that the reference point of polygon P_j is placed on coordinates (x_j, y_j). The direct comparison of the relative positions of two polygons is a complex and computationally heavy task, mainly in the general case when the polygons are non-convex. An alternative is to resort to the concept of no-fit polygon, a new polygon that captures the geometric information of both polygons. In fact the edges of the no-fit polygon are the edges of the two polygons taken in a special order. With the no-fit polygon, checking the relative position of two polygons is reduced to checking the relative position of one point against the no-fit polygon.

To build the no-fit polygon, consider polygon P_i fixed and polygon P_j sliding around P_i, always in contact but never intersecting P_i. The trace of the reference point of P_j during this movement is the no-fit polygon of P_j in relation to P_i: $NFP_{P_i P_j}$ (Fig. 3). The reference point of $NFP_{P_i P_j}$ is the same as the reference point of P_i. In order to guarantee that all the small items do not overlap, it is necessary to build the no-fit polygons for all pairs of items. They can however be built in a preprocessing phase, as they do not depend on the absolute coordinates of the reference point of P_i.

Let $NFP_{P_i P_j}$ be the no-fit polygon of polygon P_j in relation to polygon P_i, with N_{ij} vertices with coordinates $(a_{ij}^k, b_{ij}^k), k = 1, \ldots, N_{ij}$ and with its reference point placed on coordinates (x_i, y_i). Polygon P_j will not intersect polygon P_i if its reference point is on coordinates (x_j, y_j) such that:

$$\exists k \in \{1, \ldots, N_{ij} - 1\} : y_j \geq \frac{b_{ij}^{k+1} - b_{ij}^k}{a_{ij}^{k+1} - a_{ij}^k}(x_j - a_{ij}^k) + b_{ij}^k$$

i.e., the comparison of the coordinates of one point against one polygon.

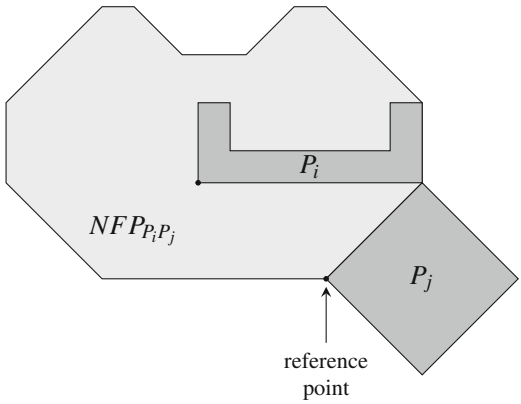


Fig. 3 The no-fit polygon of polygon P_j in relation to polygon P_i

Note that the direction of the inequality in the above expressions ("$\geq$" or "$\leq$") depends on the orientation chosen for the polygons, i.e., if vertex $k+1$ is on the left-hand side or on the right-hand side of the line defined by the vertices $k-1$ and k.

Problem Types

From the nongeometric point of view, Cutting and Packing problems can be formally defined as follows:

> A set of small items of m distinct types, with distinct geometries and values v_i, $i = 1, \ldots, m$, have to be assigned in a minimum amount of d_i^L and a maximum amount of d_i^U units, to a set of n distinct types of large objects, with distinct geometries and values V_j, $j = 1, \ldots, n$ available in a limited amount of D_j units.

The objective of the problem may be either *output maximization* or *input minimization*. If there are more small items to cut/pack (In what follows, we will use the term *pack* to refer indistinctly to Cutting and Packing processes.) than space available in the large objects, the goal is to select the subset of small items to pack so that the value extracted from the large objects (the value of the small items effectively packed) is maximized. Therefore, these problems are of *output maximization*, where the *output*, the value v_i of the small items, may be proportional or not to their area/volume. In this situation, as there are more small items to pack than large objects available, the value V_j of the large objects is irrelevant as all the large objects will be used.

If there are more large objects available than the ones needed to pack all the small items, the goal is to select the subset of large objects to use so that the value of the used large objects is minimized. Therefore, these problems are of *input minimization*, where the *input*, the value V_j of the large objects, may be proportional or not to their area/volume. In this situation, as there are more large objects available than the ones needed to pack all the small items, the value v_i of the small items is irrelevant as all small items will be packed.

Depending on the values of m, d_i^L, d_i^U, n, and D_j and on the objective of the problem, we will get the six main Cutting and Packing problem types, according to Wäscher et al.'s typology:

- Identical Item Packing Problem (IIPP)
- Placement Problem (PP)
- Knapsack Problem (KP)
- Cutting Stock Problem (CSP)
- Bin Packing Problem (BPP)
- Open Dimension Problem (ODP)

The first three are *output maximization* problems and the last three are *input minimization* problems.

In the IIPP, the small items are of only one type ($m = 1$), i.e., all the items are geometrically equal, but there is not a maximum demand for the item ($d_i^U = \infty$), i.e., that item should be packed as many times as possible. In the PP, m types of items have to be packed, but there is an upper limit on the number of items of each type to be packed (d_i^U). In the KP, all the small items are geometrically different, and therefore the number of items is equal to the number of types of items (m), and the upper limit on the number of items to pack of each type is one ($d_i^U = 1$).

In what concerns the *input minimization* problems, while in the CSP there are few types of items (m), with a demand d_i^L greater than or equal to one, in the BPP, all the small items are different and have a demand of one ($d_i^L = 1$). In both cases, the large objects may be all different, i.e., there is only one unit of each large object ($D_j = 1$), the large objects may be of several types n with an availability D_j for each type, or they may be of just one type ($n = 1$) with an infinite availability ($D_j = \infty$). In the ODP, there is an "infinite" availability of the large object due to the open (not limited) dimension of the large object. An example is the rectangular strip packing problem, in which small rectangles have to be packed on a rectangle of "infinite" length, so that the length used is minimized. This happens in the textile industry where items are cut from fabric rolls. In practical applications of ODP, the m types of small items have a demand d_i. The characterization of the six types of problems, based on the values of their parameters, is resumed in Table 2. The symbol "—" stands for *"not relevant for the problem type characterization"*.

In *output maximization* problems, in which there is an upper bound on the number of small items of each type to pack (d_i^U), there may also exist a lower bound d_i^L, meaning that it is mandatory to pack at least d_i^L small items of type i.

The six types of problems are represented in Figs. 4 and 5 for the two-dimensional rectangular Cutting and Packing problem. The small items and large objects effectively cut/used in a hypothetical solution are represented in dark gray to highlight the difference between *output maximization* problems where all large objects are used and *input minimization* problems where all small items are packed.

Table 2 Characterization of the six types of problems based on the values of their parameters

Problem	Objective	m	d_i^L	d_i^U	n	D_j
IIPP	Output maximization	1	—	∞	—	—
PP	Output maximization	≥ 1	—	≥ 1	—	—
KP	Output maximization	≥ 1	—	1	—	—
CSP	Input minimization	≥ 1	≥ 1	—	1 ≥ 1	∞ 1
BPP	Input minimization	≥ 1	1	—	1 ≥ 1	∞ 1
ODP	Input minimization	≥ 1	≥ 1	—	1	1

Additional Characteristics and Constraints

Many additional constraints to the cutting/packing patterns arise from the practical applications of these problems. These additional constraints are specific for each dimensionality or kind of geometry; however, two characteristics emerge as common to almost all two- and three-dimensional Cutting and Packing problems: small item orientation and guillotine patterns.

Given the physical characteristics of the raw materials from which the large objects are made, sometimes it is not possible to rotate small items, i.e., they have to be packed with the dimensions and orientation given: length strictly along the length and width strictly along the width. It is the case of wood boards or fabric rolls, for instance. In other cases, there is no problem in rotating the small items when placing them on the large objects, as it happens in the steel industry. Even when small item rotations are allowed, it is usual to differentiate the cases where only 90° and 180° rotations are allowed, which lead to orthogonal patterns, from other situations when other rotation angles are also admissible, in what is usually designated as free rotation. In three-dimensional packing problems, as container loading problems, there may be "this-side-up" constraints that restrict the number of feasible orientations in which a box may be loaded in the truck or container.

When dealing with rectangles, guillotine constraints may be imposed to the cutting patterns. A cutting pattern is said to be *guillotinable* if all the small items can be extracted with top-down (right-left) cuts made on the large object. Each cut produces two rectangles that may be the demanded small items or may be bigger rectangles to cut according to the same rule. Guillotine cutting patterns can be classified according to the number of times that the cutting tool has to be turned 90°. If a sequence of horizontal (vertical) cuts, generating strips, followed by a sequence of vertical (horizontal) cuts on each strip completely executes the cutting pattern, then this pattern is called a two-stage pattern. If an additional horizontal (vertical) cut is needed to cut the small items, then the pattern is classified as a three-stage pattern. The pattern is said to be n-stage if the cutting tool must be turned n times

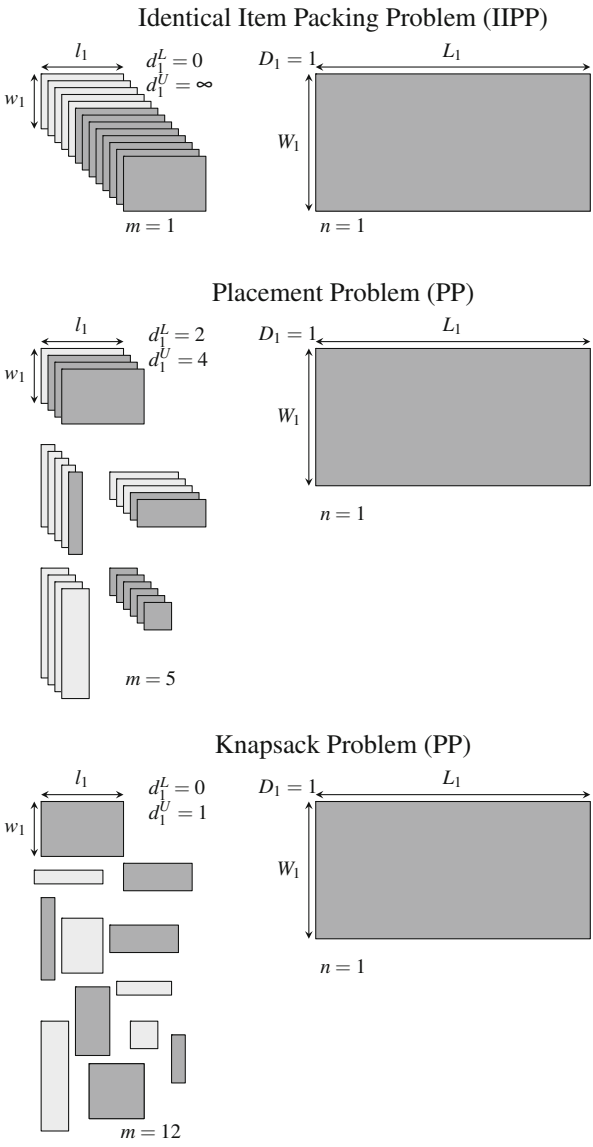


Fig. 4 Illustration of the three types of *output maximization* Cutting and Packing problems for the two-dimensional rectangular form of the problem

to complete the pattern. Figure 6 presents two- three-, and n-stage guillotine cutting patterns and a non-guillotine cutting pattern. These concepts can be extended to three dimensions, when dealing with the cut of parallelepipeds.

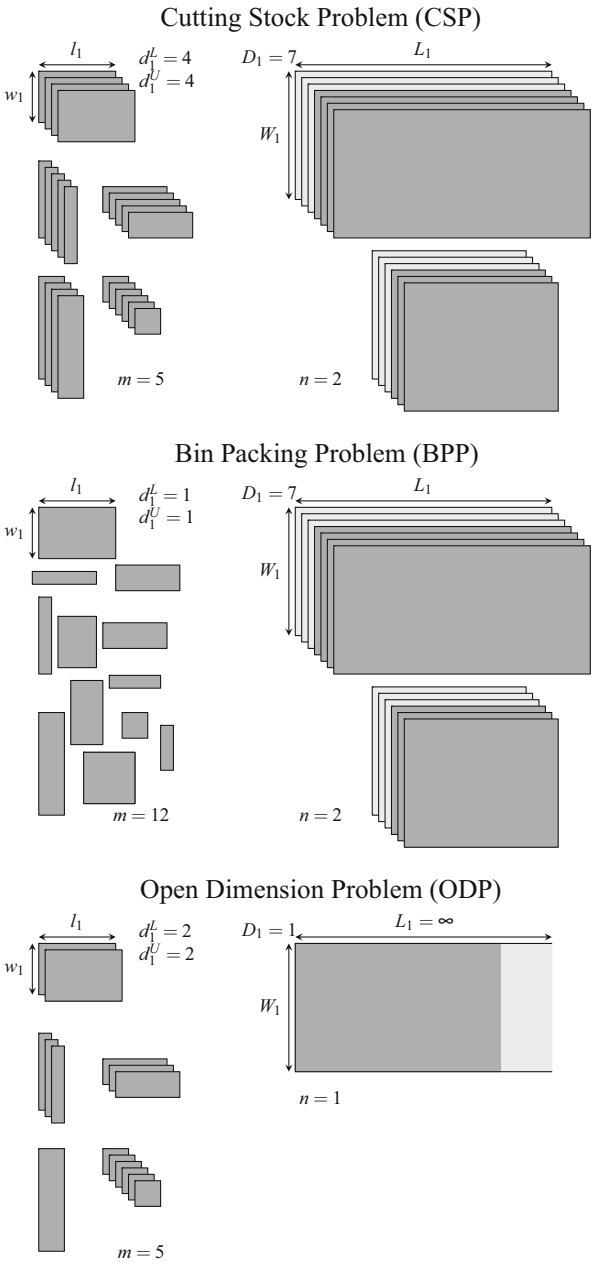


Fig. 5 Illustration of the three types of *input minimization* Cutting and Packing problems for the two-dimensional rectangular form of the problem

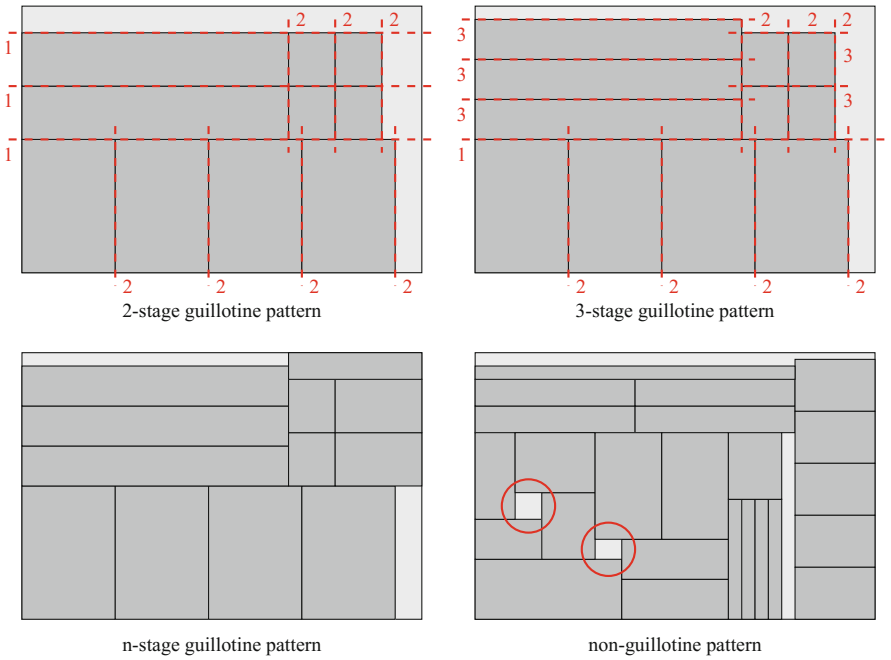


Fig. 6 Two-, three-, and n-stage guillotine cutting patterns and a non-guillotine cutting pattern

Rectangular Problems (2D and 3D)

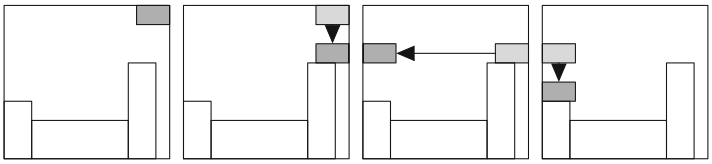
This section describes the most widely used techniques for developing heuristic algorithms for Cutting and Packing problems involving rectangular items in two and three dimensions. First, constructive procedures will be described, then local search methods, and finally metaheuristic algorithms successfully used for the basic versions of the problems. The section will conclude with some references to model-based heuristics.

Constructive Algorithms

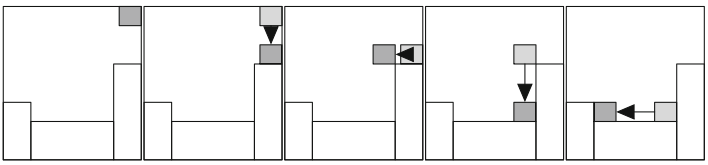
Most of the heuristic algorithms include some constructive procedure in which a feasible solution is iteratively built by adding at each iteration a subset of items to the existing partial solution. All of these procedures involve two basic steps: selection of the items to be packed and selection of the space in which to pack them. Depending on the order in which these two steps are called, two types of algorithms can be distinguished:

1. *Item-driven algorithms*

   In this type of algorithms, the item to be packed is chosen first, and then the best location for packing it is selected. The best-known example of this type is the



(a) Jakob's implementation.



(b) Liu and Teng's implementation.

Fig. 7 Bottom-left algorithm. (a) Jakob's implementation. (b) Liu and Teng's implementation

bottom-left (*BL*) algorithm, proposed by Baker et al. [14] and then used by many other authors. In the *BL* algorithm, an ordered list of items is given, and at each step, the first item in the list is taken and placed to the lowest and leftmost position in the empty space. In Jakobs's implementation [50], the position is obtained by putting the item at the upper-right corner of the space and then shifting it alternately as far as possible to the bottom and then as far as possible to the left, as in Fig. 7a. Liu and Teng [58] developed an improved implementation giving priority to the downward movement, so the item is only shifted to the left when it cannot be shifted downward, as in Fig. 7b. They showed that, unlike Jakobs's method, there is always an ordering of the items that can produce the optimal solution.

A refinement of the *BL* algorithm is the *bottom-left-fill* (*BLF*) algorithm proposed by Chazelle [25], in which the empty spaces between items already packed are also considered to accommodate the new item. The algorithm has to maintain a list of points in a bottom-left order to indicate where the new item can be placed. These points are checked in order until a feasible position is found. Figure 8 compares *BLF* with *BL* on an example. A new item has to be added to the partial solution and is initially placed at the upper-right corner (Fig. 8a). In Fig. 8b, the *BLF* algorithm is used. The figure shows the candidate points and the position selected for the new item. Using the *BL* algorithm, the solution in Fig. 8c is obtained. Hopper and Turton [47] showed that the *BLF* heuristic outperforms the *BL* routine by up to 25% with the performance gain being higher for larger problems. However, its complexity is $O(n^3)$, while the complexity of *BL* is just $O(n^2)$.

2. *Space-driven algorithms*

In this type of algorithms, a space is selected first from the list of available spaces, and then the item or the subset of items which fits best into the space is chosen to be packed.

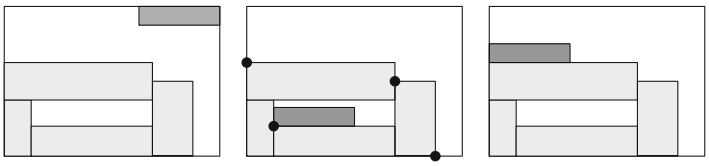


Fig. 8 Comparing bottom-left-fill and bottom-left algorithms. (a) Placing a new item. (b) Placed with BLF. (c) Placed with BL

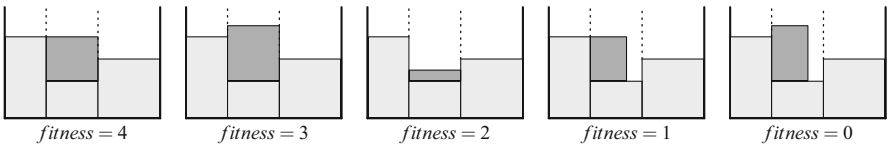


Fig. 9 Scoring rule of Leung et al. [53]

An example of this type of procedure is the *best-fill* (*BF*) algorithm proposed by Burke et al. [21] for the strip packing problem in which, at each step, the lowest space is chosen and then the item which fits best is chosen to occupy it.

A further refinement of the *BF* algorithm is the *scoring* rule by Leung et al. [55], in which the quality of the fitting is evaluated using a fitness measure that takes into account other factors besides the way in which the base of the item fills the base of the space. Figure 9 shows the empty space selected and the fitness values corresponding to candidate items.

Apart from these simple cases, the algorithms in which the space is chosen first are usually more complex, because they have to decide how to manage the empty spaces and because they usually choose subsets of items to fill the spaces instead of just one item.

Concerning the management of the spaces, there are basically two alternatives: partition the empty space into smaller disjoint spaces or consider non-disjoint maximal spaces [51].

Partitions are very useful when guillotine cuts are considered or in three-dimensional problems when the items have to be fully supported from below by other items or the base of the large object. They are also easier to manage, because every time an item is placed in a space, it produces some new spaces to substitute the previous one, without modifying any other spaces in the list.

Conversely, maximal spaces are more flexible and can produce solutions that partitions are not able to attain, but they are more difficult to manage. As maximal spaces are not disjoint, putting an item into one space may modify other spaces, and the whole list of spaces has to be checked. An example of maximal spaces appears in Fig. 10. Initially the parallelepiped is empty, and when an item is packed at its bottom-left corner, three maximal spaces are generated. Figure 11

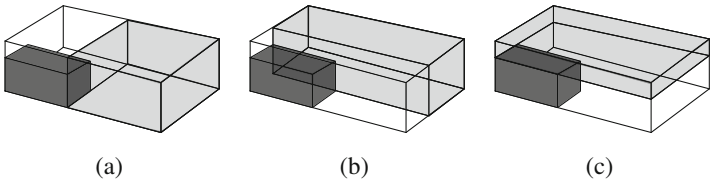


Fig. 10 Space management in 3D: maximal spaces

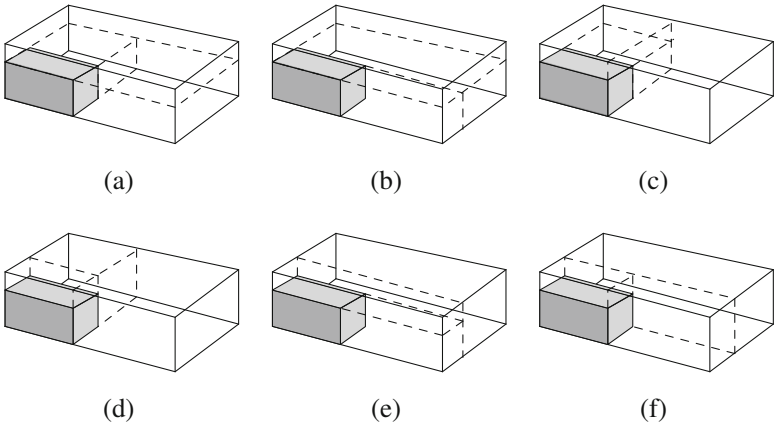
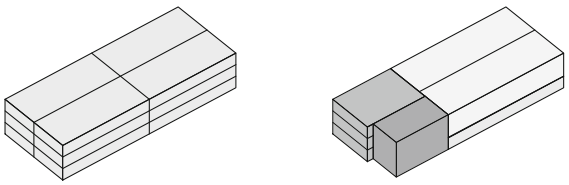


Fig. 11 Space management in 3D: partitions

Fig. 12 (a) Homogeneous and (b) heterogeneous blocks



shows an example of partition. In this case, when putting the item at the bottom-left corner of the parallelepiped, a decision has to be taken about the new spaces generated. Six possibilities appear, depending on the horizontal or vertical cuts.

The other aspect that the algorithms have to decide is if, at each step, just one item is selected to be packed or if a subset of items with a certain structure is selected. In this second case, the items come in *blocks*, that is, arranged in a structure of rows and columns. Blocks have been used by many authors and can be considered the standard way of packing items once the space has been chosen. In most cases, the blocks are *homogeneous*, that is, composed of just one type of item, as in Fig. 12a, but sometimes, in more recent studies (Fanslau and Bortfeldt [36], Zhu et al. [86], Wei and Lim [83]), they are *heterogeneous*,

including items of different types as long as their dimensions do not differ more than a given threshold, as in Fig. 12b.

Finally, there are algorithms that select simultaneously both elements, the item or block to be packed and the space in which to pack it. An example is the procedure proposed by Wu et al. [85] for packing rectangles into a rectangle, in which, following the principle that "Golden are the corners; silvery are the sides; and strawy are the voids," all the unpacked items are evaluated to be placed at each corner of the remaining empty space. More recently, the placement algorithm proposed by Wei et al. [84] considers all pairs (p, r) (position-rectangle), selects one of them according to some fitness criteria, and places the rectangle in this position.

Local Search Procedures

There are several ways to define and explore the neighborhood of a solution. Two main groups of local search procedures can be distinguished: those in which the solution is given by an ordered list of items, which will be decoded into a physical solution by means of an algorithm such as BL or BLF, and those in which the layout of the solution is used to define the neighborhood.

1. *Procedures based on ordered lists of items*

   In Hopper and Turton [47], the problem is represented by a permutation of items, giving the order in which the items are packed. They define two manipulation operations. The first one swaps two randomly selected items in the permutation. The second operator flips the orientation of one randomly selected item. In the translation to the next solution, only one of the operators is applied with a 50% chance. The initial solution is randomly generated.

   Leung et al. [56] apply the scoring algorithm [55] to an ordered list of items. If two items have the same score, the first item on the list is packed first. Therefore, different item sequences can generate different solutions. They exhaustively study the sequences produced by exchanging the positions of every pair of items and keep the best solution found.

   Ceschia and Schaerf [24] deal with a multi-container problem and work with a set of sequences, one for each container. Each element of a sequence is a homogeneous block. A move consists of moving some items of a block to another position of the same or another sequence, keeping or changing their orientation. An example appears in Fig. 13 [24], involving two containers. Part of the block in position 4 in the first sequence is moved to position 4 in the second sequence.
2. *Procedures based on the layout of the solutions*

   The movements determining the neighborhood of a solution can be defined directly on the layout of the solution, taking into account the position of the items on the large objects. Some examples of these moves can be seen in the following figures. Figure 14 [8] shows a simple move in which an item is selected to change its position on the strip. Once it is selected, it is removed and packed again on an empty space in which the item can fit or not. If it does not fit, as in the example

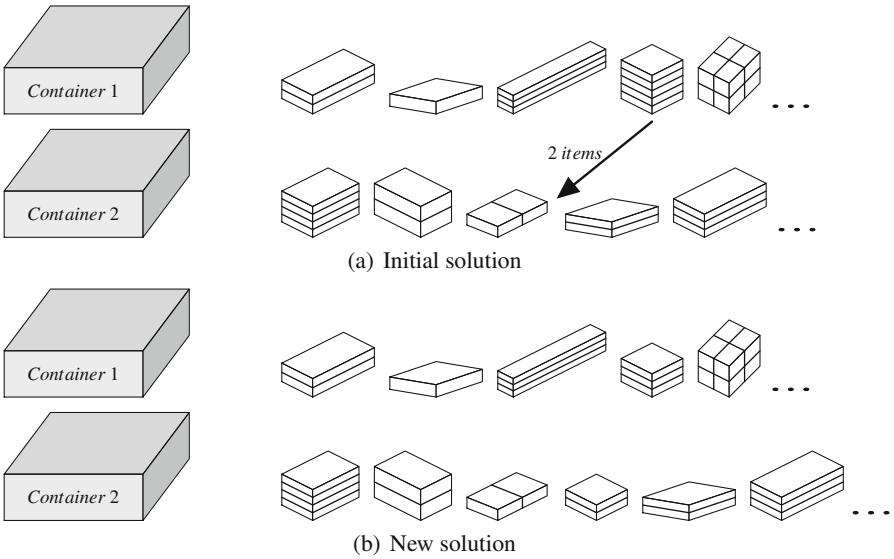


Fig. 13 Ceschia and Schaerf's movement. (a) Initial solution. (b) New solution

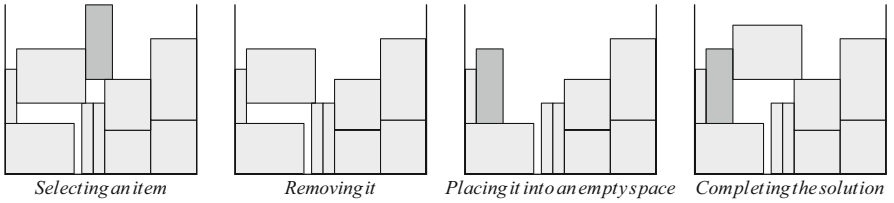


Fig. 14 Changing the position of an item

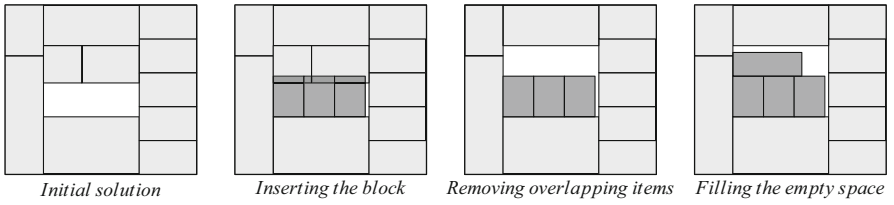


Fig. 15 Inserting a block

of the figure, the overlapping items are removed and packed again by using a constructive procedure.

In Fig. 15 [68], a block of items which were left initially unpacked are inserted into a bin. The overlapping items are removed and the new empty spaces are filled again.

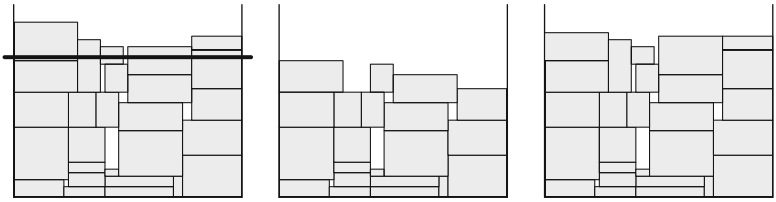


Fig. 16 Removing and filling in a two-dimensional problem

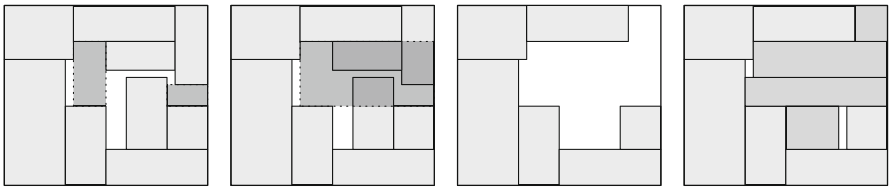


Fig. 17 Merging empty spaces in two- and three-dimensional problems

A frequently used type of movement consists of removing part of the solution and rebuilding the partial solution in a way that produces a different solution. A two-dimensional example appears in Fig. 16 [8], in which the last part of the solution of a strip packing problem is removed and the strip filled again, using a different procedure. Finally, Fig. 17 [68] shows a more complex move, which can be applied to two- and three-dimensional problems. Two empty spaces are selected, and then the smallest rectangle (or parallelepiped in 3D) containing them is determined, and the items contained totally or partially in it are removed. The resulting space is filled again, producing a new solution.

Metaheuristic Algorithms

Iterative Algorithms

Lesh et al. [53] develop the *bottom-left-descending (BLD)* algorithm, in which the known BL algorithm is used four times, ordering the items by decreasing the height, width, area, and perimeter. In order to improve BLD, they develop a stochastic variation, algorithm BLD^*. BLD^* starts with a fixed order (say decreasing height) and generates random permutations from this order. Items are selected in order, one at a time. For each selection, BLD^* goes down the list of remaining items, accepting each item with probability p, until an item is accepted. If the last item is reached and not selected, it restarts at the beginning of the list. After an item is accepted, the process starts again from the beginning of the list. The probability of obtaining some ordering y, starting from some fixed ordering x, is proportional to $(1-p)^{\text{Ken}(x,y)}$, where $\text{Ken}(x, y)$ is the Kendall tau distance between the two permutations.

Belov et al. [15] propose two iterative heuristics for the strip packing problem. The first one uses a special single-pass heuristic *SubKP* in each iteration. *SubKP* fills the selected empty space by solving a 1D knapsack problem, considering only item widths. At each iteration, the profits of the items change according to a sequential value correction method [65]. The second iterative heuristic, *BS(BLR)*, uses at each iteration a single-pass heuristic *bottom-left-right (BLR*), a modification of *BL*, in which the item can be assigned to the left or to the right in the most bottom-left free space large enough for that item. To construct item orderings for *BLR*, a simplified *BubblePermute* [52] procedure is used.

Araujo and Armentano [11] also propose a multi-start random constructive algorithm. At each step of each constructive process, a list of homogeneous blocks are evaluated and ranked. A biased random selection determines the block to pack, but, before packing it, another biased random procedure reduces the rows and/or columns of the block. These two randomizing strategies guarantee the diversity of the solutions obtained.

Genetic Algorithms

Most of the genetic algorithms proposed for solving Cutting and Packing problems use a representation in which each solution is given by an ordered list of items and a decoding algorithm translates this list into a physical solution.

Jakobs [50] already used this representation in his genetic algorithm, with a specific crossover operator and a mutation operator that exchanges the position of some elements of the permutation. The solutions are decoded by using the *BL* algorithm.

Hopper and Turton [47] use this representation, with partially matched crossover (PMX) and order-based mutation. They also consider elitism and seed the initial population with a solution obtained by a heuristic placement procedure. As decoding algorithm, they use *BL* and *BLF*.

Leung et al. [54] also use as chromosomes ordered list of items, a mutation operator consisting in swapping the positions of two items randomly chosen, and two-point and cycle crossover operators. They also include a simulated annealing operator to induce competition between parents and children. If the children are better, they are accepted, but if the children are not as good as their parents, they can still be accepted with some probability. This helps to prevent the population from becoming homogeneous too early. The solutions are decoded using a procedure similar to the *BLF*.

Iori et al. [49] also use the ordered list representation. The main difference with respect to standard procedures is that they distinguish between improving and non-improving phases, depending on the number of individuals generated without improving the best solution obtained. The phase influences the fitness evaluation, the elitism strategy, and the use of mutation. Another special feature is the use of local search to improve the individual with the largest fitness.

A different approach to genetic algorithms based on a list of items is the random-key genetic algorithm proposed by Gonçalves and Resende [44] for the bin packing problem. If the instance being solved has n items, the chromosome has $2n$ genes, the

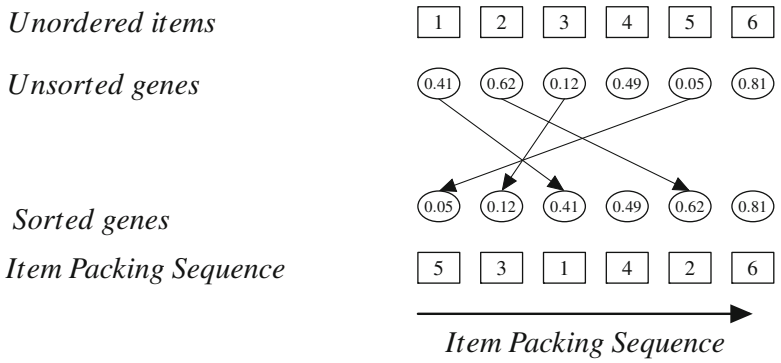


Fig. 18 Decoding of the item sequence in the algorithm of Gonçalves and Resende

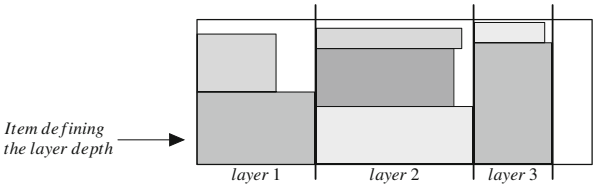


Fig. 19 Layer structure of the Bortfeldt and Gehring's solutions

first n for the sequence of items and the last n for their orientations. Each individual of the population is a vector of $2n$ random keys. Initially, the values are generated at random in the real interval $[0, 1]$. At each iteration, some mutants are added, generated in the same way that the elements of the initial population are generated. Also, a small group of elite individuals are taken from the previous generation. The remaining members are obtained by parameterized uniform crossover, combining elite and non-elite individuals. For each component of the vector, the elite individual component is taken with a given probability. Once the individuals are formed, they are decoded, first ordering the items according to the values of their random keys, to produce an item sequence, and then placing the items into the bins using some placement algorithms derived from *BLF*. Figure 18 shows how the part of the chromosome corresponding to the items is sorted to produce an item sequence.

A completely different structure for a genetic algorithm is proposed by Bortfeldt and Gehring [20] for the container loading problem. Unlike all previous algorithms, they use the layer structure of the solution. Each feasible solution is composed of a set of layers, whose height and width are the same as the container and whose depth is defined by one of their items (see Fig. 19). The constructive algorithm builds a solution with this layer structure, and the genetic operators build new solutions by combining layers of previous solutions and including new layers for completing solutions when necessary.

Simulated Annealing Algorithms

Hopper and Turton [47] propose a simulated annealing algorithm in which the neighborhood is defined by the set of solution that can be reached applying two manipulation operations. The first one swaps two randomly selected items in the permutation. The second operator flips the orientation of one randomly selected item.

In Lai and Chan [51], the neighborhood is obtained by swaps of randomly selected pairs of items and the solutions are decoded using an algorithm similar to *BLF*. In Leung et al. [55,56], simulated annealing is used jointly with a constructive and a local search algorithm to escape from local minima. Neighbors are produced again by swapping pairs of items and the constructive algorithms work as decoders. Ceschia and Schaerf [24] also enhance their local search procedure by adding a simulated annealing phase to escape from local optima.

Burke et al. [22] propose a hybrid procedure in which a solution is generated by a two-step process that, first of all, applies the best-fill heuristic to place an initial number of items onto the large object. Then, the remaining unassigned items are placed using the bottom-left-fill heuristic with different input orderings as guided by the simulated annealing search procedure. The locations of the items assigned in the initial best-fit stage of the process remain fixed throughout, whereas the items placed during the second stage will move locations according to the orderings produced by simulated annealing.

Tabu Search Algorithms

The basic elements of a tabu search algorithm, apart from a feasible starting solution, are a move defining the neighborhood of the current solution, an evaluation of neighbors, a tabu list, an aspiration criterion, and some intensification and diversification strategies. A good example is *TSpack*, developed by Lodi et al. [60] for bin packing problems. Starting from a solution in which each item is packed into a different bin, a move consists in selecting a target bin to be emptied, selecting an item in it, j, and solving the packing problem composed of a subset of k bins plus item j. Improving moves that reduce the number of bins are accepted, as well as moves maintaining the number of bins but packing j out of the target bin. The number k of bins packed again is a parameter that controls the size of the neighborhood and the search effort. Small values of k correspond to intensification, while large values of k correspond to diversification. Nevertheless, when the search stalls without finding an acceptable move and k reaches its maximum value, other more aggressive diversification strategies are used to change the way in which the target bin is selected and to destroy part of the solution and build it again in a way in which the new solution is substantially different. Algorithm *TSpack* has been adapted by Bennell et al. [19] for solving a bin packing problem with due dates.

Alvarez-Valdes et al. [4] developed a tabu search algorithm for a two-dimensional cutting problem with guillotine cuts. The initial solution is built by a greedy constructive algorithm, and the neighbors of a solution are defined taking at random one rectangle, considering its adjacent rectangles, one at a time, and emptying the

minimum region containing both and respecting the guillotine cut structure. The region is filled again several times by using a GRASP algorithm, producing a set of neighboring solutions. The best non-tabu solution among them is selected. The tabu list keeps the rectangle being emptied, and its size varies dynamically after a given number of non-improving moves. Intensification and diversification strategies are defined by modifying the value of the items for the greedy constructive procedure. If items appearing more often in the good solutions are given larger values, the search intensifies around best solutions. If these items receive lower values, they are less attractive to be in the solutions and the search diversifies.

Tabu search algorithms for Identical item packing problems and cutting stock problems are also based on the use of blocks [5, 6, 70]. The moves are defined as block insertion, block reduction, or block expansion. IIPP is an example of flat landscape, with many solutions having the same number of items. In order to distinguish among them and lead the search toward promising regions of the solution space, the original objective function is modified by adding other elements, favoring solutions with good characteristics, such as having the empty spaces as near as possible. Long-term memory is used for intensification and diversification purposes, favoring or penalizing blocks that have appeared often throughout the search.

Tabu search algorithms can also be developed using ordered lists of items, as in Wei et al. [84]. The move is defined as the swap of two items, although only a subset of non-tabu moves are explored at each iteration, because they are evaluated, building the corresponding solutions (or decoding them in the terminology of genetic algorithms).

GRASP Algorithms

GRASP algorithms for Cutting and Packing are based on the constructive procedures and the local search moves described above [2,4,8,29]. If at the improvement phase several moves are defined, the GRASP algorithm can be hybridized with a variable neighborhood descent (VND) procedure that controls the moves at each GRASP iteration. An example can be found in the algorithm by Parreño et al. [68] for the bin packing problem, in which several moves based on removing the last part of the solution are combined with a more aggressive move that empties pairs of bins, as it appears in Fig. 20. Once a solution of N bins has been obtained, the target is set to obtain a solution with $N - 1$ bins. When the constructive procedure fills $N - 1$ bins, it stops, leaving some items unpacked. Then, the improving moves try to pack all these items into the $N - 1$ bins. Many of the proposed GRASP algorithms use the reactive GRASP strategy, in which the value for the parameter controlling the randomization procedure adjusts itself using the information of the search.

Alvarez-Valdes et al. [7] solve a two-stage cutting stock problem combining two GRASP algorithms, one based on items and the other based on strips. The solutions obtained by the two GRASP algorithms have different structures, and their combination by means of a path relinking procedure leads to very efficient solutions.

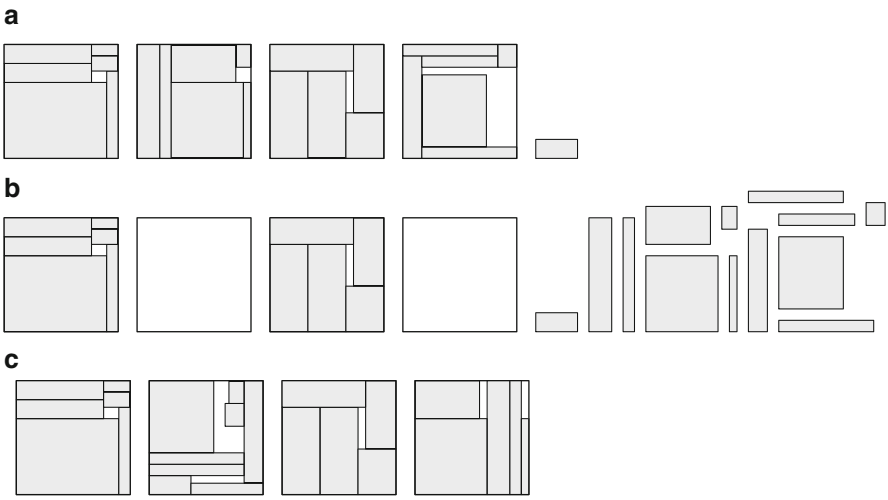


Fig. 20 Improvement move for a bin packing problem. (a) Current solution. (b) Emptying two bins. (c) Pack first the unpacked item and then the removed item

Path Relinking Algorithms

Path relinking algorithms have been applied to Cutting and Packing problems in connection with tabu search algorithms, as they were initially designed [4], but also connected to GRASP algorithms, in which a set of elite solutions is kept throughout the search. The work by Alvarez-Valdes et al. [9] on the multiple-bin-size bin packing problem combines several variants of path relinking. Figure 21, taken from it, shows the process in which some part of the layout of a guiding solution B is imposed on the initial solution A, producing a new solution that can be considered to lie in the path between them.

Tree Search Algorithms

In tree search algorithms, the solution space is searched using a tree in which each node represents a partial solution, starting from the empty solution at the root node. From each node, branching creates a new level with as many new nodes as possible ways of adding a new element, item, or block, to the partial solution. As this tree would grow exponentially, there are several ways of controlling its growth, limiting the successors of each node to only those most promising, according to a certain criterion, usually the value of the solution obtained by completing the partial solution with a greedy heuristic. When solving a container loading problem, Eley [34] uses a parameter *breadth* to control the number of successors and in his computational study fixes this breadth to 7. Ren et al. [71] follow the same strategy, but, at each level, only blocks getting the maximum value in at least one of the five proposed criteria are used to create a new branch. The scheme of Christensen and Rousoe [28] is also similar to Eley's, but they use a dynamic breadth which adjusts

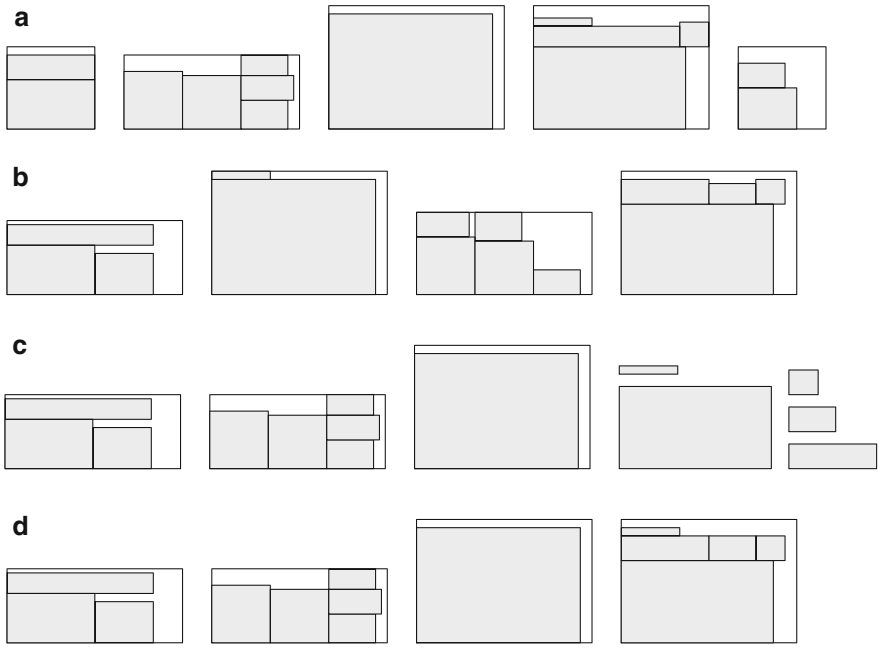


Fig. 21 Path relinking for a multiple-bin-size bin packing problem. (a) Initial solution A. (b) Guiding solution B. (c) Impose on A the first fin of B and remove bins with repeated items. (d) New complete solution

itself dynamically to exhaust the available time while guaranteeing the completion of the tree search.

Pisinger [69] also proposes a tree search for the container loading problem in which the container is first decomposed into a set of layers, whose height and width are the same as the container, and then each layer is decomposed into a set of strips. A partial solution at a node is composed of a set of layers, partially filling the depth of the container. At each branching node, a limited set of layers are considered to be added to the partial solution, according to a ranking rule, and the solution is increased by filling the selected layer with strips. Again, a limited set of strips is considered, and the strips are filled by solving 1D knapsack problems.

More recently, Fanslau and Bortfeldt [36] have developed a more complex tree search method. Again, a node of the tree is a partial solution, in this case composed of a set of heterogeneous blocks. A basic search phase transforms a partial solution with i blocks into another with $i + d$ blocks, where d is the depth of the search. The value of an augmented solution is obtained by completing it using a heuristic procedure. Basic searches can be linked to compose more complex search patterns, as shown in Fig. 22, in which, starting from a partial solution with i blocks, a search with depth $d = 3$ is divided into one or two basic searches. The basic search of depth 3 on the left-hand side produces eight augmented incomplete solutions

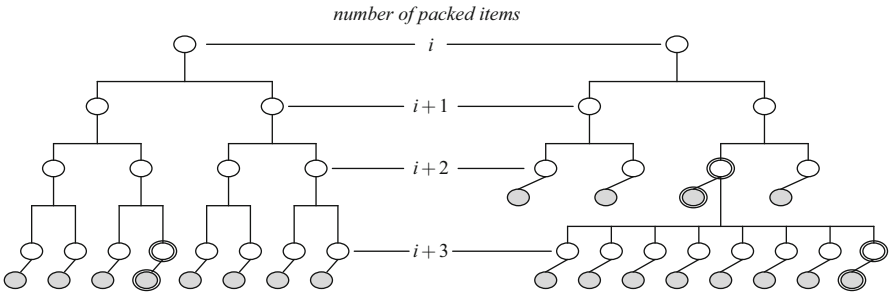


Fig. 22 Tree search structure of Fanslau and Bortfeldt

(white circles) that are completed to be evaluated (dark circles). The right-hand side shows two linked basic searches. The first one, of depth 2, produces four solutions which are completed. The best complete solution (double dark circle) indicates the incomplete solution from which to start the second basic search of depth 1. The best solution found indicates which block $i + 1$ is added to the partial solution.

The OR/AND graph approach developed by Arenales and Morabito [12] for the two-dimensional cutting problem can be also considered as a tree search procedure. At each node of the tree, there is a rectangle obtained by repeatedly cutting the original stock sheet and the cuts that can be applied to this rectangle define the successors of the node.

Other Metaheuristic Schemes

Burke et al. [23] propose a squeaky wheel optimization algorithm for the strip packing problem in which at each iteration the ordered list of items is decoded into a feasible solution by means of the Best-Fit algorithm. The items protruding a given height are considered badly placed and their value increases. Therefore, they get better positions in the ordered list and are placed earlier in the next iteration.

Guided local search has been used by Faroe et al. [37] for the 3D bin packing problem and by Hifi et al. [46] for the multidimensional knapsack problem. The guided local search moves out of a local maximum/minimum by penalizing particular solution features that it considers should not occur in a near-optimal solution. It defines a modified objective function, augmented with a set of penalty parameters on these features. The usual local search method is then used to improve the augmented objective function. The cycle of local search and penalty parameter update can be repeated as often as required.

Liu et al. [59] use the particle swarm optimization framework to develop an algorithm for the multiobjective bin packing problems. Each particle encodes a packing solution which includes the number of bins used and the order in which the items are packed into the bin by the *BLF* heuristic. Chen et al. [26] also use particle swarm optimization for solving a two-dimensional cutting problem, using

as a decoder a heuristic algorithm, based on the *BL* procedure but adapted to the specific problem being solved.

Model-Based Algorithms

For one- and multidimensional cutting stock problems, Gilmore and Gomory [39–41] proposed in 1961–1965 to solve the following integer program:

$$\text{Min} \sum_{q \in Q} c_q x_q$$

$$\text{s.t.} \quad \sum_{q \in Q} a_{iq} x_q \geq d_i, \quad i = 1, .., m \tag{1}$$

$$x_q \geq 0 \text{ and integer}, \forall\, q \in Q \tag{2}$$

where Q is the set of all feasible cutting patterns for all stock sheets S_p, x_q the number of times pattern q is used in the solution, a_{iq} the number of times item i appears in pattern q, d_i the demand of item i, and c_q the cost of pattern q.

As the set of patterns Q cannot be completely described except for very small problems, they developed a column generation scheme that can be summarized as follows:

1. Generate an initial set $\widetilde{Q}$ of m cutting patterns, where each pattern contains one type of item.
2. Solve the linear relaxation of the above formulated problem considering only the variables corresponding to patterns in $\widetilde{Q}$.
3. For each type of sheet S_p, solve the subproblem:

$$z_p = \text{Max} \sum_i \pi_i a_i$$

$$\text{s.t.} \quad \{a_1, \ldots, a_m\} \text{ is a feasible cutting pattern for } S_p$$

where π is the vector of dual prices of the LP solution. If, for some p, $z_p > c_p$, then the column corresponding to that solution is added to $\widetilde{Q}$ and step 2 is returned to in order to solve the enlarged LP problem. Otherwise, the current solution is rounded to get an integer solution and the process ends.

The question now is how to solve efficiently the subproblem of Step 3. Note that, even if the subproblem is solved by exact methods, the whole procedure is heuristic, due to the rounding step. The quality of the integer solution will depend not only on the algorithm used to solve the subproblem but also on the rounding procedure and on the structure of the demands.

Since the seminal work of Gilmore and Gomory, many authors have used this framework to solve Cutting and Packing problems or at least to have a reference or a starting point for new developments to overcome its limitations and adjust it to other problems. Examples go from Dyckhoff [32], Stadtler [76], and Valerio de Carvalho [81] for 1D problems, to Riehme et al. [72] for 2D cutting problems with extremely varying demands, or to Alvarez-Valdes et al. [3], who develop and compare several procedures to solve the subproblems as well as several rounding procedures.

Irregular Problems

Irregular packing problems are Cutting and Packing problems with convex and non-convex shapes (aka nesting problems). Although different shapes can be considered, from now onward, it will be assumed that the small items are described by polygons. In fact, the majority of the heuristics for these problems assume a polygonal shape for the small items. Moreover, although any type of Cutting and Packing problems, according to Wäscher et al. typology, can deal with irregular shapes, in practice one can almost only find placement problems and open-dimension problems. In the former, a large board and a set of irregular items with a maximum demand are given, and the goal is to lay out a subset of items on the board so that the waste is minimized, while in the latter, the large object has fixed width and variable length, big enough to accommodate all the items (in some papers, it is described as having infinite length), and the goal is to lay out all the small items, so that the length of the board that is effectively used to pack the items is minimized. This is the variant of the problem that is usually called irregular strip packing problem. Quite recent papers approach the irregular bin packing problem [62] and the irregular cutting stock problem [75], but hereafter the focus will be on the irregular packing problem with a single rectangular board, either with fixed or variable length, as this corresponds to the majority of the literature in the field.

Constructive Heuristics

Constructive heuristics build a solution for the problem by placing the pieces one by one. Two design decisions have to be made in a constructive heuristic: (1) given a set of pieces to place, in which sequence should the pieces be placed, and (2) given a set of pieces already placed on the board, where to place the next piece. The constructive heuristics vary on the strategies used for these two phases.

The bottom-left placement rule is the basis for the majority of the constructive heuristics available in the literature. The basic idea of the bottom-left rule is that each piece, in turn, is placed as low in the layout as possible and, for equal height, as much to the left as possible. The name of the rule, and the description just made,

Fig. 23 Bottom-left placement rule

supposes a vertical strip (width in the horizontal and length in the vertical). However, in irregular packing problems, the board/strip is most commonly represented with the width along the vertical direction and the length (which is aimed to be minimized in irregular strip packing) along the horizontal. Therefore, although the authors in the literature still refer to bottom-left rules, the pieces are actually placed as much to the left as possible (the minimum possible x-coordinate value), and for the same x-position, the lowest y-coordinate value is selected (Fig. 23). The basic implementation of the bottom-left placement rule places the piece on a feasible position of the layout (usually the upper-right corner of the board) and starts moving the piece to the left, considering a discrete set of points where pieces can be placed (a grid is defined over the board), and as soon as an overlap with an already placed piece is detected (resorting to one of the geometric representation and manipulation strategies presented in section "Geometry"), the movement is backtracked and a vertical move is tried. If successful, the horizontal movement is resumed. This process stops when a piece cannot move, either horizontally or vertically, without overlapping other pieces or crossing the borders of the board (Algorithm 1). It should be pointed out that in this bottom-left implementation, unused spaces may become holes in the layout, as pieces may block the access to those regions to the following pieces.

The first heuristic based on a bottom-left rule was proposed by Art [13]. On top of the bottom-left rule used to place each piece on the board, a compound rule for sequencing the pieces (selecting the next piece to place) was considered:

- consider the set of pieces that have not yet been placed;
- among these, select the set of pieces that can be placed on an x-coordinate that is within a given tolerance of the minimum x-coordinate where a piece can be placed;
- among these, select the set of pieces with an area within a given tolerance of the maximum area;

Algorithm 1: Bottom-left basic implementation

```
Place the piece on an up-right feasible placement position
while Piece does not overlap other pieces and is completely inside the board do
    Move piece one grid unit left
    if Piece overlaps another piece or is not completely inside the board then
        Move piece one grid unit right
        Move piece one grid unit down
    end if
end while
Move piece one grid unit up
Bottom-left position found
```

- among these, select the set of pieces with a minimum probable waste;
- among these, select the piece that can be placed in the lowest y-coordinate.

This compound rule is rather interesting as it highlights some of the main drawbacks of constructive heuristics.

To avoid the greedy effect of constructive heuristics, the authors try not to have a too rigid and fixed rule to select the next piece to place. Whatever the rule we choose, independently of how clever and fit to the problem it is, a greedy choice at the beginning originates leaving to the end the worst pieces to place, "destroying" solutions that looked promising at the beginning. Art tries to overcome this effect by balancing "doing well now" with "not doing bad in the end." Therefore, he does not commit to the piece that originates the most bottom-left placement, but instead considers a set of pieces that are near (doing well now) and then, among these, selects the biggest ones, which are harder to place in the end (not doing badly in the end).

Another drawback of constructive heuristics is, by definition, that they do not backtrack on the decisions made, i.e., once a piece is placed, it is not removed to try another piece. In each step, more than one piece may be tried, and the one considered best is chosen, but in later phases, this step cannot be revisited and a different decision made. Backtracking is a characteristic of search heuristics, which will be discussed in section "Local Search Heuristics". Art tries to overcome this drawback by attempting to anticipate the final effect of placing a piece through the concept of probable waste, i.e., the space that will become unusable after placing this piece. Figure 24 illustrates the concept. Similar implementations of this bottom-left placement rule were later on used by Oliveira and Ferreira [66] and Jakobs [50] as a block of more complex and sophisticated heuristics.

Art [13] introduced interesting concepts, but the computational resources available at that time prevented a proper exploration of these ideas (the geometric representation of the pieces had to be very rough), and the bottom-left placement rule was only recovered in the middle of the 1980s. Another bottom-left placement rule is proposed by Segenreich and Braga [74]. A grid is also defined over the board

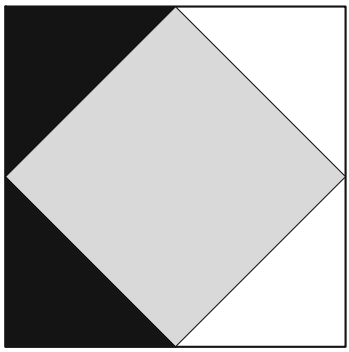


Fig. 24 Probable waste introduced by a piece (dark region on the left), according to Art [13]

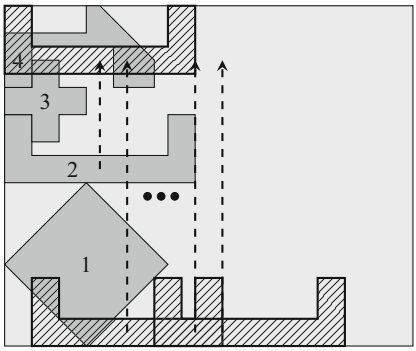


Fig. 25 Bottom-left placement rule, with the piece starting from unfeasible placement positions

and the pieces start from the origin of the coordinates. If the position is unfeasible, the piece is moved one grid unit along the y-axis and feasibility is rechecked. Once the width of the board is reached, the piece is moved one grid unit along the x-axis, the y-coordinate is reset, and the process is repeated until a feasible placement position is found. Given the way the piece moves in this search for a feasible position, this is the most bottom-left placement for the piece. To improve the heuristic performance, the initial position for a piece is not always the origin of the coordinates but the position where the last piece of that type was placed, as for sure there is not a most bottom-left position prior to that one (Fig. 25). The most important characteristic of this implementation of the bottom-left placement rule is its hole-filling capacity, i.e., spaces left empty in the middle of the layout can be used to place smaller pieces that are next in the sequence. Dowsland et al. [30] and Takahara et al. [78] used also this bottom-left implementation.

These bottom-left implementations suppose discrete movements of the pieces within the grid. Resorting to the concept of no-fit polygon, Gomes and Oliveira [42] and Dowsland et al. [31] proposed implementations of the bottom-left placement rules with continuous (real-domain) placement positions. When placing a piece j on the board, Gomes and Oliveira describe the possible placement points by:

- the vertices of the no-fit polygons NFP_{ij} of piece j in relation to pieces $i \in \mathscr{I}$, where $\mathscr{I}$ is the set of pieces that have been already placed;

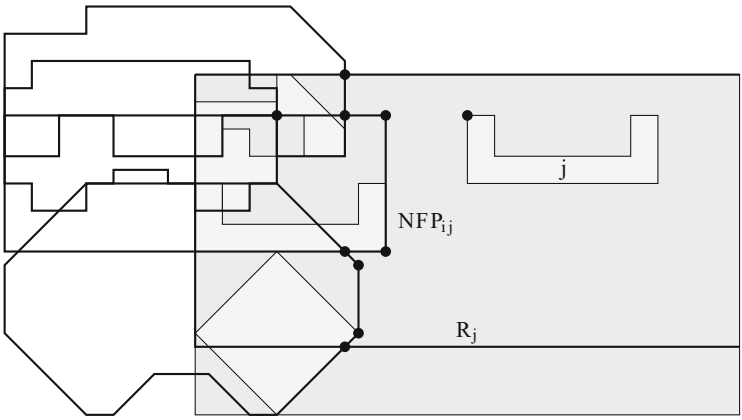


Fig. 26 Bottom-left placement rule, implemented over vertices of the NFP

- the vertices of the rectangle R_j that represents the extreme points where piece j can be placed without crossing the board border;
- the intersections of the edges of all NFP_{ij}, where $i \in \mathscr{I}$;
- the intersections of the R_j edges with the edges of all NFP_{ij}, where $i \in \mathscr{I}$.

Among these, the feasible placement points are on the boundary or outside all the NFP_{ij} and on the boundary or inside R_j, i.e., points, leading to a placement of piece j that does not overlap other already placed pieces and is inside the board (Fig. 26).

A completely different constructive heuristic was proposed in Oliveira et al. [67] and later on improved by Bennell and Song [18] – TOPOS.

In TOPOS, as in any other constructive algorithm, pieces are placed one by one on the board, building partial solutions. For each piece, the no-fit polygon of that piece in relation to the external contour of the set of pieces already placed (a partial solution), which form a kind of super-piece, is used to determine feasible placement points on the board. Once the placement point of a piece is determined, the piece is "merged" with the previous partial solution and the external contour is updated. In the original implementation of Oliveira et al., new pieces could only be placed on the exterior of the partial solution, while in Bennell and Song, pieces may be placed inside the partial solution, i.e., it allows hole-filling. The most distinctive characteristic of TOPOS is that the partial solutions do not have a fixed position on the board. Their placement on the board may change according to the need of having more space below, above, on the left, or on the right of the partial solution to better accommodate new pieces. TOPOS proposed two strategies to answer the two design decisions of a constructive heuristic: where to place a given piece and which should be the next piece to place.

To decide where to place a given piece, three different placement rules are used:

1. minimum area rectangular enclosure – the piece is placed on the point where the area of the rectangle that encloses the piece and the partial solution is minimum;
2. minimum length rectangular enclosure – the piece is placed on the point where the length of the rectangle that encloses the piece and the partial solution is minimum;
3. maximum overlap between the two rectangular enclosures – the piece is placed on the point where the area of the overlap between the piece rectangular enclosure and the partial solution rectangular enclosure is minimum.

The candidates to placement points (the points where the three criteria are evaluated) are the vertices of the no-fit polygon and special points on the edges of the no-fit polygon. On these special points, the criterion that is being used to select the placement point is optimized, when the constraint of belonging to that edge is considered. For example, in Fig. 27, point X corresponds to the point of the edge $\overline{AB}$ where the area of the rectangular enclosure of the two polygons is minimum [67].

Once selected where each potential next piece to place should be effectively placed, the several alternatives are compared using three different criteria, illustrated in Fig. 28:

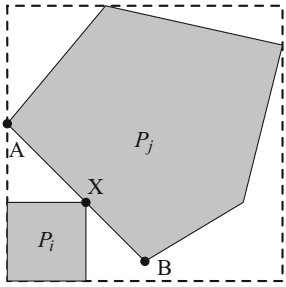


Fig. 27 X is a special point of edge $\overline{AB}$ where the area of the rectangle enclosure of the two polygons is minimum

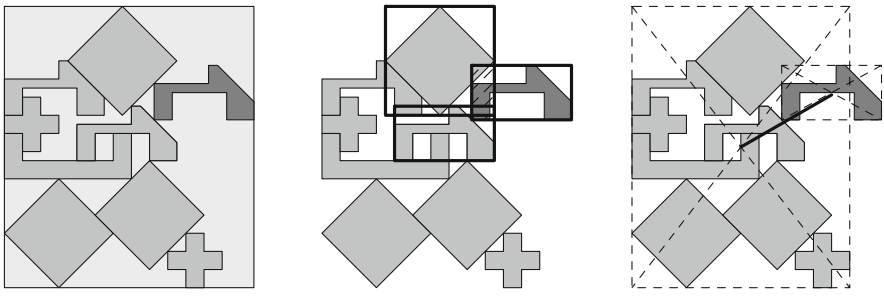


Fig. 28 TOPOS next-piece-to-place selection criteria

Waste – The piece adding less waste to the partial solution is chosen.
Overlap – The piece whose rectangular enclosure overlaps more with the rectangular enclosures of the pieces belonging to the current partial solution is chosen.
Distance – The piece that is placed nearest to the geometric center of the partial solution is chosen.

These criteria may be combined or used independently. *Waste* and *distance* can be used either in absolute value or in relative value, divided, respectively, by the area of the piece or by the half-perimeter of the rectangular enclosure of the piece. Bennell and Song [18] proposed two additional ways of using these criteria, besides the original aggregation by sum: *vector*, where the number of criteria in which a piece is better than the others is used to decide which is the best piece to place, and *priority*, where the criteria are used in a hierarchical way, with the lower levels being used to solve ties in the upper levels. Finally, in both implementations of TOPOS, all allowed orientations for each piece are tested.

Jostle [30], by Dowsland et al., is also a constructive heuristic where the way the next piece to place is selected is particularly interesting. Starting from a random or any priority-based sequence of the pieces, pieces are placed on the board following a leftmost strategy. The pieces are then sequenced by decreasing order of the x-coordinates of their rightmost vertices and placed again on the board but now according to a rightmost strategy. Pieces are reordered by increasing order of the x-coordinates of their leftmost vertices, and the process is repeated for a predefined number of iterations or until the solution does not change. This heuristic mimics the process of shaking objects on a tray so that they fit better together.

Local Search Heuristics

Local search heuristics may be divided into two groups: (1) heuristics that search over the actual problem solution space, i.e., complete layouts where all the pieces are laid down on the board; (2) heuristics that search over a solution representation or codification, which, in irregular packing problems, are usually the piece sequences. The second group requires an additional constructive placement heuristic, or decoder, to transform the sequence into an actual layout (Fig. 29). Neighborhood structures based on swaps (dotted line), inserts (dashed line), or rotations (solid line) are defined over the sequences.

Heuristics based on searching over sequences vary on how a sequence is changed into another one (the neighborhood structure) and on how the search process is controlled. Gomes and Oliveira [42] proposed a probabilistic two-exchange heuristic in which neighbor solutions are generated by swapping a pair of pieces in the sequence. Given all pairs of pieces that can be exchanged, three different criteria were explored to decide which should be the next sequence/center of neighborhood:

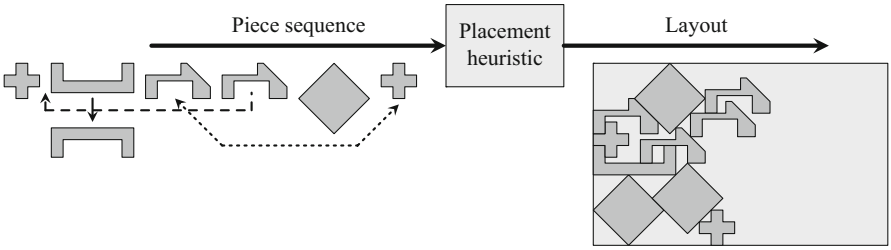


Fig. 29 Search over sequences, with swap (dotted line), insert (dashed line), and rotation (solid line) moves

- first-better – accepts the first swap that leads to a sequence which, decoded into a layout, is better than the current solution;
- best-better – accepts the best swap, i.e., the swap which, decoded into a layout, leads to the best solution among all the swaps;
- random-better – accepts a swap chosen randomly among all the swaps which, decoded into a layout, lead to solutions that are better than the current solution.

The search stops when no better solution can be generated from the current sequence.

More sophisticated search control strategies can be developed building on this "search over sequence" paradigm, as will be seen in section "Metaheuristics."

An alternative search procedure building on piece sequences is based on tree search methods. Albano and Sapuppo [1] proposed the very first tree search algorithm for irregular packing problems. At each level of the tree, a new piece was placed on the board. The root of the tree represented the empty board and the leaves a full layout. The authors applied strategies to improve the search efficiency by reducing the number of nodes explored. Some examples of these strategies are limiting the number of child nodes generated from a node and limiting the number of nodes kept in the list of nodes waiting to be explored. The decision to keep a node is based on indicators such as the waste of the partial solution or the waste added by placing a piece. These ideas became later rather popular under the name of *beam search* (see section "Metaheuristics").

The other group of local search heuristics operates over the actual layouts, directly moving around pieces by using the usual neighbor generation operators: swapping two pieces, moving a piece to another place, and rotating a piece (Fig. 30). In this type of heuristics, feasibility is usually not enforced, i.e., solutions where pieces overlap are allowed during the search, as only accepting moves to placements with no overlap would restrict the search space and eventually lead to premature stop in poor local optima.

This means that the overlap has to be penalized in the objective function. Once overlap between two pieces is detected, different measures of the overlap can be

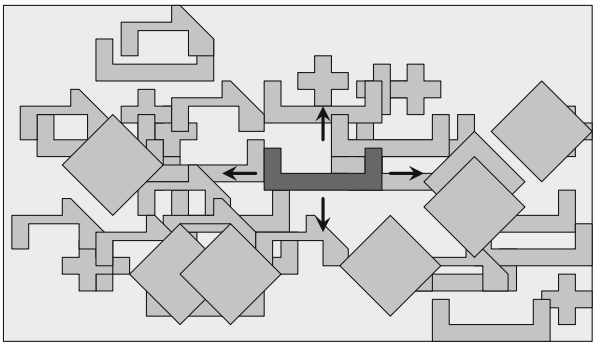


Fig. 30 Search over layouts: illustration of a search move where a piece is selected and moved to a different position

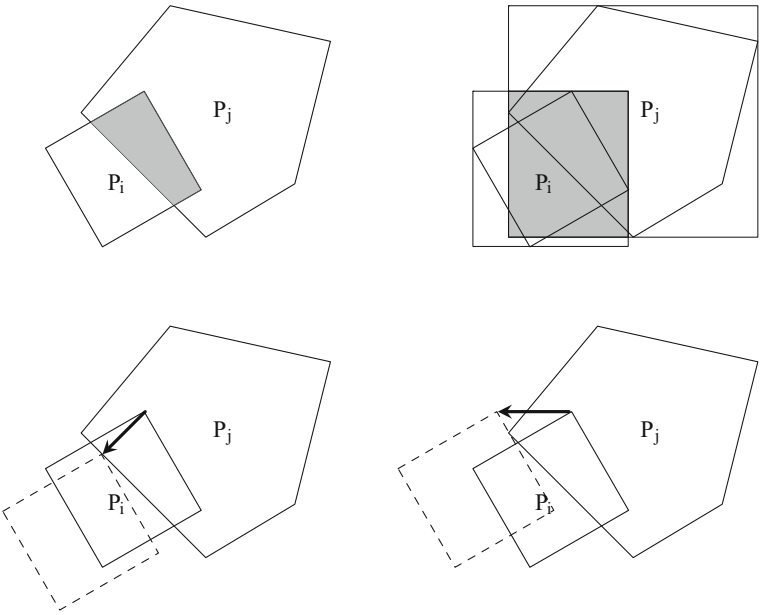


Fig. 31 Different measures of overlap can be used in the objective function: area (up-left), enclosing rectangle intersection area (up-right), penetration depth (down-left), and orthogonal penetration depth (down-right)

used in the objective function, being the most common the area, the enclosing rectangle intersection area, the penetration depth, and the orthogonal penetration depth (Fig. 31).

The use of the overlap measure in the objective function will depend on the type of problem tackled:

- For problems where the width and the length of the board are fixed, the set of pieces to be placed is given, and the objective is to find a placement such that the overlap is minimized. If a solution with no overlap is found, the heuristic is run with an additional piece, so that the number of pieces placed on the board is maximum.
- For problems where the board has a fixed width and a variable length, all pieces have to be placed. The objective is then to find a placement with no overlap such that the used board's length is minimized. To achieve that goal, the objective function has two components: the length of the layout and the amount of overlap. The weight of the overlap component determines the behavior of the search algorithm: a high weight leads to a fast convergence to a layout with no overlap but potentially with a big length; a low weight may lead to a solution with a small length but with some overlap, an infeasible solution for the irregular packing problem.

Because of their intrinsic difficulty in eliminating overlap, these heuristics have been mainly used embedded in more sophisticated solution approaches, either hybridized with mathematical programming models (section "Mathematical Programming-Based Heuristics") or in metaheuristic frameworks (section "Metaheuristics").

Mathematical Programming-Based Heuristics

Most of the mathematical programming models for the irregular packing problem rely on the concept of no-fit polygon to write the no-overlap constraints. Consider the polygons P_i and P_j and their $NFP_{P_i P_j}$, represented in Fig. 32. Guaranteeing that P_j does not overlap P_i is just a question of imposing conditions to the variables representing the coordinates of the placement point of P_j, i.e., x_j and y_j. These conditions must lead to the consequence that (x_j, y_j) is on the exterior or on the boundary of $NFP_{P_i P_j}$. The mathematical programming models that use the placement points of the pieces as decision variables (e.g., Gomes and Oliveira [43], Fischetti and Luzzi [38] and Alvarez-Valdes et al. [10]) describe the exterior of the no-fit polygon in different ways. The most common approach uses a covering model, i.e., decomposes the exterior into a set of regions that may overlap but cover completely the exterior of the no-fit-polygon. In the example depicted in Fig. 32, the equations of the supporting lines of the edges of the no-fit polygon, denoted by $e_{ijk}(x_i, y_i, x_j, y_j)$, $k = 1, \ldots, 12$, would provide the following systems of linear inequalities (in this case, each system has either one or three linear constraints):

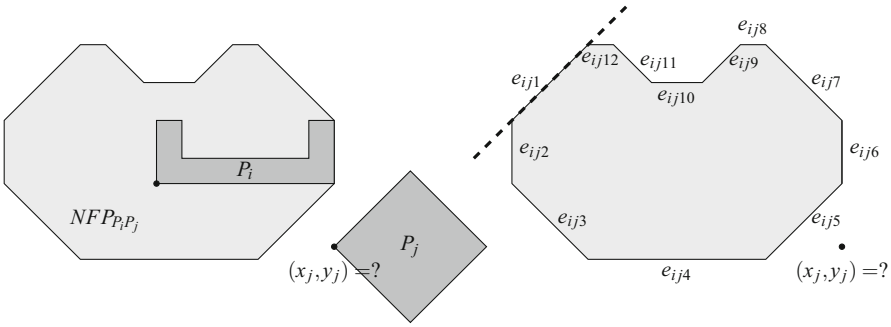


Fig. 32 Geometric information provided by the no-fit polygon to write the no-overlap constraints in a mathematical programming model

$$
\begin{cases}
e_{ij1}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij2}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij3}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij4}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij5}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij6}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij7}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij8}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij9}(x_i, y_i, x_j, y_j) \geq 0 \wedge e_{ij10}(x_i, y_i, x_j, y_j) \geq 0 \wedge e_{ij11}(x_i, y_i, x_j, y_j) \geq 0 \\
e_{ij12}(x_i, y_i, x_j, y_j) \geq 0
\end{cases}
$$

As only one of the systems has to hold, this leads to the use of auxiliary binary variables to model the disjunction of constraints. Notice that this set of systems of linear inequalities has to be defined for each pair of polygons, and at least one inequality or system of inequalities by pair of polygons has to hold. An alternative to this covering model is the partitioning model (e.g., [10]) where the systems of inequalities are defined so that their geometric representations on the plan do not overlap. In this case, exactly one inequality or system of inequalities has to hold by pair of polygons. However, in any case, a feasible placement for all pieces (a layout) corresponds to fixing specific binary variables. The inequality or system of inequalities that hold for each pair of polygons on this layout will be henceforth designated as the active constraints of the layout and altogether build a linear programming model, as the non-fixed variables (the placement coordinates of all pieces) are real-domain variables.

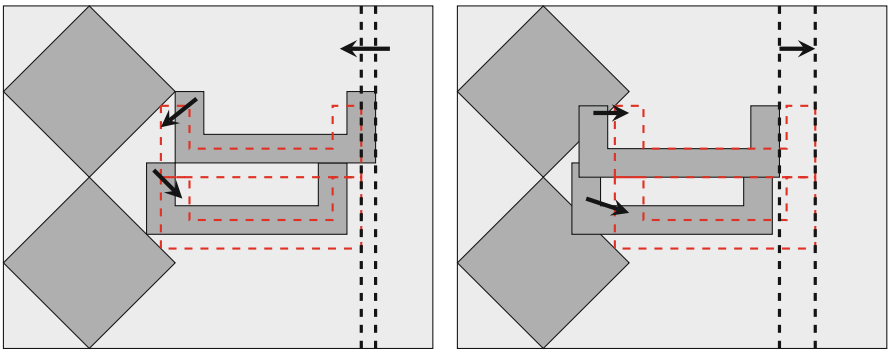


Fig. 33 Applying the compaction (on the left) and separation (on the right) models, with the solid gray pieces representing the initial solution and the dashed outlines representing the optimal solution

The linear programming model describing a feasible layout does not fix the absolute positions of the pieces but just their relative positions. Therefore, depending on the objective function, different layouts may be obtained. If the objective function is the minimization of layout's length (i.e., the rightmost vertex of all pieces' vertices), then the *compaction problem* is defined. In this problem, pieces are coordinately and continuously moved so that their relative positions are kept, but the length of the layout is minimized, as represented in Fig. 33 [16, 43, 63, 77]. It is also possible to use a similar model to eliminate overlap in an unfeasible layout (Fig. 33). For that, an active constraint has to be chosen for each pair of pieces, and for those pairs that overlap, an artificial variable is added so that the constraint holds with a nonzero value of the artificial variable and the objective function will be the minimization of the sum of the artificial variables. If the optimal solution of the linear programming has all artificial variables equal to zero, then the layout is feasible (without overlap).

Based on these compaction and separation models, hybrid solution methods can be developed [16, 43]. The basis of these methods is local search heuristics that search over the layouts and generate new solutions by swapping two pieces or moving a single piece to a different position on the layout (insertion move). By doing this, most probably the layout will have overlaps. A separation problem will be solved to try to eliminate the overlap, and if it obtains a feasible solution, the solution will be improved through a compaction problem. The way the search over the solution space is controlled varies a lot, but metaheuristic approaches have been used most of the times (see section "Metaheuristics").

A different idea was explored by Martinez-Sykora [61] which is to design a constructive heuristic based on the mixed-integer mathematical programming model, i.e., with both the continuous and binary variables. In this approach, a piece is inserted in the layout by adding to the model its placement variables, the constraints concerning all the no-fit polygons of this piece in relation to all pieces already placed in the layout, and the binary variables related to

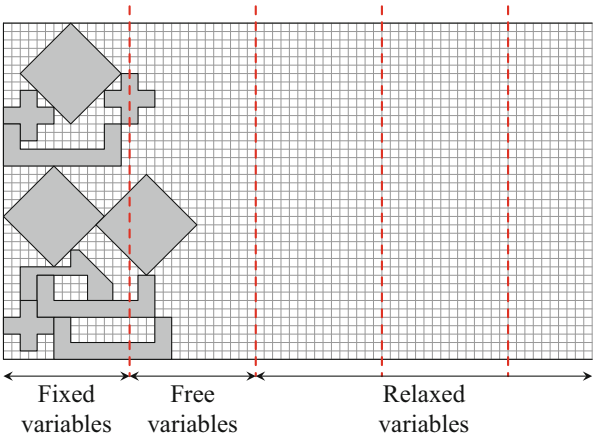


Fig. 34 Relax-and-fix iteration applied to the *dotted-board model*

the edges of these no-fit polygons. All binary variables concerning the pieces already placed are kept fixed in their previous values (i.e., the relative positions of the pieces already placed cannot change), but their continuous placement variables are set free, to enable adjustments to make room for the new piece.

Another approach was proposed by Cherri et al. [27]. This heuristic is based on a different model, the *dotted-board model* [79], a discrete model, where the pieces are constrained to be placed on a discrete set of points of the layout (a grid). The decision variables are the points on the grid and state if a piece of a given type is placed on that point or not. To impose the no-overlap constraints, no-fit polygons are used to exclude points that would lead to overlap situations from being used as placement points. Although this models solves larger instances than previous models, the *dotted-board model* is not yet viable for large instances, and a relax-and-fix heuristic with local branching constraints is proposed in [27]. Following the relax-and-fix paradigm, in each iteration of the method, there are a set of variables that are already fixed, another one that is free to be set by the model, and a third set of variables that are relaxed. In the following iteration, the variables of the second set become the first set variables, a group of variables of the third set is promoted to the second set, and new variables are added to the third set. The method stops when the third set is empty. To solve the irregular packing problem, the board is divided into vertical slices, and the decision variables belonging to each slice will progressively be relaxed, free, and fixed (Fig. 34). In order to add some flexibility in the previously placed pieces, a few pieces in the fixed set are also set free (local branching constraints).

Metaheuristics

The majority of the most recent approaches to the irregular problem solution are metaheuristics. However, the building blocks of these, sometimes rather sophisticated, search methods are the constructive, local search, and mathematical model-based heuristics presented in the previous sections. After all, metaheuristics are "just" a more sophisticated way of conducing the search over the solution space, so that it does not stop at local minima.

Two of the very first metaheuristic approaches for the irregular packing problem were already referred in the previous section [16, 43]. In both cases, pieces are moved around the layout by swap and insert moves; overlap is eliminated by a separation algorithm and compacted again by a compaction algorithm. In Bennell and Dowsland [16], the search is guided by a tabu search procedure, while in Gomes and Oliveira [43], simulated annealing is used.

Egeblad et al. [33] propose a local search scheme in which the neighborhood is any horizontal or vertical translation of a given polygon from its current position. The goal is to minimize overlap for a fixed layout length, where the minimization of the layout length is achieved by iteratively reducing the layout length. To escape local minima, the metaheuristic-guided local search is applied as described in Algorithm 2.

A three-level algorithm is proposed by Imamichi et al. [48] for the irregular strip packing problem. As the length of the board is variable, the upper level controls it by increasing the length when it is not possible to place all the pieces without overlap and decreases it when the layout is feasible. The algorithm stops when, after having a feasible solution (without overlap), decreasing the length originates again a layout with overlap. The second level is responsible for minimizing the overlap, given a board with fixed length. An iterated local search metaheuristic is used in which the perturbation phase consists in swapping two randomly chosen polygons and the local search phase is a separation algorithm that aims to eliminate existing overlap. The separation algorithm (third level of the approach) is based on the resolution of a mathematical programming model with a nonlinear objective function. The nonlinearity of the objective function is derived from the use of the penetration depth as the overlap measure.

Umetani et al. [80] propose an approach that is a mix of the two previous ones. For the overlap minimization problem, the orthogonal penetration depth is used, while guided local search is used to generate new solutions during the search for the best layout given a fixed length for the board. The latter starts with a value large enough to admit a feasible solution, and then it is iteratively decreased until the best solution found with the guided local search method has overlap. The algorithm stops and returns the solution obtained with the previous board's length.

The same three-level structure can be found in Leung et al. [57]. The overall approach (Algorithm 3) is rather complex and is based on two neighborhood structures, one based on swapping two polygons and the other based on moving a polygon to a different position in the layout. A nonlinear programming model

Algorithm 2: Guided local search for irregular packing

Given a set of polygons $\mathscr{P}$.
Generate an initial layout p.
for each pair of polygons $s_i, s_j \in \mathscr{P}$ **do**
  Set penalty counter $\Phi_{ij} = 0$.
end for
while $g(p) > 0$ (p contains overlap) **do**
  Local search:
  while p is not a local minimum **do**
    Select polygon s_i.
    Create p' from p by sliding s_i horizontally and vertically
    to the position in which $h(p')$ is minimized, with
    $h(p) = g(p) + \lambda \sum_i \sum_j \Phi_{ij} I_{ij}(p)$, in which $I_{ij}(p) = 0$ if $\text{overlap}_{ij}(p) = 0$, 1 otherwise.
    Set $p = p'$.
  end while
  penalize:
  for each pair of polygons $s_i, s_j \in \mathscr{P}$ **do**
    Compute the utility function $\mu_{ij}(p) = I_{ij}(p) \frac{\text{overlap}_{ij}(p)}{1+\Phi ij}$.
  end for
  for each pair of polygons $s_i, s_j \in \mathscr{P}$ such that μ_{ij} is maximal **do**
    Set $\Phi ij = \Phi ij + 1$.
  end for
end while
Return p

is used to minimize the overlap during the search process (separation algorithm), and the overall search has two phases: local search and tabu search. In the end, a heuristic is run to improve the final solution. The parameters r_{dec} and r_{inc} control how aggressively the length of the board is decreased and increased during the search procedure.

The same structure is used in Elkeran [35]. The initial solution is generated by the constructive algorithm by Gomes and Oliveira [42]. To improve the efficiency of the algorithm, a pairwise clustering of the most non-convex pieces is done, although these clusters do not show up in the final solutions. The board's length is also alternately decreased and increased. The most innovative part of the algorithm is the use of the populational metaheuristic cuckoo search for the overlap minimization level. The cuckoo search algorithm uses a balanced combination of a local random walk and a global explorative random walk, controlled by a switching parameter, to evolve from one population to another. Cuckoo search is responsible for moving a polygon on the board to a less overlapping position, while guided local search is used to escape local minima.

Algorithm 3: Extended local search for irregular packing

Best length: L_{best}.
Current length: L_{curr}.
Generate an initial solution $\longrightarrow$ L_{curr}
$L_{best} = L_{curr}$.
while in a time limit **do**
  Decrease the length of the board: $L_{curr} = (1 - r_{dec})L_{best}$.
  Run local search algorithm based on swapping two pieces and minimizing the consequent overlap by a nonlinear separation algorithm $\rightarrow$ L_{curr}.
  if local optimum is feasible (has no overlap) **then**
    $L_{best} = L_{curr}$
  else
    $L_{curr} = (1 + r_{inc})L_{best}$.
    Run tabu search algorithm based on moving a piece to a different position on the layout $\rightarrow$ L_{curr}.
  end if
end while
Run heuristic procedure over the best solution found:
small movements of the pieces + overlap minimization + separation algorithm
Return best solution found

All the previously presented metaheuristics search over the layouts. Other metaheuristic approaches are based on searching over sequences. It is the case of Bennell and Song [18] that propose the use of the beam search metaheuristic, which is a tree search procedure that tries to avoid the drawbacks of fixed tree search strategies (e.g., depth-first, breadth-first) without the computational burden of an exhaustive search. In this approach, solutions are represented by a sequence of pieces to be packed. To decode these sequences, the modified version of TOPOS (see section "Constructive Heuristics") is used. The tree represents the construction of a partial solution where each node adds a new element to the sequence. At each level, a local evaluation function (in this case TOPOS evaluation criteria) evaluates all child nodes. Child nodes will only compete with other children branching from the same parent node. This evaluation provides a crude approximation of the solution quality by measuring the incremental cost of adding an element to the partial solution. A subset of size α of the best nodes, using this evaluation, is selected (α is the filter width). The selected filtered nodes are then subject to global evaluation, and the best β nodes are retained for branching (β is the beam width).

Another metaheuristic based on searching over sequences can be found in Sato et al. [73]. It is a bi-level approach with the top level controlling the board's length (increasing and decreasing it) and the bottom level finding a layout without overlap. The overlap minimization algorithm is simulated annealing. The initial solution is random, and the neighborhood is based on modifications of the placement sequence

(two items are swapped), piece orientation (several rotations are allowed), and piece position (for non-simple NFPs, in which contour or degenerated vertex or edge the placement occurs). The decoding heuristic is the bottom-left NFP vertex-oriented heuristic of Gomes and Oliveira [42].

Conclusion

Due to the combinatorial nature of Cutting and Packing problems, exact techniques are not capable of efficiently tackling large instances, and therefore heuristic approaches must be used. In this chapter, the most important heuristic techniques that have been applied to Cutting and Packing problems have been presented, going from simple constructive heuristics to rather complex hybrid metaheuristic approaches and including local search heuristics and mathematical model-based approaches. To frame these methods, a comprehensive section on Cutting and Packing problems was written, including the discussion of these problems under the typology of Wäscher et al. and the rather important geometric representation layer that these problems require.

Given the broad range of Cutting and Packing problems, we chose to focus on two- (both rectangular and irregular) and three-dimensional (just rectangular) problems, as these better illustrate the full use of heuristic methods when approaching this class of problems. One-dimensional problems have no geometric considerations, on one hand, and on the other hand are better solved by exact methods than the other problems. Among the two-dimensional problems, and due to space constraints, circle packing was not referred to in this chapter. However, not only the main approaches closely follow what has been done for the other two-dimensional problems, but also the interested reader can easily refer to the review by Hifi and M'Hallah [45] and the following work. Integrating Cutting and Packing problems with other optimization problems that in real life have a direct relation to them, in the sense that the solutions of one problem impact the resolution of the other, is a recent trend in the field. Once more, given the difficulty and complexity of the problems, heuristic techniques have to be used to tackle the integrated problems, such as vehicle routing and container loading [64].

Cross-References

- Genetic Algorithms
- GRASP
- Guided Local Search
- Iterated Local Search
- Matheuristics
- Random-Key Genetic Algorithms

- Tabu Search
- Variable Neighborhood Descent
- Variable Neighborhood Search

References

1. Albano A, Sapuppo G (1980) Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Trans Syst Man Cybern 10(5):242–248
2. Alonso M, Alvarez-Valdes R, Parreño F, Tamarit J (2014) A reactive GRASP algorithm for the container loading problem with load-bearing constraints. Eur J Ind Eng 8:669–694
3. Alvarez-Valdes R, Parajon A, Tamarit J (2002) A computational study of LP-based heuristic algorithms for the two-dimensional guillotine cutting stock problems. OR Spectr 24:179–192
4. Alvarez-Valdes R, Parajon A, Tamarit J (2002) A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Comput Oper Res 29:925–947
5. Alvarez-Valdes R, Parreño F, Tamarit J (2005) A tabu search algorithm for the pallet loading problem. OR Spectr 27:43–61
6. Alvarez-Valdes R, Parreño F, Tamarit J (2007) A tabu search algorithm for a two-dimensional non-guillotine cutting problem. Eur J Oper Res 183:1167–1182
7. Alvarez-Valdes R, Marti R, Parajon A, Tamarit J (2007) GRASP and path relinking for the two-dimensional two-stage cutting-stock problem. INFORMS J Comput 19:261–272
8. Alvarez-Valdes R, Parreño F, Tamarit J (2008) Reactive GRASP for the strip packing problem. Comput Oper Res 35:1065–1083
9. Alvarez-Valdes R, Parreño F, Tamarit J (2013) A GRASP/Path relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. Comput Oper Res 40: 3081–3090
10. Alvarez-Valdes R, Martinez A, Tamarit J (2013) A branch & bound algorithm for cutting and packing irregularly shaped pieces. Int J Prod Econ 145(2):463–477
11. Araujo O, Armentano V (2007) A multi-start random constructive heuristic for the container loading problem. Pesquisa Operacional 27(2):311–331
12. Arenales M, Morabito R (1995) An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. Eur J Oper Res 84:599–617
13. Art RC (1966) An approch to the two dimensional, irregular cutting stock problem. IBM Cambridge Scientific Center Report, pp 1–35
14. Baker B, Coffman E, Rivest R (1980) Orthogonal packings in two dimensions. SIAM J Comput 9(4):846–855
15. Below G, Scheithauer G, Mukhacheva E (2008) One-dimensional heuristics adapted for two-dimensional rectangular strip packing. J Oper Res Soc 59:823–832
16. Bennell JA, Dowsland KA (2001) Hybridising tabu search with optimisation techniques for irregular stock cutting. Manag Sci 47(8):1160–1172
17. Bennell JA, Oliveira JF (2008) The geometry of nesting problems: a tutorial. Eur J Oper Res 184(2):397–415
18. Bennell JA, Song X (2010) A beam search implementation for the irregular shape packing problem. J Heuristics 16(2):167–188
19. Bennell J, Lee L, Potts C (2013) A genetic algorithm for two-dimensional bin packing with due dates. Int J Prod Econ 145:547–560
20. Bortfeldt A, Gehring H (2001) A hybrid algorithm for the container loading problem. Eur J Oper Res 131:143–161
21. Burke E, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem. Oper Res 52(4):655–671
22. Burke E, Kendall G, Whitwell G (2009) A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. INFORMS J Comput 21(3):505–516

23. Burke E, Hyde M, Kendall G (2011) A squeaky wheel optimization methodology for two-dimensional strip packing. Comput Oper Res 38:1035–1044
24. Ceschia S, Schaerf A (2013) Local search for a multi-drop multi-container loading problem. J Heuristics 19:275–294
25. Chazelle B (1983) The bottom-left bin-packing heuristic: an efficient implementation. IEEE Trans Comput 32(8):697–707
26. Chen W, Zhai P, Zhu H, Zhang Y (2014) Hybrid algorithm for the two-dimensional rectangular layer-packing problem. J Oper Res Soc 65:1068–1077
27. Cherri L, Toledo F, Carravilla MA (2016) A model-based heuristic for the irregular strip packing problem. Pesquisa Operacional 36(3):447–468
28. Christensen S, Rousoe D (2009) Container loading with multi-drop constraints. Int Trans Oper Res 16:727–743
29. da Silveira J, Miyazawa F, Xavier E (2013) Heuristics for the strip packing problem with unloading constraints. Comput Oper Res 40:991–1003
30. Dowsland KA, Dowsland WB, Bennell JA (1998) Jostling for position: local improvement for irregular cutting patterns. J Oper Res Soc 49:647–658
31. Dowsland KA, Vaid S, Dowsland WB (2002) An algorithm for polygon placement using a bottom-left strategy. Eur J Oper Res 141(2):371–381
32. Dyckhoff H (1981) A new linear programming approach to the cutting stock problem. Oper Res 29:1092–1104
33. Egeblad J, Nielsen BK, Odgaard A (2007) Fast neighborhood search for two- and three-dimensional nesting problems. Eur J Oper Res 183(3):1249–1266
34. Eley M (2002) Solving container loading problems by block arrangement. Eur J Oper Res 141:393–409
35. Elkeran A (2013) A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. Eur J Oper Res 231(3):757–769
36. Fanslau T, Bortfeldt A (2010) A tree search algorithm for solving the container loading problem. INFORMS J Comput 22(2):222–235
37. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for the three-dimensional bin-packing problem. INFORMS J Comput 15(3):267–283
38. Fischetti M, Luzzi I (2009) Mixed-integer programming models for nesting problems. J Heuristics 15(3):201–226
39. Gilmore P, Gomory R (1961) A linear programming approach to the cutting-stock problem. Oper Res 9:849–859
40. Gilmore P, Gomory R (1963) A linear programming approach to the cutting-stock problem – part II. Oper Res 11:863–888
41. Gilmore P, Gomory R (1965) Multistage cutting stock problems of two and more dimensions. Oper Res 13:94–120
42. Gomes AM, Oliveira JF (2002) A 2-exchange heuristic for nesting problems. Eur J Oper Res 141(2):359–370
43. Gomes AM, Oliveira JF (2006) Solving irregular strip packing problems by hybridising simulated annealing and linear programming. Eur J Oper Res 171(3):811–829
44. Gonçalves J, Resende M (2013) A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 145:500–510
45. Hifi M, M'Hallah R (2009) A literature review on circle and sphere packing problems: models and methodologies. Adv Oper Res 2009(150624):1–22
46. Hifi M, Michrafy M, Sbihi A (2004) Heuristic algorithms for the multiple-choice multidimensional knapsack problem. J Oper Res Soc 55:1323–1332
47. Hopper E, Turton B (2001) An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. Eur J Oper Res 128:34–57
48. Imamichi T, Yagiura M, Nagamochi H (2009) An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Discret Optim 6(4): 345–361

49. Iori M, Martello S, Monaci M (2003) Metaheuristic algorithms for the strip packing problem. In: Pardalos PM, Korotkikh V (eds) Optimization in industry: new frontiers. Kluwer Academic Publishers, Dordrecht, pp 159–179
50. Jakobs S (1996) On genetic algorithms for the packing of polygons. Eur J Oper Res 88: 165–181
51. Lai K, Chan J (1997) Developing a simulated annealing algorithm for the cutting stock problem. Comput Ind Eng 32:115–127
52. Lesh N, Mitzenmacher M (2006) Bubblesearch: a simple heuristic for improving priority-based greedy algorithms. Inf Process Lett 97(4):161–169
53. Lesh N, Marks J, McMahon A, Mitzenmacher M (2003) New heuristic and interactive approaches to 2D rectangular strip packing. Technical report TR2003-18. Mitsubishi Electric Research Laboratories, Cambridge
54. Leung T, Chan C, Troutt M (2003) Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. Eur J Oper Res 145:530–542
55. Leung S, Zhang D, Sim K (2011) A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. Eur J Oper Res 215:57–69
56. Leung S, Zhang D, Zhou C, Wu T (2012) A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. Comput Oper Res 39:64–73
57. Leung SC, Lin Y, Zhang D (2012) Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem. Comput Oper Res 39(3):678–686
58. Liu D, Teng H (1999) An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. Eur J Oper Res 112:413–420
59. Liu D, Tan K, Huang S, Goh C, Ho W (2008) On solving multiobjective bin packing problems using evolutionary particle swarm optimization. Eur J Oper Res 190:357–382
60. Lodi A, Martello S, Vigo D (2004) TSpack: a unified tabu search code for multi-dimensional bin packing problems. Ann Oper Res 131:203–213
61. Martinez Sykora A (2013) Nesting problems: exact and heuristic algorithms. Phd thesis, Univesity of Valencia
62. Martinez-Sykora A, Alvarez-Valdes R, Bennell J, Tamarit JM (2015) Constructive procedures to solve 2-dimensional bin packing problems with irregular pieces and guillotine cuts. Omega 52:15–32
63. Milenkovic VJ, Daniels K (1999) Translational polygon containment and minimal enclosure using mathematical programming. Int Trans Oper Res 6:525–554
64. Moura A, Oliveira JF (2008) An integrated approach to the vehicle routing and container loading problems. OR Spect 31(4):775–800
65. Mukhacheva E, Belov G, Kartak V, Mukhacheva A (2000) Linear one-dimensional cutting-packing problems: numerical experiments with sequential value correction method (SVC) and a modified branch-and-bound method (MBB). Pesquisa Operacional 20:153–168
66. Oliveira JF, Ferreira JS (1993) Algorithms for nesting problems. In: Vidal RVV (ed) Applied simulated annealing. Lecture notes in economics and mathematical systems. Springer, Berlin, pp 256–279
67. Oliveira JF, Gomes AM, Ferreira JS (2000) TOPOS – a new constructive algorithm for nesting problems. OR Spektr 22(2):263–284
68. Parreño F, Alvarez-Valdes R, Oliveira J, Tamarit J (2010) Neighbourhood structures for the container loading problem: a VNS implementation. J Heuristics 16(1):1–22
69. Pisinger D (2002) Heuristics for the container loading problem. Eur J Oper Res 141:382–392
70. Pureza V, Morabito R (2006) Some experiments with a simple tabu search algorithm for the manufacturers pallet loading problem. Comput Oper Res 33:804–819
71. Ren J, Tian Y, Sawaragi T (2011) A tree search method for the container loading problem with shipment priority. Eur J Oper Res 214:526–535
72. Riehme J, Scheithauer G, Terno J (1996) The solution of two-stage guillotine cutting stock problems having extremely varying order demands. Eur J Oper Res 91:543–552

73. Sato AK, Martins TC, Tsuzuki MSG (2012) An algorithm for the strip packing problem using collision free region and exact fitting placement. Comput Aided Des 44(8):766–777
74. Segenreich SA, Braga LMPF (1986) Optimal nesting of general plane figures: a Monte Carlo heuristical approach. Comput Graph 10(3):229–237
75. Song X, Bennell Ja (2013) Column generation and sequential heuristic procedure for solving an irregular shape cutting stock problem. J Oper Res Soc 65(7):1037–1052
76. Stadtler H (1990) A one-dimensional cutting stock problem in the aluminium industry and its solution. Eur J Oper Res 44:209–223
77. Stoyan YG, Novozhilova MV, Kartashov AV (1996) Mathematical model and method of searching for a local extremum for the non-convex oriented polygons allocation problem. Eur J Oper Res 92(1):193–210
78. Takahara S, Kusumoto Y, Miyamoto S (2003) Solution for textile nesting problems using adaptive meta-heuristics and grouping. Soft Comput 7:154–159
79. Toledo FMB, Carravilla MA, Ribeiro C, Oliveira JF, Gomes AM (2013) The dotted-board model: a new MIP model for nesting irregular shapes. Int J Prod Econ 145:478–487
80. Umetani S, Yagiura M, Imahori S, Imamichi T, Nonobe K, Ibaraki T (2009) Solving the irregular strip packing problem via guided local search for overlap minimization. Int Trans Oper Res 16(6):661–683
81. Valerio de Carvalho J (2002) LP models for bin packing and cutting stock problems. Eur J Oper Res 141:253–273
82. Wäscher G, Hauß ner H, Schumann H (2007) An improved typology of cutting and packing problems. Eur J Oper Res 183:1109–1130
83. Wei L, Lim A (2015) A bidirectional building approach for the 2D constrained guillotine knapsack packing problem. Eur J Oper Res 242:63–71
84. Wei L, Oon W, Zhu W, Lim A (2011) A skyline heuristic for the 2D rectangular packing and strip packing problems. Eur J Oper Res 215:337–346
85. Wu Y, Huang W, Lau S, Wong C, Young G (2002) An effective quasi-human based heuristic for solving the rectangle packing problem. Eur J Oper Res 141:341–358
86. Zhu W, Oon W, Lim A, Weng Y (2012) The six elements to block-building approaches for the single container loading problem. Appl Intell 37:431–445

Diversity and Equity Models

32

Fernando Sandoya, Anna Martínez-Gavara, Ricardo Aceves, Abraham Duarte, and Rafael Martí

Contents

Introduction 980
Mathematical Models and Formulation 981
Diversity Models 982
Equity Models 982
Related Models 983
Correlation Between Models 984
Diversity, Dispersion, and Equity Examples 986
Metaheuristics 990
The Max-Min Model 990
The Max-Sum Model 991
The Max-Mean Model 993
The Max-MinSum Model 993

F. Sandoya
Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador
e-mail: fsandoya@espol.edu.ec

A. Martínez-Gavara
Departamento de Estadística e Investigación Operativa, Facultad de Matematicas, Universidad de Valencia, Valencia, Spain
e-mail: gavara@uv.es; Ana.Martinez-Gavara@uv.es

R. Aceves
Department of Systems, Universidad Nacional Autónoma de México, Mexico City, Mexico
e-mail: aceves@unam.mx

A. Duarte
Department of Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: abraham.duarte@urjc.es

R. Martí (✉)
Statistics and Operations Research Department, University of Valencia, Valencia, Spain
e-mail: Rafael.Marti@uv.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_61

The Min-Diff Model 995
Conclusions 996
Cross-References 996
References 997

Abstract

The challenge of maximizing the diversity of a collection of points arises in a variety of settings, and the growing interest of dealing with diversity resulted in an effort to study the management of equity. While the terms diversity and dispersion can be found in many optimization problems indistinguishable, we undertake to explore the different models behind them.

In particular, this chapter describes the mathematical models for two diversity and three equity models. Additionally, we also review two related models that have recently received special attention. This chapter also reviews heuristics and metaheuristics for finding near-optimal solutions for these problems, where constructive and local search-based methods, such as greedy randomized adaptive search procedure (GRASP) and tabu search, play an important role.

Keywords

Diversity · Dispersion · Equity

Introduction

The problem of maximizing diversity deals with selecting a subset of elements from a given set in such a way that the diversity among the elements is maximized [24]. Several models have been proposed to deal with this combinatorial optimization problem. All of them require a diversity measure, typically a distance function in the space where the objects belong. The definition of this distance between elements is customized to specific applications. As described in [25], models have applications in plant breeding, social problems, ecological preservation, pollution control, product design, capital investment, workforce management, curriculum design, and genetic engineering. As Scott Page states in his book [39]: *Diverse perspectives and tools enable collections of people to find more and better solutions and contribute to overall productivity*. As a result, the problem of identifying diverse groups of people becomes a key point in large firms and institutions, as described in [9].

The most studied model related to diversity is probably the Maximum Diversity Problem (MDP) also known as the Max-Sum Diversity Model [20], in which the sum of the distances between the selected elements is maximized. The Max-Min Diversity Problem (MMDP), in which the minimum distance between the selected elements is maximized, has been also well documented in recent studies [46]. Although the MDP and the MMDP are related, we should not expect a method developed for the MDP to perform well on the MMDP or vice versa. We illustrate it with an example in section "Correlation Between Models".

There are some models closely related to diversity called equity models, which incorporate the concept of fairness among candidates. These models, however, have been mostly ignored, and only very recently, specific solving methods have been proposed to them. Special attention deserves the work in [43] in which four different equity models are proposed. These models appear in the context of urban public facility location, diverse/similar group selection, and subgraph identification, in which one may address fair diversification or assimilation among members of a network. In this chapter we target both diversity and equity models.

When formulating a model, the definition of distance between elements is customized to the specific application. Generally speaking, it is assumed that each element in the application can be represented by a set of attributes. Let s_{ik} be the state or value of the k-th attribute of element i, where $k = 1, \cdots, K$. Then the distance between elements i and j may be defined as

$$d_{ij} = \sqrt{\sum_{k=1}^{K} \left(s_{ik} - s_{jk}\right)^2} \tag{1}$$

In this case, d_{ij} is simply the Euclidean distance between i and j, but other distance or affinity functions can be considered as well. A common affinity measure in the context of the equity is the cosine distance, computed as

$$d_{ij} = \frac{\sum_{k=1}^{K} s_{ik} s_{jk}}{\sqrt{\sum_{k=1}^{K} s_{ik}^2} \sqrt{\sum_{k=1}^{K} s_{jk}^2}} \tag{2}$$

The cosine similarity between two elements can be viewed as the angle between their attribute vectors, where a small angle between elements indicates a large degree of similarity. It takes values in $[-1, 1]$ reflecting the affinity between the individuals.

This chapter is organized as follows: In section "Mathematical Models and Formulation" we define the different diversity models and the correlation between their solutions. In section "Diversity, Dispersion, and Equity Examples" we review the different diversity measures, and their mathematical expression, and graphically explore their meaning. Finally, in section "Metaheuristics" we describe the most recent papers about heuristic and metaheuristic algorithms to solve the models previously introduced.

Mathematical Models and Formulation

Given a graph $G = (V, E)$, where V is the set of n nodes and E is the set of edges, let d_{ij} be the inter-element distance between any two elements i and j, let $M \subseteq V$ be the set of m selected elements, and define $U = V \setminus M$ as the set of unselected elements.

Diversity Models

The *Max-Min Diversity Problem* (MMDP) can be formulated as follows:

$$\begin{aligned} &\text{(MMDP) Maximize} && z_{MM}(x) = \min_{i<j} d_{ij} x_i x_j \\ &\text{st} && \sum_{i=1}^{n} x_i = m \\ & && x_i \in \{0,1\} \qquad\qquad i = 1,\ldots,n. \end{aligned} \tag{3}$$

The Maximum Diversity Problem, MDP, also called Max-Sum can be formulated in a similar way by simply replacing the objective function, $z_{MM}(x)$, in the formulation above with the expression $z_{MS}(x)$ as

$$z_{MS}(x) = \sum_{i<j} d_{ij} x_i x_j \tag{4}$$

The Max-Sum and Max-Min literature includes extensive surveys [1, 15, 31], exact methods [1, 20, 41], and heuristics [4, 12, 13, 20, 28, 30, 44, 46].

Equity Models

In [43], the authors summarize some of the different dispersion models defining them as follows. The *Max-Mean Dispersion Problem* (Max-Mean) can be formulated as the following 0–1 quadratic integer programming problem:

$$\begin{aligned} &\text{Maximize} && \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j}{\sum_{i=1}^{n} x_i} \\ &\text{st} && \sum_{i=1}^{n} x_i \geq 2 \\ & && x_i \in \{0,1\} \qquad i = 1,\ldots,n. \end{aligned} \tag{5}$$

In (5) the cardinality restriction is not imposed. This problem is a version of the Max-Sum problem where the number of elements to be selected is unknown. The generalized version of this problem is called *Generalized Max-Mean Dispersion Problem* and is formulated as follows:

$$\begin{aligned} &\text{Maximize} && \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j}{\sum_{i=1}^{n} w_i x_i} \\ &\text{st} && \sum_{i=1}^{n} x_i \geq 2 \\ & && x_i \in \{0,1\} \qquad i = 1,\ldots,n. \end{aligned} \tag{6}$$

where w_i is the weight assigned to element $i \in V$.

Then, the *Maximum MinSum Dispersion Problem* (Max-MinSum DP) consists in selecting a set $M \subseteq V$ in m elements such that the smallest total dispersion associated with each selected element i is maximized. The problem is formulated in [43] as follows:

$$\begin{array}{lll} \text{Maximize} & \left\{ \min\limits_{i:x_i=1} \sum\limits_{j:j\neq i} d_{ij} x_j \right\} & \\ \text{st} & \sum\limits_{i=1}^{n} x_i = m & \\ & x_i \in \{0,1\} & i = 1,\ldots,n. \end{array} \tag{7}$$

Finally, the *Minimum Differential Dispersion Problem* (Min-Diff DP) consists in finding the best subset $M \subseteq V$ with respect to the measure "Diff" defined in Table 4. This problem is a 0–1 integer programming problem and can be described as

$$\begin{array}{lll} \text{Minimize} & \left\{ \max\limits_{i:x_i=1} \sum\limits_{j:j\neq i} d_{ij} x_j - \min\limits_{i:x_i=1} \sum\limits_{j:j\neq i} d_{ij} x_j \right\} & \\ \text{st} & \sum\limits_{i=1}^{n} x_i = m & \\ & x_i \in \{0,1\} & i = 1,\ldots,n. \end{array} \tag{8}$$

Related Models

Given a graph $G = (V, E)$ where V is a set of n nodes and E is a set of edges, let $w_i \geq 0$ be the weight of node $i \in V$ and let c_{ij} be the benefit of edge $(i, j) \in E$. The *Capacitated Clustering Problem* (CCP) consists in partitioning V into p clusters in such a way that the sum of the weights of the elements in each cluster is within some integer capacity limits, L and U, and the sum of the benefits between the pairs of elements in the same cluster is maximized. The CCP can be formulated as a quadratic integer program with binary variables x_{ik} that take the value of 1 if element i is in cluster k and 0 otherwise:

$$\begin{array}{lll} \text{Maximize} & \sum\limits_{k=1}^{p} \sum\limits_{i=1}^{n-1} \sum\limits_{j>i}^{n} c_{ij} x_{ik} x_{jk} & \\ \text{st} & \sum\limits_{k=1}^{p} x_{ik} = 1 & i = 1,\ldots,n \\ & L \leq \sum\limits_{i=1}^{n} w_i \, x_{ik} \leq U & k = 1,\ldots,p \\ & x_{ik} \in \{0,1\} & i = 1,\ldots,n; k = 1,\ldots,p \end{array} \tag{9}$$

The objective function adds the total benefit of all pairs of elements that belong to the same cluster. The first set of constraints forces the assignment of each element to a cluster. The second set of constraints forces the sum of the weights of the pairs

of elements in the same cluster to be between L and U. The Maximally Diverse Grouping Problem (MDGP) consists in grouping a set of elements into p mutually disjoint groups in such a way that the diversity among the elements in each group is maximized, as described in [19]. The diversity among the elements in a group is calculated as the sum of the individual distance between each pair of elements. The objective of the problem is to maximize the overall diversity, i.e., the sum of the diversity of all groups, when the size of each group is within a specified range. Clearly, the MDGP is a special case of the CCP for which $w_i = 1$ for all node i, and the distance between each pair of nodes (i, j) is the benefit c_{ij}. Therefore, from the CCP formulation above, the MDGP can be formulated as

$$\begin{aligned} \text{Maximize} \quad & \sum_{k=1}^{p} \sum_{i=1}^{i=n-1} \sum_{j>i}^{n} c_{ij} x_{ik} x_{jk} & & \\ \text{st} \quad & \sum_{k=1}^{p} x_{ik} = 1 & & i = 1, \ldots, n \\ & L \leq \sum_{i=1}^{n} x_{ik} \leq U & & k = 1, \ldots, p \\ & x_{ik} \in \{0, 1\} & & i = 1, \ldots, n; k = 1, \ldots, p \end{aligned} \tag{10}$$

The MDGP is called the k-partition problem in [16] and the equitable partition problem in [38].

Correlation Between Models

In [46], the authors illustrated by the example on Table 1 that the correlation between the values of the solutions in Max-Sum and Max-Min problems can be relatively low.

Let us consider that we have seven elements of which we need to select five. Furthermore, the distances between each pair of elements are given by the matrix of Table 1. For such a small example, we can enumerate all the possible solutions (selections of elements) and compute for each one the value of the Max-Sum DP and the value of the Max-Min DP. The correlation between both objective functions

Table 1 Distance matrix of an instance with seven elements

	1	2	3	4	5	6	7
1	–	4.6	6.2	2.1	3.5	3.6	4.4
2	4.6	–	6.6	7.1	8.2	2.4	5.3
3	6.2	6.6	–	2.1	3.5	3.6	4.4
4	2.1	7.1	2.1	–	5.5	1.1	2.3
5	3.5	8.2	3.5	5.5	–	6.4	3.4
6	3.6	2.4	3.6	1.1	6.4	–	5.4
7	4.4	5.3	4.4	2.3	3.4	5.4	–

Table 2 Correlation coefficients among the mean results in the 30 instances, with $d_{ij} \in U[0, 20]$, $n = 20, m = 5$

Mean value				
	Max-Sum	Max-Min	Max-MinSum	Min-Diff
Max-Sum	1	0.60*	0.96*	−0.17
Max-Min	0.60*	1	0.73*	−0.63*
Max-MinSum	0.96*	0.73	1	−0.44
Min-Diff	−0.17	−0.63*	−0.44	1

*Correlation is significant at the 0.01 level

Table 3 Correlation coefficients among the best results in the 30 instances, with $d_{ij} \in U[0, 20]$, $n = 20, m = 5$

Best value				
	Max-Sum	Max-Min	Max-MinSum	Min-Diff
Max-Sum	1	0.78*	0.90*	0.07
Max-Min	0.78*	1	0.77*	0.03
Max-MinSum	0.90*	0.77*	1	−0.07
Min-Diff	0.07	0.03	−0.07	1

*Correlation is significant at the 0.01 level

is 0.61, which can be considered relatively low. Therefore, we should not expect a method for one of these problems to obtain good solutions for the other one.

We extend now that analysis by including the Max-MinSum, Min-Diff, Max-Sum, and Max-Min models. We do not include the Max-Mean models because they do not set the number of elements to select as the others. We generate 30 instances with 20 elements from which 5 have to be selected. Distances are randomly generated according to a uniform distribution in the range $[0, 20]$. For each instance we enumerate all its solutions (selections of $five$ elements) and compute the $four$ objective values (Max-Sum, Max-Min, Max-MinSum, and Min-Diff). Then, we compute two values per instance and objective, the mean and best values across all the solutions. Tables 2 and 3 show the correlations obtained from these two values, respectively.

There is no significant relation between the Min-Diff model solutions and the rest of models. Despite obtaining a correlation coefficient of −0.63 in the mean values between Min-Diff and Max-Min, this correlation is very low in the best case. So, this problem can be considered different enough to the rest of models. The largest correlation coefficient is obtained between Max-Sum and Max-MinSum. The correlation coefficient of 0.78 between Max-Sum and Max-Min best value solutions is also remarkable. Although these correlation values seem large enough, we have empirically found that a method developed for one model problem does not necessarily perform well on another or vice versa.

Diversity, Dispersion, and Equity Examples

The terms diversity, dispersion, and equity can be found in many optimization problems in the context of selection. While the first two are indistinguishable, the term equity seems to have a subtle difference with them. On the other hand, in most of the papers, these tree terms are used with certain ambiguity.

The problem is that the term diversity itself is not properly defined, and therefore it results in different interpretations. In any case, we can say that it is associated with a global characteristic of the selected subset and should be evaluated according to the distances d_{ij} considered.

The growing interest of leading with diversity resulted in the last years in an effort to study the management of equity, i.e., organizations are becoming increasingly interested in ensuring an equitable treatment of individuals or institutions. Generally speaking, equity is synonymous with fairness, objectivity, or impartiality. Many authors, such as French in [18], argue that equity is relative to justice, such as the distribution of resources or public service infrastructure.

In graphical terms, when we select a diverse subset from a given set, we expect them to be scattered. In other words, we don't expect them to be concentrated in a small region. In this section, we graphically show the solutions obtained when solving different models (Max-Sum, Max-Min, Max-MinSum, and Min-Diff), in an attempt to disclose their characteristics. Given a set $V = \{1, 2, \cdots, n\}$, an inter-element relationship d_{ij} defined between each pair of elements, and a subset $M \subseteq V$, we denote the diversity of M as div(M). In the example shown in Fig. 1, inter-element distances are calculated using the Euclidean metric; the results were obtained with mathematical programming models formulated in section "Mathematical Models and Formulation" and obtaining the exact solution with CPLEX 2.

If we are looking for solutions scattered in the set of elements, we might say that objects selected in Fig. 1b, obtained with the Max-Min model, are better distributed than those selected in the other models in Fig. 1 (see section "Mathematical Models and Formulation" for more details in these models). In this figure the elements with red square icons are the diverse set selected; the other ones are the unselected elements. In the Max-Sum and Max-MinSum models (Fig. 1a, c, respectively), the selected elements are located in the outer part of the graphic, which seems to indicate those that are disperse.

Notice that in this example, we are using the Euclidean metric, and therefore our interpretation is biased by our geometric conception. Particularly, this interpretation is no longer valid for a general measure as defined in Eq. (2). Then, a more general interpretation of diversity implies to abandon this intuitive and geometric conception, and establish diversity from another point of view, forgetting the Euclidean approach. Thus, the problem of measuring diversity in a given set is not unique and gives rise to different problems depending on the metric used. We have identified five different measures to compute the diversity, equity, or dispersion in M. Table 4 summarizes them.

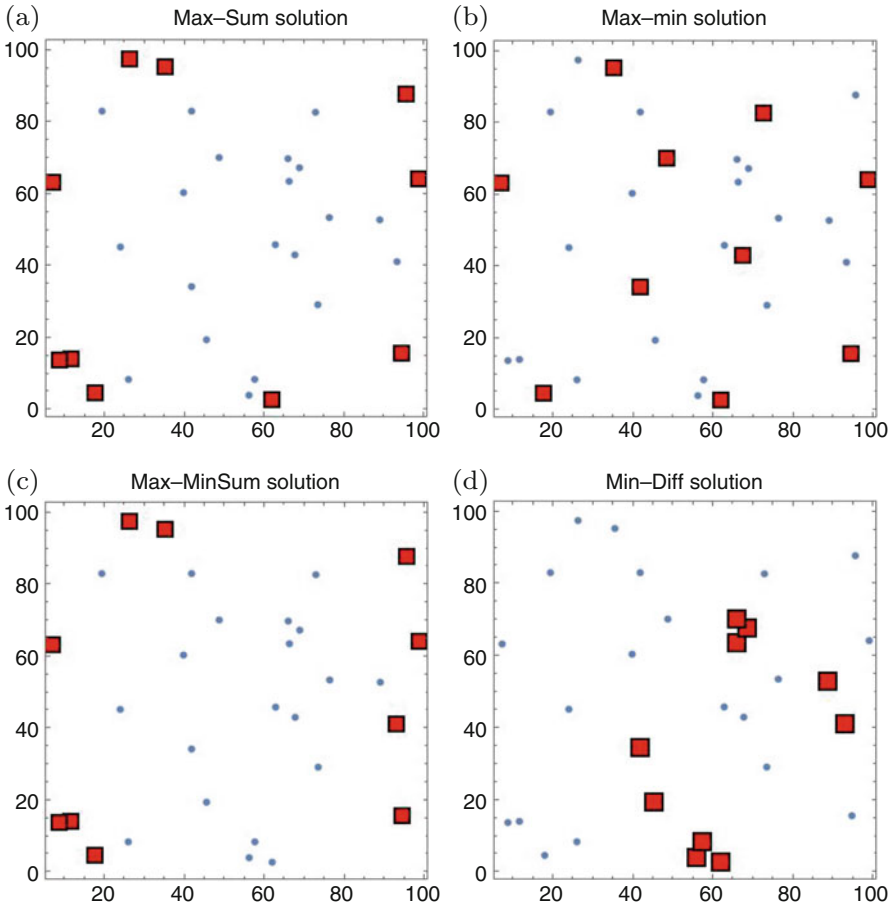


Fig. 1 Comparison of different models of a set of 30 elements with 10 selected elements

Consider two examples in the rectangle $[0, 10] \times [0, 10]$ to illustrate the difference among the different dispersion measures based on the Euclidean distance. The position of the elements is obtained at random in both examples. In the first example, Example 1, we consider $n = 12$ elements, whose position is represented in Fig. 2.

The aim of this example is to observe the value of the different dispersion measures. Consider two different subsets of selected elements. Solution A contains the elements $\{2, 3, 7, 9\}$ and solution B contains the elements $\{1, 3, 8, 10\}$. In Fig. 3, we show in big squares in red the selected elements of these two solutions.

The numerical value of the different diversity measures is shown in Table 5. We can realize that the Max-Min, Max-MinSum, and Min-Diff models give better

Table 4 Description of the different diversity measures

Measure	Mathematical function	Description
Sum	$\sum_{i<j,i,j\in M} d_{ij}$	This measure may address diversification among selected elements to distance
Min	$\min_{i<j,i,j\in M} d_{ij}$	Focus on the minimum distance among the selected elements
Mean	$\frac{\sum_{i<j,i,j\in M} d_{ij}}{\lvert M\rvert}$	Related to the sum measure is an average equity measure
MinSum	$\min_{i\in M} \sum_{j\in M, j\neq i} d_{ij}$	This measure considers the minimum aggregate dispersion among elements
Diff	$\max_{i\in M} \sum_{j\in M, j\neq i} d_{ij} - \min_{i\in M} \sum_{j\in M, j\neq i} d_{ij}$	This measure can be understood as the difference between the largest and smallest values of the dispersion sum

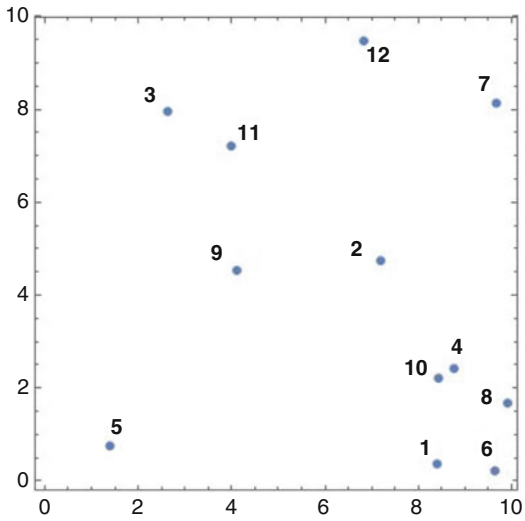


Fig. 2 Position of the elements in Example 1

results in solution A than in B. However, solution B is better than A in the other models. According to this, one could think that the five models considered can be grouped into two categories. However, the next example illustrates that it is not always in this way.

Consider now a second example, Example 2, with 500 elements, where we have to select 80 elements. As above, we consider two solutions with different subset of elements (see Fig. 4), and we show the numerical measures of each model in Table 6.

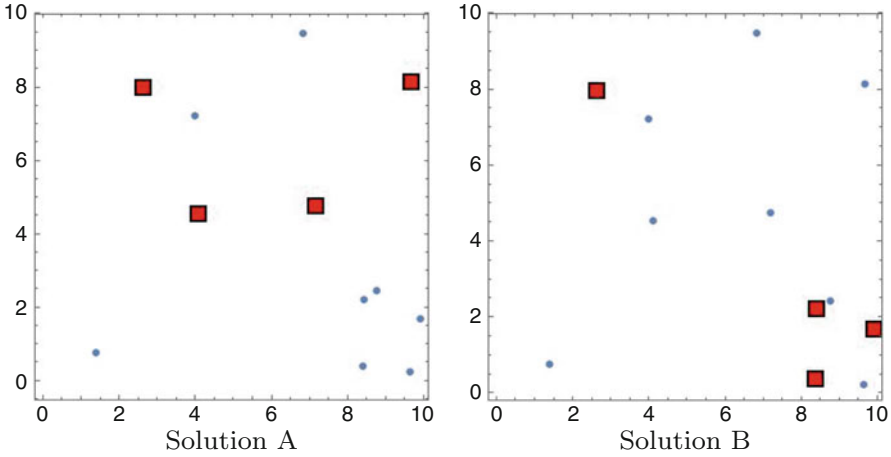


Fig. 3 The two subsets of selected elements in Example 1

Table 5 Diversity measures for the two subsets of selected elements in Example 1

Model	Max-Sum	Max-Min	Max-Mean	Max-MinSum	Min-Diff
Solution *A*	30.26	3.09	7.56	12.87	4.99
Solution *B*	32.10	1.70	8.10	12.30	15.30

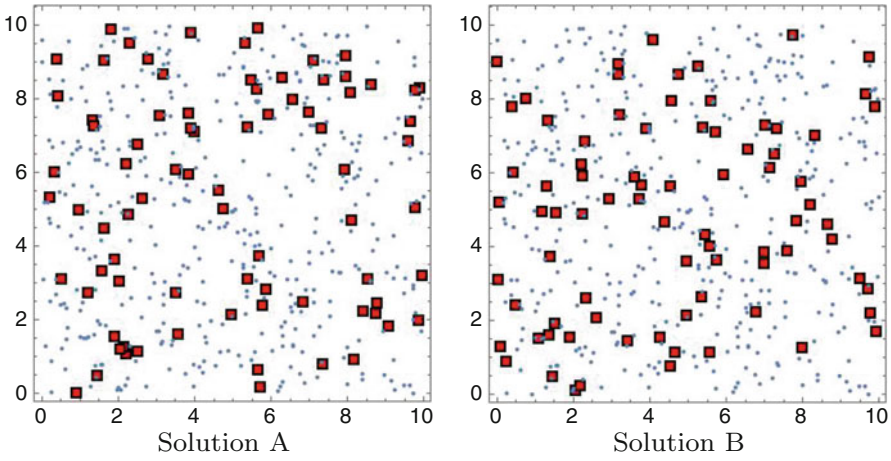


Fig. 4 The two subsets of selected elements in Example 2

In this example, *A* exhibits better diversity measures than *B* in all cases except in the Max-Min model. These results seem to confirm that the diversity depends on the model we are applying. Therefore, as it is customary when solving an optimization problem, we first have to model it, which in our case means to select the diversity model that better reflects our goal.

Table 6 Diversity measures for the two subsets of selected elements in Example 2

Model	Max-Sum	Max-Min	Max-Mean	Max-MinSum	Min-Diff
Solution A	17021.0	0.123	212.76	315.73	264.43
Solution B	15995.9	0.169	199.95	293.51	287.97

Finally, it is easy to compute that the complexity of evaluation of the diversity is $\mathcal{O}(m^2)$, where m is the number of selected elements, in the case of Max-Sum, Max-Min, and Max-Mean; however, in Max-MinSum and Min-Diff models, the complexity is $\mathcal{O}(m^3)$.

Metaheuristics

We can find a large number of heuristic and metaheuristic algorithms to solve all kinds of diversity problems in the literature. However, the research has been quite dispersed, and there are very few review articles, probably due to the different names used for these problems. So, we can only find incomplete comparisons among the methods proposed to solve these models. The absence of a precise definition of what diversity means entails the absence of a framework that integrates all these models, and each one has arisen independently. In this section, we review the main heuristic and metaheuristic proposed for the Max-Sum, Max-Min, Max-Mean, Max-MinSum, and Min-Diff models. In [2], the authors present an extensive computational experiments to compare 10 heuristics and 20 metaheuristics for the Max-Sum model.

The Max-Min Model

Two construction methods based on *greedy randomized adaptive search procedure* (GRASP) and different approaches to hybridize them with path relinking methodology are proposed in [46]. The first GRASP construction consists of evaluating each candidate element with a greedy function in order to identify the best elements and add them to the so-called restricted candidate list (RCL). Then, the next element to be included in the partial solution is randomly chosen from RCL. On the other hand, the other GRASP construction method is based on an alternative scheme introduced in [45], in which the RCL is constructed totally at random from elements in the candidate list (CL). Therefore, RCL can be thought in this latter case of as a random sample of CL of a preestablished size. Then, the greedy function is applied to evaluate all the elements in RCL in order to choose the best. In this alternative design, the sequence of applying randomness followed by greediness is inverted. The local search mechanism implements the mildest ascent technique and the path relinking methodology. The second strategy is more efficient than the classic one in this model. The same behavior of GRASP scheme is observed in other models of

maximum diversity as in [33]. In [42], the authors proposed a simple and effective tabu search algorithm based on drop and add moves. The local search consists of switch (or swap) a selected element with an unselected element, if it is better than the current solution. The stopping criterion is met when there is no improvement in a number of iterations. A short-term tabu search memory is added to this local search by declaring tabu the most recent moves.

Della Groce et al. proposed in [10] an *Iterated Local Search* heuristic algorithm based on reformulating the Max-Min Dispersion Problem as a dichotomic search, where at each iteration of the search a clique decision problem has to be solved.

The Max-Sum Model

The algorithm described in [29] is based on the memetic approach, which starts generating the initial population using a greedy randomized procedure. To ensure the diversity of this initial set of solutions, according to the classical binary pattern, each solution of size n contains m bits equals to 1. Two parents are randomly chosen with the restriction that none of them are used more than once in each generation. In the next iteration, the new parents are performed by selecting the best previous parents and the best generated children, without repetition. After that, $p/2$ children are obtained by using a crossover and mutation procedure, where p denotes the number of parents. Uniform crossover generates only one child, in which the chromosomes are selected from one parent with probability lower or equal than 0.5 and the second parent with a probability higher than 0.5. Mutation changes the offspring by flipping bits from 1 to 0 or from 0 to 1. Mutation can occur at each bit position in the string with some probability. After each crossover and mutation, a repair process is applied in order to ensure feasibility, i.e., the chromosome contains m bits which equals to 1.

A local search process based on $k - flip$ $(k - opt)$ movements is applied to the feasible solutions in the memetic algorithm. These movements consists of two loops. The inner one generates solutions by flipping the elements of a solution x, as long as it does not find the best solution or it performs t repetitions. In an outer loop, the best solution found is evaluated. An overview of the algorithm allows to say that it works well, although execution times are higher than other algorithms.

In [17], the authors propose a memetic self-adaptive evolution strategy. Evolution strategies are a class of evolutionary algorithms primarily dependent on mutation. The authors compare their algorithm with the algorithms of the state of the art: a variable neighborhood search (VNS) [5] method, a learnable tabu search (LTS) [47], and an iterated tabu search (ITS) [40]. The evolution strategy provides satisfactory results when the goal is not to achieve the optimal results; in other cases, simple heuristics may be preferred.

Some metaheuristics for this problem are compared in [2] to identify additional features that provide the best improvement. All of them are based on a common tabu search module [12, 23]: an exploring tabu search [9], a variable neighborhood search [26], a scatter search [32], and a random restart algorithm.

All these algorithms are competitive with the state of art. The best performing algorithms are the ones based on random restart or a nearly random restart in which the elements of the current best-known solution are forbidden. The authors suggest that, given the simple structure of the MDP, simple algorithms might work better. The authors consider the best algorithm available in literature is the one proposed in [5]. In that paper, the authors describe a study to solve the problem "heaviest k-subgraph" (HSP); the Max-Sum problem is a special case of HSP. The computational experiments are performed by using the instances of the Max-Sum problem; the aim is to test the efficiency of the proposed algorithms. The authors propose construction methods based on GRASP, greedy, scatter search, tabu search, and a multi-boot method. The variable neighborhood search method used in the article is equivalent to that proposed in [27]. This method is more efficient for the mechanism of variable neighborhoods adopted, compared with state-of-the-art heuristics that use specialized hybrid versions of GRASP and tabu search. The conclusion-based method VNS can be seen as advantageous for both the problem HSP and Max-Sum problem, providing better quality results but more expensive computationally.

A generalization of Max-Sum problem is the Capacitated Clustering Problem (CCP), which consists of forming a specified number of clusters or groups from a set of elements in such a way that the sum of the weights of the elements in each cluster is within some capacity limits, and the sum of the benefits between the pairs of elements in the same cluster is maximized. The Maximally Diverse Grouping Problem (MDGP) is a special case of the CCP in which all the elements have a weight of one unit. The MDGP is also known as the k-partition problem in [16] or the equitable partition problem in [38]. In [19], the authors propose a heuristic procedure for the MDGP based on the tabu search with strategic oscillation (TS-SO). The methodology starts by using a greedy construction method for the initial solution; then the search neighborhood in TS-SO consists of all node insertions and swaps. Finally, the third phase of TS-SO, the strategic oscillation (SO) phase, explores solutions for which the group cardinality restrictions may be violated. In particular, the method applies the neighborhood search, but the number of the elements in a group is allowed to be outside the specified limits by a certain amount so. To create the oscillation pattern, the value of so is reset to one after every successful application of the improvement method. Recently, in [35], the authors propose a tabu search and several GRASP variants to find high-quality solutions to this NP-hard problem. These variants are based on several neighborhoods, including a new one, in which they implement a one-for-two swapping strategy. In this paper, the authors also hybridize both methodologies to achieve improved outcomes. The algorithms are compared with the state of the art:

- In [11], the authors proposed a greedy randomized adaptive search procedure (GRASP) with variable neighborhood search (VNS) that, according to their computational study, outperforms previous approaches. An interesting application arises in the context of facility planners at mail processing and distribution centers within the US Postal Service.

- In [37], the authors proposed a GRASP with path relinking for the handover minimization in the context of mobility networks. As a mobile transceiver moves between areas, it may need to connect over time to several base stations. The transfer of connection from one base station to another is called a handover. Each base is connected to one radio network controller (RNC), which controls many of its operations, including its traffic and handover. Handovers between base stations connected to different RNCs tend to fail. The handover minimization problem is to assign base stations to RNCs such that RNC capacity is not violated, and the number of handovers between base stations connected to different RNCs is minimized. The set of base stations assigned to a RNC can be viewed as a cluster, and the minimization of handovers between different clusters is equivalent to the maximization of handovers within the same cluster. Therefore, this problem is equivalent to the CCP, and in [35], they also compare their new method with the previous heuristic proposed by [37].

The Max-Mean Model

Two recent papers proposed heuristic methodologies to solve the Max-Mean model. The first one is [33], where the authors test a GRASP constructive algorithm based on a nonstandard combination of greediness and randomization. The greedy function considers the mathematical properties of this problem, based on the quasi-concave shape of the partial solutions. Then it is applied as a local search strategy based on the variable neighborhood descent methodology, which includes three different neighborhoods, and a path relinking post-processing. This latter method is based on a measure to control the diversity in the search process. The authors compare their algorithm with the best algorithms for the Max-Sum problem adapted to this particular problem. On the other hand, in [7] the authors propose a method to solve the Max-Mean problem based on a greedy construction phase and a tabu search (TS) method. First of all, an initial solution is constructed by two greedy algorithms for generating good initial solutions to the tabu search procedure. After that, a two-phase tabu search method is applied. The first stage (called short-term TS) is devoted to the intensification of the search. The second stage (called long-term TS) is focused on a diversification strategy to explore new regions of the solution space. Additionally, both begin from the current solution, and after termination, they return the overall best solution and their current solution. Note that the current and the best overall are not usually the same solution since these phases usually deteriorate the current solution in order to escape from its basin of attraction. The search terminates after a maximum number of iterations have elapsed without improving the overall best solution.

The Max-MinSum Model

The Max-MinSum DP is a difficult combinatorial optimization problem and a perfect platform to study the effectiveness of search mechanisms. The computational

experiments presented in [43] include results of applying a GRASP methodology to the Max-MinSum dispersion problem. In their GRASP implementation, the authors define M_k as a partial solution with k selected elements ($1 \leq k < m$). Each construction phase of GRASP starts by randomly selecting an element in order to initialize M_1. Then, in each iteration, the method builds a candidate list (CL) that consists of all the unassigned elements. For each element i in CL, the method computes a marginal contribution of the element toward the objective function associated with M_{k+1}:

$$\triangle f^k(i) = \min \left\{ \min_{j \in M_k} \{s^k(j) + d_{ij}, s^k(i)\} \right\} - f(M_k) \tag{11}$$

where $f(M_k)$ is the value of the objective function corresponding to the partial solution M_k and

$$s^k(i) = \sum_{j \in M_k} d_{ij} \tag{12}$$

Elements in CL are ordered according to the marginal contributions, that is, from largest to smallest $\triangle f^k$ values. Then, a restricted candidate list (RCL) is constructed with the top α elements in CL. The value of α is selected at random, in each construction step, from a uniform distribution with parameters 1 and $|CL|$. The element to be included in the partial solution M_{k+1} is randomly chosen from RCL. An improvement phase is executed after a solution M has been constructed. The improvement phase consists of an exchange mechanism in which a selected element i is replaced with an unselected element j. The method randomly selects both elements and exchanges them if and only if the objective function value of the resulting solution improves; otherwise, the (i, j) pair is discarded. The improvement phase finishes after 100 iterations without any improvement.

Finally, in [34] the authors test six different GRASP variants in which they considered different ways to generate and improve a solution. For the improvement phase, the authors consider two different designs. Both designs are pure local searches in the sense that they terminate when the exploration of the entire neighborhood of the current solution does not yield a move that improves the objective function value. Given a solution M, the first approach, $IM1$, consists of the exhaustive exploration of the neighborhood defined by all (i, j) exchanges. An exchange results in a neighbor solution M'. Let M^* be the neighbor solution M' with the best objective function value. Then if $f(M^*) > f(M)$, then the search moves to M^* and the neighborhood is explored. Otherwise, the improvement method ends. The second design, $IM2$, instead of exploring all possible exchanges, considers the contribution of the selected elements as well as the potential contribution of the unselected elements to create a priority list. In the best construction phase proposed in [34], called $CM3$, instead of constructing solutions by adding elements to a partial solution with too few elements, a feasible solution can be found by

deleting elements from an unfeasible solution with too many elements. The authors added a new ingredient in the construction phase, the notion introduced by Glover in [21] called strategic oscillation (SO). The authors focus on the construction phase of GRASP and oscillate around the feasibility boundary defined by the constraint

$$\sum_{i=1}^{n} x_i = m.$$

The constraint indicates that a feasible solution must have m elements. Therefore, the boundary around which the authors define their SO separates unfeasible M_k solutions into those with $k < m$ and those with $k > m$. The single-sided oscillation patterns may be seen as intensification mechanisms, while the two-sided oscillation pattern favors search diversification. Finally, the authors performed several experiments with instances previously used in the literature. The experiments show that the double-sided strategic oscillation with the $CM3$ constructive method and the $IM2$ improvement method provides the best outcomes overall. Moreover, the results indicate that the proposed hybrid heuristic compares favorably to an existing specialized procedure and a general-purpose optimizer (LocalSolver, which is available at http://www.localsolver.com).

The Min-Diff Model

The most recent papers on the Min-Diff model are [3, 13, 36]. In [3], the authors proposed a two-phase heuristic algorithm for this problem. The constructive phase exploits the relation between this problem and the Maximum Clique Problem (MCP); however, the improvement phase is a tabu search procedure. The two phases are repeated iteratively adding a simple diversification mechanism in the constructive phase. This work also shows a way to reduce the computational complexity at each iteration, from a quadratic evaluation to a linear update mechanism.

In [13], the authors proposed several new hybrid heuristics. The best one consists of a GRASP with sampled greedy construction with variable neighborhood search for local improvement. The authors considered eight constructive procedures; four local search procedures, including one based on VNS; and four path relinking strategies. In [22], Glover introduced the concept of exterior path relinking, or path separation. The authors in [13] used this variant of the more common interior path relinking for the first time. Extensive computational experiments on 190 instances from the literature demonstrated the competitiveness of the algorithm. Not only was it able to outperform the GRASP heuristic of [43] and find optimal solutions to all but one of the instances that CPLEX is able to solve, but it also improved the CPLEX upper bound on all but one of the instances that CPLEX failed to solve.

Finally, in [36], the authors propose a basic variable neighborhood search (VNS) heuristic, limited to interchange neighborhood structures both in intensification and

diversification phases. The authors show that VNS is faster than an efficient and effective method that combines GRASP, variable neighborhood search, and exterior path relinking metaheuristics. As a conclusion, the authors state that their results will be a reminder of what the original goal of heuristics is: to create an efficient and effective algorithm so to be as simple as possible or, in short, *less is more*.

Conclusions

In this chapter we reviewed two diversity and three equity models, and we have also considered two related models. We identified previous mathematical models and heuristic methods for these seven variants. Most of them are based on constructive and local search-based methods. This is the case of the GRASP and tabu search methodologies, which have been customized for some of this models with excellent results. Surprisingly, evolutionary methods, such as the popular genetic algorithms, have been considered in just a few cases. We noted that although the problems considered here are related, all previous efforts are devoted to tackle very specific models, and no generic solver has been proposed to deal with the entire family. Additionally, we have studied the solution structure obtained with the different models, and we have identified some differences and similarities between them.

We can conclude that the current state-of-the-art models and solution algorithms for the diversity and equity models seem not really satisfactory. Whereas very good feasible solutions can be found for large problems using the comprehensive toolbox of heuristics, fairly small problems can still be difficult for exact algorithms and cannot be solved to optimality in reasonable time. Additionally, the development of generic solvers that deal with several models would be of interest. So there is definite need for further research on algorithms, mainly for exact algorithms but also for heuristics.

Cross-References

▸ Tabu Search
▸ Theory of Local Search
▸ Variable Neighborhood Descent

Acknowledgments This research has been partially supported by the Ministerio de Economía y Competitividad of Spain (Grant Refs. TIN2012-35632-C02 and TIN2015-65460-C2), the Generalitat Valenciana (ACOMP/2014/A/241 and Prometeo 2013/049), and the University of Valencia (UV-INV-PRECOMP13-115334).

References

1. Aǧca S, Eksioglu B, Ghosh JB (2000) Lagrangian solution of maximum dispersion problems. Nav Res Logist 47:97–114
2. Aringhieri R, Cordone R (2011) Comparing local search metaheuristics for the maximum diversity problem. J Oper Res Soc 62:266–280
3. Aringhieri R, Cordone R, Grosso A (2015) Construction and improvement algorithms for dispersion problems. Eur J Oper Res 242:21–33
4. Asada H, Toyama F, Shoji K, Miyamichi J (2010) Genetic local search with adaptative crossover probability and its application to maximum diversity problem. IEEJ Trans Electron Inf Syst 130:529–520
5. Brimberg J, Mladenovic N, Urosevic D, Ngai E (2009) Variable neighborhood search for the heaviest k-subgraph. Comput Oper Res 36:2885–2891
6. Campos V, Laguna M, Martí R (2005) Context-independient scatter and tabu search for permutation problems. INFORMS J Comput 17(1):111–122
7. Carrasco R, Pham A, Gallego M, Gortázar F, Martí R, Duarte A (2015) Tabu search for the max-mean dispersion problem, Knowledge based systems 85:256–264
8. Castillo O, Melin P, Gamez E, Kreinovich V, Kosheleva O (2009) Intelligence techniques are needed to further enhance the advantage with diversity in problem solving. In: IEEE workshop hybrid intelligent models and applications (HIMA'09), Nashville, pp 48–55
9. Dell'Amico M, Trubian M (1998) Solution of large weighted equicut problems. Eur J Oper Res 106:500–521
10. Della Groce F, Grosso A, Locatelli M (2009) A heuristic approach for the max-min diversity problem based on max-clique. Comput Oper Res 36(8):2429–2433
11. Deng Y, Bard JF (2011) A reactive GRASP with path relinking for capacitated clustering. J Heuristics 17:119–152
12. Duarte A, Martí R (2007) Tabu search and GRASP for the maximum diversity problem. Eur J Oper Res 178:71–84
13. Duarte A, Sánchez-Oro J, Resende M, Glover F, Martí R (2015) GRASP with exterior path relinking for differential dispersion minimization. Inf Sci 296(1):46–60
14. Erkut E (1990) The discrete p-dispersion problem. Eur J Oper Res 46:48–60
15. Erkut E, Neuman S (1989) Analytical models for locating undesirable facilities. Eur J Oper Res 40:275–291
16. Feo T, Goldschmidt O, Khellaf M (1992) One-half approximation algorithms for the k-partition problem. Oper Res 40:S170–S173
17. Freitas ARR, Guimarães FG, Pedrosa Silca RC, Souza MJF (2014) Memetic self-adaptive evolution strategies applied to the maximum diversity problem. Optim Lett 8(2): 705–714
18. French E (2005) The importance of strategic change in achieving equity in diversity. Strateg Change 14:35–44. Wiley InterScience
19. Gallego M, Laguna M, Martí R, Duarte A (2013) Tabu search with strategic oscillation for the maximally diverse grouping problem. J Oper Res Soc 64(5):724-734
20. Ghosh JB (1996) Computational aspects of the maximum diversity problem. Oper Res Lett 19:175–181
21. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(1):156–166
22. Glover F (2014) Exterior path relinking for zero-one optimization. Int J Appl Metaheuristic Comput 5(3):1–8
23. Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers, Boston
24. Glover F, Kuo CC, Dhir KS (1995) A discrete optimization model for preserving biological diversity. Appl Math Model 19:696–701

25. Glover F, Kuo CC, Dhir KS (1998) Heuristic algorithms for the maximum diversity problem. J Inf Optim Sci 19(1):109–132
26. Hansen P, Mladenovic N (2001) Variable neighborhood search: principles and aplications. Eur J Opns Res 130:449–467
27. Hansen P, Mladenovic N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook in metaheuristics. Kluwer Academic Publishers, Boston
28. Hassin R, Rubinstein S, Tamir A (1997) Approximation algorithms for maximum dispersion. Oper Res Lett 21:133–137
29. Katayama K, Narihisa H (2005) An evolutionary approach for the maximum diversity problem. In: Recent advances in memetic algorithms, vol 166. Springer, Berlin/New York, pp 31–47
30. Kincard RK (1992) Good solutions to discrete noxious location problems via metaheuristics. Ann Oper Res 40:265–281
31. Kuo CC, Glover F, Dhir KS (1993) Analyzing and modeling the maximum diversity problem by zero-one programming. Decis Sci 24:1171–1185
32. Laguna M, Marí R (2003) Scatter search: methodology and implementations in C. Kluwer Academic Publishers, Dordrecht
33. Martí R, Sandoya F (2013) GRASP and path relinking for the equitable dispersion problem. Comput Oper Res 40:3091–3099
34. Martínez-Gavara A, Campos V, Laguna M, Martí R (2016) Heuristic solution approaches for the maximum MinSum dispersion problem. J Glob Optim https://doi.org/10.1007/s10898-016-0429-1
35. Martínez-Gavara A, Campos V, Gallego M, Laguna M, Martí R (2015) Tabu Search and GRASP for the Capacited Clustering Problem. Comput Optim Appl 62:589–607
36. Mladenović N, Todosijević R, Urosević D (2016) Less is more: basic variable neighborhood search for minimum differential dispersion problem. Inf Sci 326:160171
37. Morán-Mirabal LF, González-Velarde JL, Resende MGC, Silva RMA (2013) Randomized heuristics for handover minimization in mobility networks. J Heuristics 19:845–880
38. O'Brien FA, Mingers J (1995) The equitable partitioning problem: a heuristic algorithm applied to the allocation of university student accommodation. Warwick Business School, Research Paper no. 187
39. Page SE (2007) The difference: how the power of diversity creates better groups, firms, schools, and societies. Princeton University Press, Princeton
40. Palubeckis G (2007) Iterated tabu search for the maximum diversity problem. Appl Math Comput 189(1):371–383
41. Pisinger D (2006) Upper bounds and exact algorithms for p-dispersion problems. Comput Oper Res 33:1380–1398
42. Porumbel D, Hao J, Glover F (2011) A simple and effective algorithm for the MaxMin diversity problem. Ann Oper Res 186(1):275–293
43. Prokopyev OA, Kong N, Martinez-Torres DL (2009) The equitable dispersion problem. Eur J Oper Res 197:59–67
44. Ravi SS, Rosenkrantz DJ, Tayi GK (1994) Heuristic and special case algorithms for dispersion problems. Oper Res 42:299–310
45. Resende MGC, Werneck R (2004) A hybrid heuristic for the p-median problem. J Heuristics 10(1):59–88
46. Resende MGC, Martí R, Gallego M, Duarte A (2010) GRASP and path relinking for the max−min diversity problem. Comput Oper Res 37(3):498–508
47. Wang J, Zhou Y, Cai Y, Yin J (2012) Learnable tabu search guided by estimation of distribution for maximum diversity problems. Soft Comput 16(4):711–728

Evolutionary Algorithms for the Inverse Protein Folding Problem

33

Sune S. Nielsen, Grégoire Danoy, Wiktor Jurkowski, Roland Krause, Reinhard Schneider, El-Ghazali Talbi, and Pascal Bouvry

Contents

Introduction . . . 1000
Amino Acids and Protein Structure . . . 1001
Inverted Protein Folding . . . 1003
Diversity Preservation as a Tool . . . 1003
Related Work . . . 1004
Protein Design . . . 1004
Multimodal Optimization and Niching . . . 1006
Problem Description . . . 1007
Sequence Identity . . . 1007
Problem Model . . . 1008
Secondary Structure Definition . . . 1008
Secondary Structure Estimation . . . 1009
Diversity Measure . . . 1010
Algorithm Design . . . 1010

S. S. Nielsen (✉) · G. Danoy · P. Bouvry
Computer Science and Communications (CSC) Research Unit, FSTC, University of Luxembourg, Luxembourg City, Luxembourg
e-mail: sune.nielsen@uni.lu; sune.nielsen.pro@gmail.com; gregoire.danoy@uni.lu; pascal.bouvry@uni.lu

W. Jurkowski
The Genome Analysis Centre (TGAC), Norwich Research Park, Norwich, UK
e-mail: wiktor.jurkowski@tgac.ac.uk

R. Krause · R. Schneider
Luxembourg Centre for Systems Biomedicine (LCSB), University of Luxembourg, Luxembourg City, Luxembourg
e-mail: roland.krause@uni.lu; reinhard.schneider@uni.lu

E.-G. Talbi
Université des sciences et technologies de Lille, INRIA Lille Nord Europe, Villeneuve d'Ascq, France
e-mail: el-ghazali.talbi@inria.fr; el-ghazali.talbi@lifl.fr

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_59

Removal of Doubles . . . 1011
Quantile Constraint . . . 1011
Algorithm Experiments . . . 1012
Protein Samples . . . 1012
Experimental Setup . . . 1012
Algorithm Results . . . 1013
Structure Validation . . . 1016
Primary and Secondary Structure Validation Results . . . 1016
Tertiary Structure Validation Results . . . 1018
Conclusion . . . 1020
Cross-References . . . 1021
References . . . 1022

Abstract

Protein structure prediction is an essential step in understanding the molecular mechanisms of living cells with widespread application in biotechnology and health. The inverse folding problem (IFP) of finding sequences that fold into a defined structure is in itself an important research problem at the heart of rational protein design. In this chapter, a multi-objective genetic algorithm (MOGA) using the diversity-as-objective (DAO) variant of multi-objectivization is presented, which optimizes the secondary structure similarity and the sequence diversity at the same time and hence searches deeper in the sequence solution space. To validate the final optimization results, a subset of the best sequences was selected for tertiary structure prediction. Comparing secondary structure annotation and tertiary structure of the predicted model to the original protein structure demonstrates that relying on fast approximation during the optimization process permits to obtain meaningful sequences.

Keywords

Genetic algorithm · Diversity preservation · Inverse folding problem

Introduction

The relation between the amino acid sequence of a protein and its three-dimensional structure is a principal research effort of structural biology. Obtaining the folded structure of a protein allows functional studies in silico and has given rise to the field of protein engineering.

Proteins are responsible for the majority of molecular functions in a cell. A simplified illustration of a real protein is provided in Fig. 1. Understanding protein folding has immense implications from health to biotechnology applications. Protein engineering in general aims at designing molecules with desired properties, and a method that allows to successfully design such molecules would find applications in a number of areas. For example, it could allow to design improved enzymes for biotechnology applications such as wastewater treatment or biomass production [7]

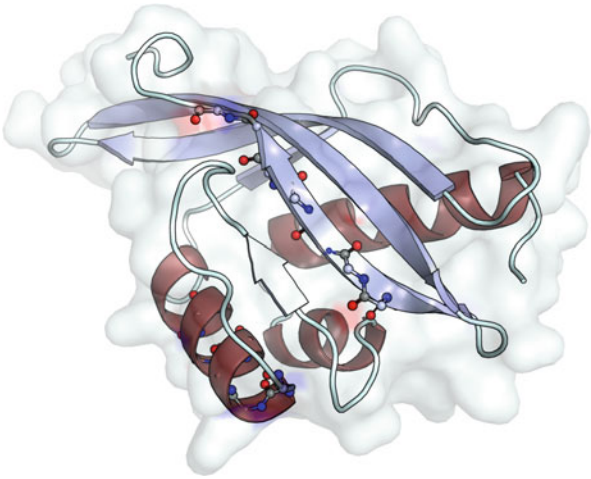


Fig. 1 Protein example *1OH0* with its surface shown semitransparent. *Helix* and *sheet* secondary structure segments are shown in dark red and light blue, respectively. Selected atoms are displayed for further clarification

or new antibodies specific toward already known targets, e.g., a given pathogen like HIV, by binding to its envelope spikes to neutralize the virus [19]. Since the advent of genome sequencing, all protein-coding genes of an organism can be obtained with ease, but structure prediction capabilities were only slightly improved over the last two decades and remain poor. If no homologous structure to a given sequence exists (the ab initio problem), finding the correct structure remains an essentially intractable, which hampers even the comparably easy task of classifying protein sequences into families.

Amino Acids and Protein Structure

A protein sequence is the code that describes the linear combination of any of the 20 common amino acids, also referred to as residues. The amino acid residues are basic organic building blocks consisting mainly of carbon (C), hydrogen (H), oxygen (O), and nitrogen (N) atoms. Common for all amino acids are their *amine* and *carboxylic acid* functional groups which bind through peptide bonds to form the protein backbone of $N - C_\alpha - C$ atoms as shown in Fig. 2. When ordered from left to right, as in the figure, the *amine* group, here represented by its nitrogen (N) atom for simplicity, is situated to the left of the amino acid, respectively, at the beginning of the chain. The side chains, noted as R_i, vary with each of the possible amino acids and can vary both in size and other properties, such as charge, acidity, and hydropathy. A typical protein sequence is 50–300 residues long. Due to the rotational freedom of the atom bonds and the molecular forces acting between the residues, it folds into one canonical three-dimensional structure. These intermolecular forces are the sum of a number of complex interaction forces largely depending on the mentioned properties of the residues, but also on the distance and orientation of interacting atoms and structures. In general the protein

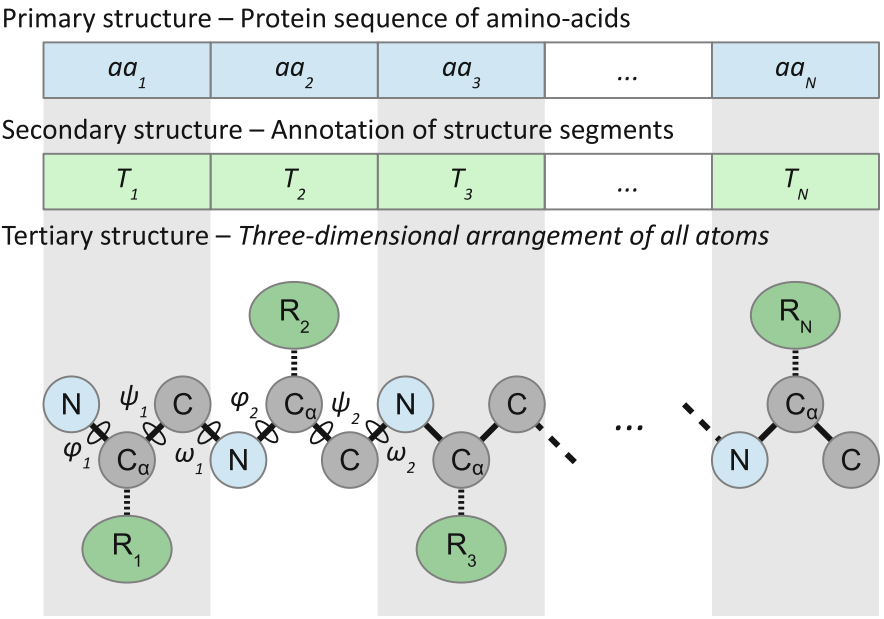


Fig. 2 Three levels of protein structure

structure will try to adapt a lower-energy configuration like a bolder that will roll down a mountain into the valley due to the gravitational force. In the case of proteins, such a more relaxed state corresponds to parts of the protein being either stacked or curled together referred to as *sheets* or *helices* as seen in Fig. 1. The remaining unstructured segments are commonly referred to as *loops* and serve as flexible connections between the other segments. The structure of a protein can be defined in different levels (see Fig. 2). The primary structure is the protein sequence of N amino acids $\{aa_i\}$ where $1 \leq i \leq N$ is the residue position. The secondary structure defines the organization of *helices*, *sheets*, and *loops* of the tertiary structure and can be expressed by a type $\{T_i\} \in \{H, E, L\}$ for each position i in the protein. If, for example, a protein consists of a *helix* and two *sheets*, its secondary structure would look like this: $\{L, L, H, H, H, H, L, E, E, L, E, E, L\}$. The tertiary structure completely describes the arrangement of all atoms of a protein in the three-dimensional space. The ensemble of three-dimensional positions of C_α atoms is commonly referred to as the alpha-trace which provides a rough residue type- and rotation-independent view of the protein configuration. Similar protein sequences generally obtain the same configuration or fold, but sequences not recognizable by similarity can nevertheless fold into 3D structures that are easily brought into congruence. Recommended reading for more in-depth information about proteins and their function in cells is the book by Alberts et al. [2].

Inverted Protein Folding

Conventional protein folding prediction research is concerned with finding or predicting the folded structure of a given amino acid sequence. As the problem is not solved, even to the present day, scientists have early on sought to simplify the task by solving the inverse problem. With the hierarchical definition of Fig. 2 in mind, the inverse folding problem (IFP) can be defined as follows: given a primary structure (protein sequences) and its corresponding tertiary structure, find alternative sequences that will result in the same tertiary structure. The inverted problem is thought of as a simplification because the structure is given, and sequence to structure compatibility becomes the main difficulty. When the structure is unknown (the ab initio case), the number of possible configuration solutions is enormous. A central part of any protein design process is to obtain, or come close to, a target tertiary structure with a certain degree of freedom in the choice of protein sequence. Hence solving the IFP would be a key to successfully engineer proteins. Furthermore, the IFP is of general scientific interest to study the size, shape, and characteristics of the sequence space that matches a given target structure.

Diversity Preservation as a Tool

In this chapter, the fact that matching secondary structures is a necessary, but not a sufficient condition for proteins to have the same tertiary structures, is exploited to reduce the IFP to its simplest formulation: given a protein's secondary structure and its corresponding protein sequence as input, find a set of highly dissimilar protein sequences that could result in the most similar secondary structure. The multi-objective genetic algorithm (MOGA) variant presented here is hence designed for maintaining high diversity, which in turn allows it to explore the decision space of sequences more efficiently and find better solutions than a conventional algorithm. This essentially makes the diversity preservation characteristic central in two aspects: (1) it increases the algorithm's performance in that it continuously pushes the exploration to new areas of the search space while (2) addressing the part of the problem statement of finding a set of protein sequences (i.e., problem solutions) that show large diversity among each other. The latter aspect is thoroughly covered in ▸ Chap. 32, "Diversity and Equity Models", though it should be noted that the representation of solutions and distance among them is different from this work.

An extended validation test is run predicting the final folded structure of as many as 300 generated sequences which are then analyzed in terms of secondary and tertiary structure. This test aims at answering the question of how well the target tertiary structure can be matched solely by taking the secondary structure into account.

The remainder of this chapter is organized as follows: in section "Related Work" the current work is situated in related literature; then a detailed description

of the problem and the biological background is introduced in section "Problem Description" and modeling thereof described in section "Problem Model". In section "Algorithm Design" the methodology achieving adjustable level of diversity in the genetic algorithm is presented. Section "Algorithm Experiments" describes the experiments conducted and the results obtained in terms of algorithm performance, with a validation study of secondary and tertiary structure in sections "Primary and Secondary Structure Validation Results" and "Tertiary Structure Validation Results". Finally the results and perspectives are summarized in section "Conclusion".

Related Work

This section reviews some of the most relevant works related to the two main areas covered in this chapter: protein design and diversity preservation in metaheuristics.

Protein Design

Most applied work of the IFP is concerned with protein design. Since the first design of a peptide by Gutte et al. [14] using secondary structure rules, numerous works have described different approaches to the IFP problem. The earliest reference to the inverted approach is found in an article by Pabo [24] referring to Drexler [11] stating that protein design engineers could in theory choose from a vast subset of possible sequences containing strategically placed groups that would have a predictable fold. Another early attempt at tackling the IFP is done by Ponder and Richards [25] who used a systematic exhaustive approach of enumerating a selected subset of residue positions. Central to the approach is the focus on packing criteria of core residues, taking a latest available side-chain rotamer library into account. Core residues are internal or buried residues not in contact with solvent. They contribute to the general structure of the protein and rather seldom to its primary function. A rotamer library is a library of known side-chain arrangements in 3D for each residue which is important to consider when evaluating the space filling of the core structure.

A few years later, Bowie et al. [5] introduced a 3D to 1D score for each secondary structure type and six environmental classes determined by (1) area buried in the protein structure and (2) fraction of polar side-chain area. By analyzing 16 known structures, the overall relative probability of observing a residue in a defined environment class is computed. From this and the target tertiary structure, a 3D profile can be generated taking the environment at each residue position into account. The 3D to 1D score is calculated by matching a sequence to the 3D profile of a structure. The result is expressed relatively using the Z-score, indicating the number of standard deviations above the mean of other sequences of same length. Using this method they were able to clearly separate homologs (evolutionary-related proteins) in terms of Z-score from a large set of sequences. Kuhlman and Baker [21] used a Monte Carlo approach of residue and rotamer substitution at 11 nonadjacent core positions, evaluating a free energy function. The lowest energy sequence of five

algorithm runs was chosen, and as a final result half of the generated residues were identical to the reference protein, referred to as "wild type."

The first to use a genetic algorithm (GA) was Jones in [17]. To assess 3D-1D compatibility and define an objective function, a set of statistically determined potentials known from fold recognition are used: pairwise potential and solvation potential. To prevent the generation of unlikely sequences, a residue composition term with an arbitrary weight is added corresponding to the target folding class ($\alpha\alpha$, $\alpha\beta$, $\beta\beta$). Jones concluded that there is no way to be sure the resulting sequences have not been overdesigned as the optimal sequence scores significantly better than the reference. He speculates that the energy optimal shape might be very steep and too hard for the real-world protein to fold into. Therefore, the algorithm should possibly be stopped earlier.

Mayo et al. [29] successfully used backbone flexibility in the design process by generating a set of perturbed backbones and applying enumeration of ten varying residue positions applying dead-end elimination to cut the search space. Similarly, Harbury et al. [15] incorporated such backbone freedom in their design approach. Both latter approaches were evaluated by synthesizing the proteins in the lab. Isogai et al. [16] used a recursive approach searching the 3D profile of the target structure keeping two residues fixed and applying a penalty to residues that protrude into the space with a repulsive function. Collisions among side chains were removed manually by replacing residues with smaller ones. The design was successfully synthesized, but the binding site did not stably bind oxygen.

Wernisch et al. [33] sought to combine the latest approaches into an automatic software solution named DESIGNER. The CHARMM package [6] is used for force-field calculation among side chains and backbone taking all hydrogen atoms bonded and nonbonded into account as well as adding van der Waals forces and electrostatic interaction. Both an exact enumeration approach (branch and bound) and a simple heuristic selecting the optimal rotamer for one random position at the time until a local optimum has been reached were tested. Different experiments aimed at analyzing different setting effects on the results were conducted. One test compares the effect of neglecting the reference energy and solvation energy terms, respectively, when redesigning 11 buried positions in the core. The choice of energy terms largely impacts the amount of polar amino acids, and neglecting the solvation term produced better packing with less cavities. Another test aimed at optimizing the protein surface with its larger proportion of polar amino acids. Again 11 positions are variable and varying settings are tested. First backbone and rotamers are kept fixed, and then alternative rotamers were allowed. Wernisch et al. consider that the energy calculations are approximations. Therefore, the software allows for outputting multiple solutions within a user-defined energy window. When packing constraints apply, DESIGNER generated sequences close to the reference.

Voigt et al. [32] combined the field of directed evolution with that of computational design and seek to benefit from both. Directed evolution is concerned with improving specific protein properties or functions mainly by applying a series of mutations to the target as mutagenesis in nature. In their computational method, energy was used to predict structural stability, and residues with low

entropy are detected as more tolerant to mutations. They also argued that coupled residues should be substituted together as several replacements need to take place to demonstrate improvement. High variability was observed on the exposed residues, and in general the variability should guide mutagenesis to allow the generation of a family of divergent sequences with structural integrity intact.

Klepeis et al. [20] presented a two-stage approach where an integer program is first used to generate a list of low-energy sequences which are then evaluated in terms of their fold. Using a force field based on pairwise C_α, distance-dependent interaction potential gives a more relaxed backbone flexibility constraint with less empirically tuned parameters. Validation was done by improving the activity of Compstatin, a 13-residue-long peptide fundamental in inhibiting complement activation. Certain residue positions and types were restricted based on knowledge about the functional nature and with the goal of increasing activity. Experimental results on 14 designed sequences showed significant activity improvements in most cases, one analogue was six to seven times more active than the wild-type underlining. This two-stage approach with small variations is used to design a template for human β-defensiv-2 in [12] and with more advanced second stage in [3,4].

Smadbeck et al. [28] have recently streamlined the two-stage process and present a server implementation with a usage example. The web interface allows for specifying all inputs: template (rigid/flexible), energy function (C_α, centroid, or any), and biological constraints (on charge and content). Stage two workflow consists of two independent fold specificity and approximate binding affinity modules. These include programs such as CYANA, TINKER, and AMBER for the, first, Rosetta (ab initio, dock, and design) and OREO for the latter.

Finally, Mitra et. al [23] used templates of structure families in combination with a force field to guide the search rather than physics-only-based force fields. Due to shortcomings of the latter, evolutionary-based designs have been demonstrated to be more stable. Experiments were conducted with one of the leading protein structure prediction frameworks, I-TASSER[37]. Previous works have shown that I-TASSER is able to distinguish successful designs from unsuccessful ones and is therefore used as validation of the results also in this work.

The research of the last three decades on the IFP problem has produced many methods, but their complexity and exhaustive nature effectively limits the size of the sequence or decision space that can be sampled. In addition, the final output of these methods consists of a single or few sequences close to the input sequence, where a larger and more diverse set of sequences would be desirable for practitioners.

Multimodal Optimization and Niching

In metaheuristics the subject of exploration vs. exploitation characteristics has been thoroughly studied. For population-based optimization algorithms, it is well known that a higher level of population diversity results in more exploration at the expense of exploitation. An elevated population diversity is especially desirable for *multimodal*, *deceptive*, and/or *dynamic* problems. In general, if diversity tends

toward zero, it indicates that the algorithm has converged toward a single solution, which might be an undesired behavior if it occurs too early. A number of works have focused on maintaining and controlling diversity, such as crowding methods by DeJong [8], fitness sharing by Goldberg and Richardson [13], cellular algorithms by Alba and Dorronsoro [1], and diversity-preserving selection strategies based on hamming distance by Shimodaira [27] and on altruism by Laredo et al. [22]. Another approach consists in designing new objectives through multi-objectivization, with which the problem is transformed into a bi- or multi-objective one. Extending problems with an objective designed specifically for diversity preservation has been proposed by Toffolo and Benini [30], by Deb and Saha [9], and most recently by Wessing et al. [34]. In these works, objectives have been designed based on the hamming distance to the closest neighbor, the distance to the nearest better, and number of individuals in the neighborhood.

In this chapter, the diversity-preserving objective is based on the average distance of each individual to all others which directly targets the global diversity measure stated by the problem, contrary to the pairwise local view of existing works. Given the discrete nature, complexity, and multimodality of the problem, an effective diversity limiting mechanism is required. The proposed approach achieves this with the added value of making the population diversity highly variable depending on a single algorithm setting.

Problem Description

The focus in this work is on finding multiple and diversified solutions to the inverse folding problem (IFP).

A simplified model is developed to match solely the reference secondary structure – a requirement for the tertiary structure; see Fig. 3 for a schematic representation. This is motivated by the fact that computing the tertiary structure of a given input sequence is computationally very expensive which would prevent the usage of a metaheuristic on the entire sequence. The found solutions should be a collection of very dissimilar sequences, as well as dissimilar to the input sequence or its homologs, the naturally occurring, evolutionary-related sequences of the input sequence.

A single solution is represented as a sequence $A = \{aa_i\}$ composed of N residue positions, where $1 \leq i \leq N$ and $aa_i \in \{1, 2, \ldots 20\}$ correspond to the set of 20 possible amino acids. As the solution space consists of a total of 20^N different combinations, considering that N is around 50–200 for typical design targets, alternatives to exhaustive exploration are mandatory.

Sequence Identity

Sequence identity is a common measure designed to assess the similarity of proteins occurring in nature in terms of their primary structure only. When computing

sequence identity, gaps are taken into account during the alignment of the sequences to be able to detect evolutionary relations among the compared proteins even if their sequences are of different lengths. In this work, all sequences being compared have the length of the target sequence and are generated by a random process. The chances of the same subsequence to occur in two different sequences with an offset diminish quickly as the subsequence length increases, which justifies ignoring gaps in the model. For the comparison of final results, the generally accepted approach with taking gaps into account is used.

Problem Model

This section presents the corresponding optimization problem. Two objective functions are first defined for integer encoded solutions $A = \{aa_i\}$. The first function estimates the similarity of secondary structure, in which definition and estimation are provided in sections "Secondary Structure Definition" and "Secondary Structure Estimation," respectively. The second function presented in section "Diversity Measure" is designed to address the diversity requirements of the problem and of the algorithm.

Secondary Structure Definition

Secondary structure refers to the annotation of structure segmentation as seen in Figs. 2 and 3. These segments are the result of the protein naturally folding so that different parts of its 3D structure connect through bonds between amino acids on separated residue positions in the sequence. Tertiary structure annotations are

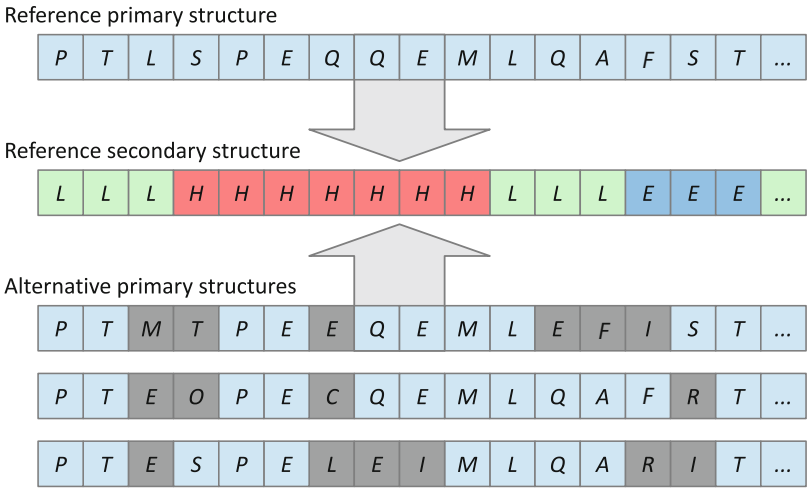


Fig. 3 Primary and secondary structure in the inverted folding problem

done using the "Define Secondary Structure of Proteins" (DSSP) tool [18]. As only the three structure types, *helices (H)*, *sheets (E)*, and *loops (L)*, are considered throughout this work, some simplification is required. In the documentation of DSSP, the following possible annotation types are found:

- G = 3-turn helix (310 helix). Min length three residues.
- H = 4-turn helix (α helix). Min length four residues.
- I = 5-turn helix (π helix). Min length five residues.
- T = hydrogen-bonded turn (three, four, or five turn)
- E = extended strand in parallel and/or antiparallel β-sheet conformation. Min length two residues.
- B = residue in isolated Îš-bridge (single pair β-sheet hydrogen bond formation)
- S = bend (the only non-hydrogen bond-based assignment).
- C = coil (residues which are not in any of the above conformations).

With *helices* characterized by a corkscrew shape, *sheets* as parallel-connected segments, and *loops* as everything else, the above structure types are simplified as follows:

$$G, H, I \Rightarrow H;\ E, B \Rightarrow E;\ T, C, S \Rightarrow L$$

Secondary Structure Estimation

The goal of this objective is to distinguish sequences by assigning better score to sequences that may match the reference secondary structure better. Using the tool PROFphd, updated to ReProf [26], the likely secondary structure type $T_{\text{pred}}(i)$ can be predicted per amino acid aa_i in A with a reliability $R_{\text{pred}}(i) \in \{0 \ldots 9\}$ by means of posterior neural network training. With $T_{\text{ref}}(i)$ the actual type found at position i of the reference secondary structure, the estimated similarity score $F_{\text{sec}}(A)$ is calculated as a sum of reliability weighted (mis)matches:

$$F_{\text{sec}}(A) = \frac{\Sigma_{\max} - \sum_{i=1}^{N} s_i \cdot (C^R_{\text{pred}} + R_{\text{pred}}(i))}{\Sigma_{\max}},\ F_{\text{sec}}(A) \in \{0 \ldots 2\}\ . \tag{1}$$

where

$$s_i = \begin{cases} 1 & \text{if } T_{\text{pred}}(i) = T_{ref}(i) \\ -1 & \text{if } T_{\text{pred}}(i) \neq T_{ref}(i) \end{cases}$$

and

$$\Sigma_{\max} = (C^R_{\text{pred}} + \max R_{\text{pred}}) \cdot N$$

Equation 1 is normalized by the maximum possible sum, $\Sigma_{\max}$, which may occur if all positions are perfectly matched with the highest possible probability.

In this case the score becomes 0 and it can never become negative. C^R_{pred} is a constant which purpose is to increase the contribution to the score of a matching type prediction that has a low reliability R_{pred}. In the current work, it was chosen such that $C^R_{\text{pred}} + \max R_{\text{pred}} = 20$. The reference types $T_{\text{ref}}(i)$ are extracted from the reference structure S_{ref} per residue position i as described in section "Secondary Structure Definition".

Diversity Measure

As a requirement stated in the problem description, the algorithm should not only find a single very good solution, but rather a number of good solutions as different as possible from each other and from the reference sequence. This diversity requirement is closely related to the models described in the ▸ Chap. 32, "Diversity and Equity Models". However, as the problem solutions in this work represent protein sequences, not binary selection of elements, a slightly different approach to the distance measure is taken. An effective and simple measure of distance between two sequences is the Hamming distance, defined as the number of single-point permutations necessary to convert one into the other. Not taking gaps or varying sequence lengths into account, for two sequences $A = \{aa_i\}$ and $A' = \{aa'_i\}$ where $1 \leq i \leq N$, the Hamming distance between them is defined as:

$$d_{\text{Hamm}}(A, A') = \sum_{i=1}^{N} d_i, \; d_i = \begin{cases} 0 & \text{if } aa_i = aa'_i \\ 1 & \text{otherwise} \end{cases} . \tag{2}$$

To obtain a *nonnegative* objective value for minimization, the average Hamming distance to all other $M - 1$ individuals in the current population minus the sequence length N is computed:

$$F_{\text{div}}(A) = N - \frac{1}{M-1} \sum_{i=1}^{M-1} d_{\text{Hamm}}(A, A_i), \; F_{\text{div}}(A) \in \{0 \ldots N\} . \tag{3}$$

This function favors individuals farthest away from the rest of the population. In addition, if a sequence similar to the input sequence exists in the population, the function will have a mutually repulsive effect and penalize it. In summary the function addresses two problem requirements: (1) promoting diversity and (2) promoting sequences which are not equal to the reference sequence.

Algorithm Design

In this chapter the DAO-QC NSGA-II algorithm proposed to tackle the IFP is presented. The modification of the NSGA-II [10], a well-known multi-objective

genetic algorithm, includes the diversity objective (DAO) $F_{\text{div}}(A)$ that enhances the explorative characteristic of the algorithm.

This favorable feature for solving *multimodal* problems is complemented by two modifications of the original algorithm highlighted in Algorithm 1: removal of doubles described in section "Removal of Doubles" and quantile constraint to promote good individuals in section "Quantile Constraint".

Removal of Doubles

In the context of diversity preservation, having two or more identical individuals in the population is undesired. Especially as in [30] when diversity for a sequence A is defined as the minimal distance to any other sequence A', a sequence $A = A'$ must be avoided. With the diversity calculation proposed in section "Diversity Measure", this issue has less impact, but nevertheless doubles are removed in this work. The procedure is executed in line 6 of Algorithm 1 after the application of genetic operations and before non-dominated sorting and crowding-based truncation of the unified population R_t takes place in NSGA-II.

When two identical sequences are detected, one of them is mutated with a probability of $\frac{5}{N}$ to distance the individual with a Hamming distance of 5 on average.

Quantile Constraint

A consequence of the nature of the objectives $F_{\text{sec}}(A)$ and $F_{\text{div}}(A)$ is that the latter is much easier to optimize; hence, the population quickly consists of very diversified individuals with poor fitness according to $F_{\text{sec}}(A)$. To counter this effect, the quantile constraint (QC) is introduced at the end of every generation, in line 9 of Algorithm 1. Given a quantile size C_q, the population P_t at time t is divided according to $F_{\text{sec}}(A)$ into a C_q%-sized partition and a $100 - C_q$%-sized partition. All individuals in the former, less fit, partition are assigned a constraint penalty that prevents the constrained individuals from mating and surviving the next generations. Hence, the population is cleaned from individuals far spread in

Algorithm 1: DAO-QC NSGA-II

1: $Initialize(P_0)$ {randomly generated individuals}
2: $t \leftarrow 0$
3: **while** $t < t_{\max}$ **do**
4: $\quad Q_t \leftarrow makeNewPop(P_t)$ {selection, mutation, recombination}
5: $\quad R_t \leftarrow P_t \cup Q_t$
6: $\quad mutateDoubles(R_t)$ {eliminate doubles by mutation}
7: $\quad F \leftarrow fastNonDominatedSort(R_t)$
8: $\quad P_t \leftarrow truncate(F)$ {based on domination and crowding}
9: $\quad setQuantileConstraint(P_t)$ {to penalize worst quantile}
10: **end while**

the solution space, but with poor $F_{\text{sec}}(A)$ score. The selection pressure can then be selectively adjusted by changing the size of the quantile C_q, which has been tested using $C_q \in \{0\%, 5\%, 10\%, 25\%\}$.

Algorithm Experiments

This section presents and compares the experimental results obtained with the proposed DAO-QC MOGA to a standard generational GA on two protein samples. The experimental setup is first introduced, starting with the two protein samples used and followed by the algorithms' parameters. These initial experiments focus on analyzing the effect of different quantile constraint settings on the proposed algorithms' performance. To this end, the diversity, convergence, and final fitness are compared to a standard generational GA for both of the test samples.

Protein Samples

The two chosen protein samples, namely, *1OAI* and *1URR*, are illustrated in Fig. 4a, b, respectively. *1OAI* is characterized by a length of 59 residues and a secondary structure that consists of 4 helices. *1URR* is 97 residues long, and its secondary structure is composed of 2 helices and 6 beta-sheets.

Experimental Setup

Table 1 summarizes the settings of both the standard generational GA and the proposed DAO-QC MOGA, i.e., the GA extended by multi-objectivization with diversity as objective (DAO) and quantile constraint (QC). Both algorithms use a population of 100 individuals, a binary tournament selection, 1-point crossover

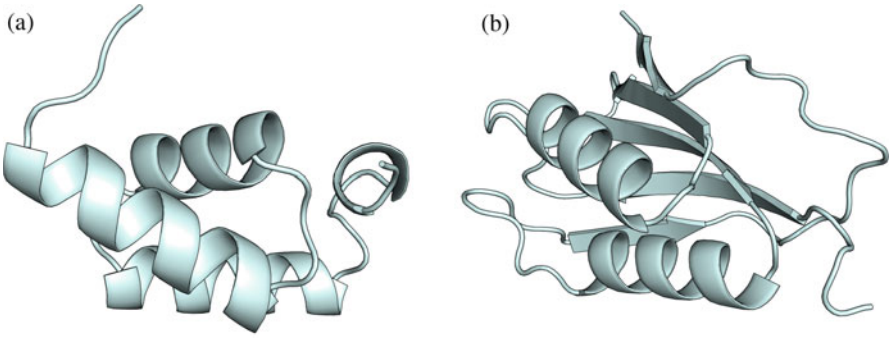


Fig. 4 Three-dimensional structure of the samples. (**a**) *1OAI*. (**b**) *1URR*

Table 1 Algorithm settings

Setting	Value
Population size	100
Algorithm	NSGA-II and std GA
Termination condition	30,000 function evaluations
Selection	Binary tournament (BT)
Crossover operator	1-point, $p_c = 1.0$
Mutation operator	Uniform, $p_m = \frac{1}{N}$
Quantile constraint	$C_q \in \{0\%, 5\%, 10\%, 25\%\}$

Table 2 *1OAI* average fitness cross-comparison

	GA	DAO-QC0	DAO-QC5	DAO-QC10	DAO-QC25
GA	⁄	−0.101∇	−0.0272∇	0.000138 -	0.00896▲
DAO-QC0		⁄	0.0743▲	0.102▲	0.110▲
DAO-QC5			⁄	0.0273▲	0.0361▲
DAO-QC10				⁄	0.00882▲
DAO-QC25					⁄

Table 3 *1OAI* average diversity cross-comparison

	GA	DAO-QC0	DAO-QC5	DAO-QC10	DAO-QC25
GA	⁄	−46.996∇	−45.068∇	−36.946∇	−14.065∇
DAO-QC0		⁄	1.928▲	10.050▲	32.931▲
DAO-QC5			⁄	8.122▲	31.003▲
DAO-QC10				⁄	22.880▲
DAO-QC25					⁄

with probability p_c=1.0, and uniform mutation with probability $p_m = \frac{1}{N}$. The termination condition was set to 30,000 fitness function evaluations, and each experiment was repeated 30 times. Four different values of quantile constraint C_q are considered for DAO-QC NSGA-II: 0%, 5%, 10%, and 25% of the population.

Algorithm Results

In the following the results of the standard GA and the DAO-QC NSGA-II with four different C_q settings are presented and compared in terms of average population fitness, population diversity, and convergence of these indicators based on 30 individual runs.

Tables 2, 3, 4, and 5 show all pairwise comparisons of the algorithm mean value difference. The Wilcoxon test indicator [35] with a 5% significance level provides statistical confidence in comparing the sets with symbols "▲," "∇," and "-" indicating superior, inferior, and no difference. In terms of fitness, the algorithms

Table 4 *1URR* average fitness cross-comparison

	GA	DAO-QC0	DAO-QC5	DAO-QC10	DAO-QC25
GA	／	−0.117∇	−0.026∇	0.0229▲	0.0321▲
DAO-QC0		／	0.0909▲	0.140▲	0.149▲
DAO-QC5			／	0.0489▲	0.058▲
DAO-QC10				／	0.00911▲
DAO-QC25					／

Table 5 *1URR* average diversity cross-comparison

	GA	DAO-QC0	DAO-QC5	DAO-QC10	DAO-QC25
GA	／	−46.611∇	−42.754∇	−25.651∇	−1.389 -
DAO-QC0		／	3.857▲	20.959▲	45.221▲
DAO-QC5			／	17.102▲	41.365▲
DAO-QC10				／	24.262▲
DAO-QC25					／

are ordered in the following way: DAO-QC25 > DAO-QC10 ∼ GA > DAO-QC5 > DAO-QC0 for sample *1OAI* and DAO-QC25 > DAO-QC10 > GA > DAO-QC5 > DAO-QC0 for sample *1URR* with statistical confidence. In terms of diversity, the order becomes DAO-QC0 > DAO-QC5 > DAO-QC10 > DAO-QC25 > GA and DAO-QC0 > DAO-QC5 > DAO-QC10 > DAO-QC25 ∼ GA for samples *1OAI* and *1URR*, respectively. As expected, the higher diversity of the DAO-QC0 approach comes at the expense of a lower average fitness due to the exploration/exploitation trade-off. However, an increase of C_q to 10% or 25% leads to increased exploitation, allowing the DAO-QC NSGA-II algorithm to be constantly ahead of the GA in terms of average fitness until depletion of the evaluation budget as seen in Figs. 5 and 6. Further, the appropriate setting ($C_q = 25\%$ for *1OAI*, $C_q = 10\%$ for *1URR*) allows the DAO-QC NSGA-II to outperform the GA in terms of fitness and diversity *at the same time*. Remaining observations to mention are steeper final fitness slopes for the sample *1URR* with settings $C_q \in \{10\%, 25\%\}$, than the standard GA and specifically for the sample *1OAI*; the diversity is observed to clearly start increasing once the fitness has converged. The steeper final slopes and the increased performance in fitness can be partially explained by the constantly high, and at times increasing, diversity combined with the highly *multimodal* nature of the problem. An elevated diversity clearly increases the chances of the algorithm discovering good new solutions in the rugged fitness landscape of this type of problem.

Table 6 shows the final average fitness and diversity values of all algorithms on both samples with their respective standard deviation. In each column the darker background emphasizes the best result, while the lighter background emphasizes the worst result. With $C_q = 25\%$ the proposed algorithm clearly outperforms the GA with statistical confidence for both samples with average values 0.105 vs. 0.136 and 0.193 vs. 0.242, respectively. From the figure and the table, it is also evident that the

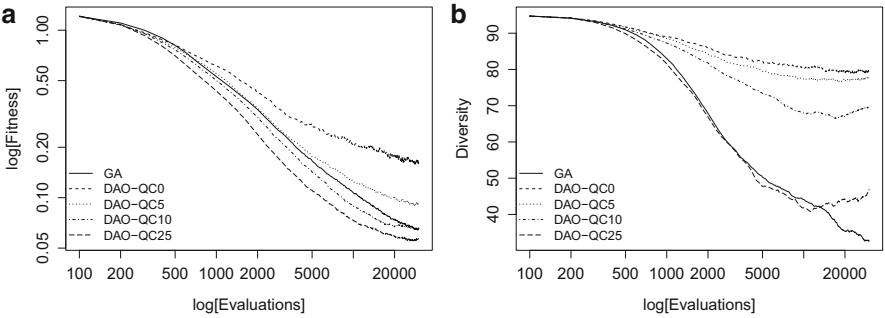


Fig. 5 Convergence of *1OAI*. (**a**) Average fitness convergence. (**b**) Average diversity convergence

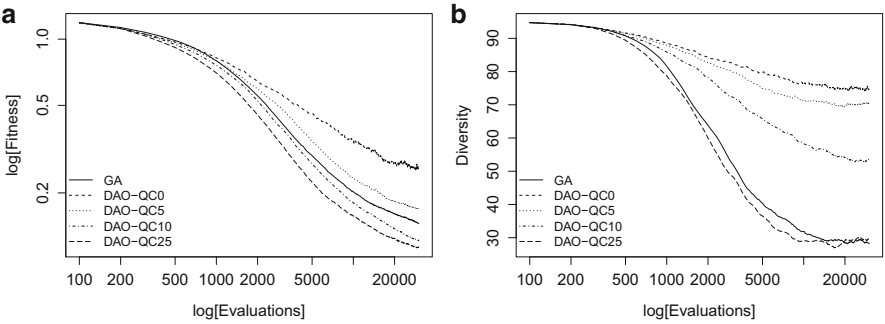


Fig. 6 Convergence of *1URR*. (**a**) Average fitness convergence. (**b**) Average diversity convergence

Table 6 Summary of final values

	1OAI		1URR	
	Average fitness	Average diversity	Average fitness	Average diversity
GA	$0.136_{\pm 1.52e-01}$	$43.578_{\pm 1.32e+01}$	$0.242_{\pm 1.77e-01}$	$35.565_{\pm 1.42e+01}$
DAO-QC0	$0.235_{\pm 1.41e-01}$	$80.992_{\pm 3.23e+00}$	$0.363_{\pm 1.62e-01}$	$77.163_{\pm 4.29e+00}$
DAO-QC5	$0.156_{\pm 1.45e-01}$	$78.598_{\pm 3.69e+00}$	$0.271_{\pm 1.80e-01}$	$72.763_{\pm 5.74e+00}$
DAO-QC10	$0.124_{\pm 1.45e-01}$	$70.564_{\pm 6.56e+00}$	$0.218_{\pm 1.75e-01}$	$59.400_{\pm 1.00e+01}$
DAO-QC25	$0.105_{\pm 1.37e-01}$	$47.484_{\pm 1.11e+01}$	$0.193_{\pm 1.65e-01}$	$34.182_{\pm 1.39e+01}$

value of the quantile or QC setting has a direct impact on the population diversity, providing an effective tool for achieving the level of exploitation vs. exploration preferred.

Structure Validation

In this second experimental step, the protein sequences generated by the best performing algorithm are validated. To this end the I-TASSER [37] prediction tool is used to generate their secondary and tertiary structures that will be compared to the structure of the targeted protein. For each sample, the 5 best generated sequences of the final population in each of the 30 individual runs are selected. This means a total of 300 I-TASSER runs for the 2 protein samples, each run taking around 2 days, which amounts to almost 2 years of CPU time. It is to be noted that the I-TASSER prediction itself is subject to erroneous results; hence, a 100% certainty can never be achieved unless the proteins are synthesized in a wet lab. In the following, the sequences and their I-TASSER predictions are analyzed in terms of primary and secondary structure in section "Primary and Secondary Structure Validation Results" and then tertiary structure in section "Tertiary Structure Validation Results".

Primary and Secondary Structure Validation Results

The goal in this section is to analyze how well the secondary structure of the reference protein is reproduced in the predicted model.

Table 7 shows a summary of the two proteins tested. Clearly, the generated sequences share very little resemblance with the original input sequence seen from a sequence identity of about 20% and 15%, respectively, with a very low deviation. Achieving low sequence identity by itself is not a challenging task unless a good structure match is obtained at the same time. The table shows this as the average percentage, μ, of positions in the secondary annotation of the I-TASSER predicted model that correctly matches those of the input annotation. Average percentage μ and standard deviation of the average percentage σ are given for each of the three structure types *H*, *E*, and *L*. As it can be seen, the helices are correctly predicted on more than 90% of the positions in both proteins. For the slightly bigger *1URR* sample which contrary to *1OAI* contains many extended sheets, the sheet match percentage is lower – slightly below 50%.

Table 7 Summary of secondary structure prediction match

Protein	μ_{Identity}	σ_{Identity}	μ_{Helix}	σ_{Helix}	μ_{Sheet}	σ_{Sheet}	μ_{Loop}	σ_{Loop}
1OAI	20.67	4.100	93.348	6.343	0	0	82.814	7.368
1URR	15.23	3.225	93.787	6.563	42.108	8.898	85.523	8.239

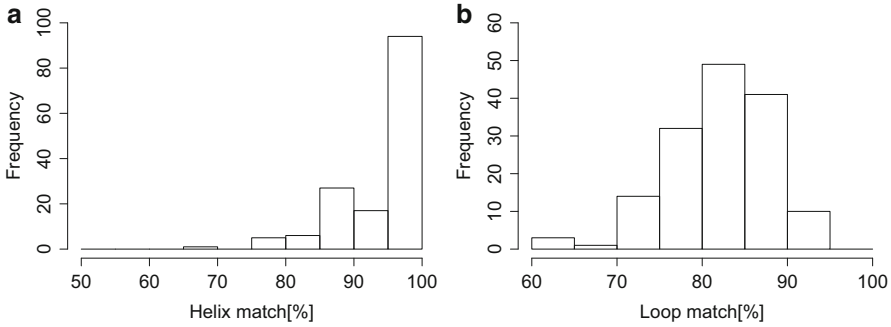


Fig. 7 Match histograms of *1OAI*. (**a**) *Helix* match histogram. (**b**) *Loop* match histogram.

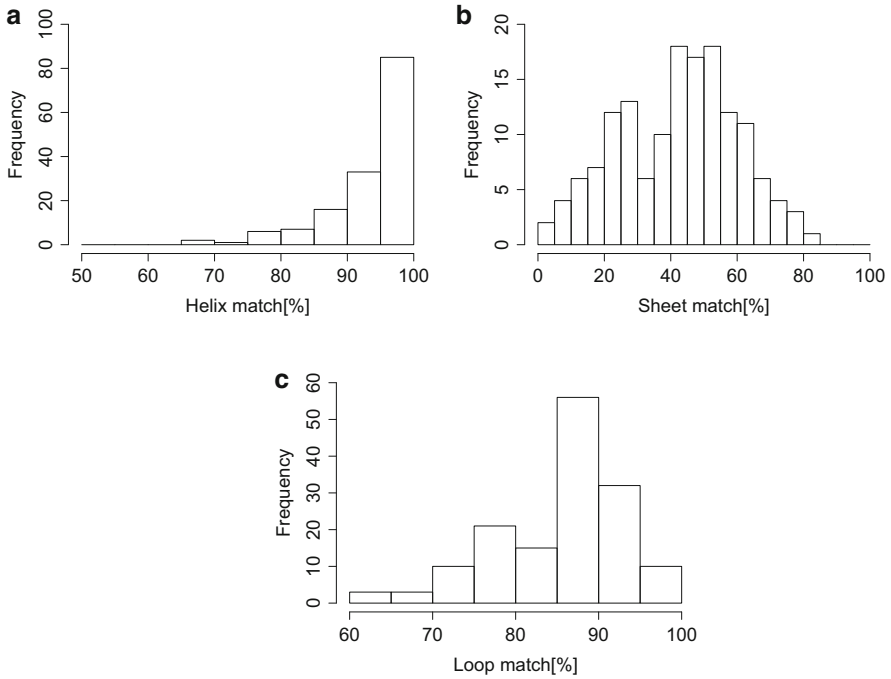


Fig. 8 Match histograms of *1URR*. (**a**) *Helix* match histogram. (**b**) *Sheet* match histogram. (**c**) *Loop* match histogram

Figures 7 and 8 illustrate the same data as histograms. Figures 7a and 8a clearly demonstrate that *helix* structures are very well matched in all 300 structure predictions. Almost all of the tested generated individuals have a match percentage of over 80%, and the majority is above 90% for both samples.

For *loop* segments presented in Figs. 7b and 8c, the majority is still above 80% but with a high spread. The statistics for *sheet* segments show that there is a limit to

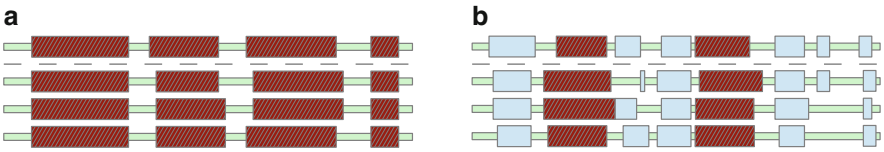


Fig. 9 Secondary structure of reference (on top) compared to three selected generated models. Darker sections are *helices*, lighter are *sheets*, and the rest represents *loop* structure. (**a**) *1OAI*. (**b**) *1URR*

the performance of an approach optimizing only an approximate secondary structure prediction. Considering that the *1URR* sample consists of six *sheet* segments across the whole of its length, then 42% can be considered as a rather good result. The lower success rate of predicting sheets is due to the fact that a sheet can only be observed in the secondary structure if the I-TASSER predicted structure actually did fold close enough to the reference tertiary structure, to allow the extended sheet to form. A *helix* is a much more local structure mostly independent of the global fold, hence easier to achieve in this analysis.

Figure 9 and Table 9 show the alignment of three of the best aligned individually generated sequences. This is to show specific examples of the results which have been averaged in Table 7, and the tendency remains the same: *helices* are very well defined with above 95% positions matched, *loops* slightly less with $\pm 90\%$, and $\pm 80\%$ for samples *1OAI* and *1URR*, respectively. The *1OAI* sample is clearly an easier target due to its *helix*-only structure compared to the majority of *sheet* structures in the *1URR* sample. The other columns of the table will be discussed in the next section.

Tertiary Structure Validation Results

In the following the tertiary structure of the predicted proteins is validated by three-dimensional comparison.

The TM-Score detailed in [39] is a measure that is used to assess the similarity between two structures, with larger values indicating greater resemblance and 1.0 a maximum value for identical structures. According to Xu and Zhang [36], two proteins can be considered to be in the same fold if comparing them gives a TM-Score above 0.5. Though the average TM-Score is above 0.4 and close to 0.5 for the first sample, this is actually the case for $1{-}in{-}5$ for *1OAI* and $1{-}in{-}15$ for *1URR* as seen in Table 8. The table further shows the number N of predictions that had a TM-Score above 0.2, 0.4, 0.6, 0.7, and 0.8. The general results presented in section "Primary and Secondary Structure Validation Results" are confirmed here, and it is clear that the *sheet* structures of *1URR* are hard to match and that the approach is much more successful in predicting *helix* structures (see Table 9).

Table 8 Summary of tertiary structure prediction match

Protein	$\mu_{TM\text{-Score}}$	$\sigma_{TM\text{-Score}}$	$N_{TM>0.2}$	$N_{TM>0.4}$	$N_{TM>0.5}$	$N_{TM>0.6}$	$N_{TM>0.7}$	$N_{TM>0.8}$
1OAI	0.493	0.135	150	102	51	32	18	4
1URR	0.416	0.061	150	91	10	0	0	0

Table 9 Three selected generated models and their alignment scores with *1OAI* and *1URR* as reference

Nr.	Identity	N< $5A$	$\text{RMSD}_{N<5A}$	RMSD	GDT_{TS}	TM-Score	Helix	Sheet	Loop
1	13.6	58	1.21	1.760	92.797	0.8667	95.12	0	94.44
2	25.4	58	1.35	1.838	88.983	0.8350	95.12	0	88.89
3	18.6	56	1.84	2.722	88.136	0.8015	97.56	0	94.44

Nr.	Identity	N< $5A$	$\text{RMSD}_{N<5A}$	RMSD	GDT_{TS}	TM-Score	Helix	Sheet	Loop
1	19.6	73	2.85	7.484	50.258	0.5374	96	75.68	71.43
2	20.6	67	3.20	4.933	50.773	0.5027	100	81.08	80
3	17.5	74	2.94	9.059	48.711	0.5138	100	72.97	80

The last step in the tertiary validation consists in superposing the fully I-TASSER predicted tertiary structure model of one generated sequence with the target reference. This is illustrated in Figs. 10 and 11 where the first of the three individually generated sequences in Table 9 and Fig. 9 is used.

The models for *1OAI* are all very close to the reference seen from the high *helix* and *loop* match percentage, and in addition the first model for *1OAI* has a very low sequence identity and at the same time very high TM and GDT scores (see Table 9). The first model for *1URR* also has very high *helix* match percentage and good *loop* and *sheet* percentages. However, the TM and GDT scores are less satisfactory. This result is visible in Fig. 11 where the *helices* and *sheets* cannot be fully aligned with the reference and the fact that one *sheet* has been bound to the structure in the wrong location (at the top of the figure rather than at the bottom).

In Table 9 the second column shows sequence identity with gaps, the third shows the length of the longest continuous segment $N < 5A$ that can be fitted below a $5A$ threshold after super-positioning the two structures. The root-mean-square deviation (RMSD) measure is based on the pairwise distance between every residue position in the two tertiary structures, and the fourth column regards only those positions counted in column three, the fifth column regards the total of position. The global distance test (GDT) total score (TS) is a measure indicating the total average of the average percentage of residue positions that can be fitted below each of the thresholds $\{0.5A, 1.0A, 1.5A, \ldots 10.0A\}$. The final four columns are TM-Score and the percentage match of *helix*, *sheet*, and *loop* positions already discussed. Columns three to six were computed with the tertiary structure alignment tool LGA detailed in [38] with default global distance test (GDT) and longest continuous segment (LCS) analysis settings.

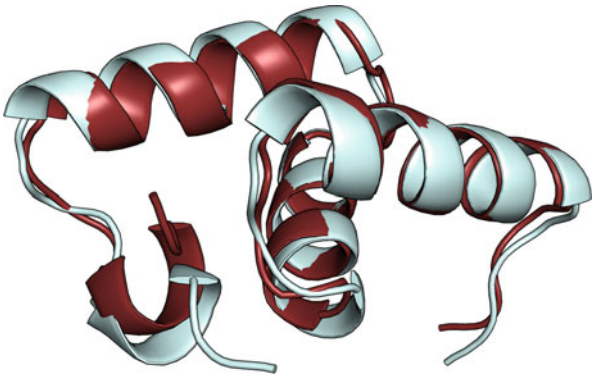


Fig. 10 Super-positioning of a predicted model (dark) with 1OAI reference (light)

Fig. 11 Super-positioning of a predicted model (dark) with 1URR reference (light)

Conclusion

In this chapter an evolutionary-based approach to find a large amount of protein sequences that may result in a given reference secondary and tertiary structure was presented. This problem, referred to as the inverse folding problem (IFP), has received a lot of attention in theoretical chemistry and biophysics over the last 30 years, mostly for its potential application in protein design. It is also of interest to study the extent of the sequence space that may produce similar tertiary structures and how far from the original reference sequence such solutions can be found.

By defining the task as finding highly diverse sequences with most similar secondary structures, an optimization problem was modeled to find many well-scoring sequences in a few hours, which is fast compared to state-of-the-art methods. To achieve high diversity, the requirement has been adapted as an additional objective and extending the problem through *multi-objectivization* to become multi-objective with diversity as objective (DAO). Combining the quantile constraint (QC) with the DAO approach allows to shift focus arbitrarily between diversity and fitness, and final results found significantly better than the standard GA with statistical significance. At the same time, the final diversity remains

significantly higher for all QC-settings except the DAO-QC25 which produces diversity comparable to the standard GA for the *1URR* sample. For the *1OAI* sample with increased QC setting, a clear increase in diversity is observed toward the end of the run, once very good fitness values have been found. In addition to the higher performance on diversity, the algorithm fitness convergence was observed as being generally faster and partially steeper toward the end of runs, than for the standard GA.

For further validation, the five best generated sequences of each independent run of the $DAO-QC25$ algorithm variant were selected systematically and their folded structure predicted by I-TASSER, an established structure prediction software. The 300 predicted tertiary structures were annotated by DSSP for secondary structure analysis of *helix*, *sheet*, and *loop* formations. As could be expected, the method works better for the sample with more defined *helical* secondary structure, and less well in *sheet* and *loop* regions, especially as the latter region is not expressed by the objective function. Indeed *sheet* formations require the tertiary structure to fold properly to be captured in secondary structure. Nevertheless the *1URR* sample *sheet* match percentage is slightly below 50% averaged over all generated predictions. In addition the majority of match percentages are above 80% for *loops* and above 90% for *helices* in both samples. Tertiary structure validation was done by comparing the predicted structures to their respective reference by tertiary structure super-position. For both samples meaningful predictions were generated with a $TM-Score$ above 0.5 observed $1-in-5$ for *1OAI* and $1-in-15$ for *1URR*. These results indicate that this approach is able to generate a massive amount of sequences, with a reasonable amount being likely to actually fold as expected. At the same time, the limits in terms of achieving larger formations of *sheets* are demonstrated.

Future and ongoing works will address the identification of those sequences that actually fold into the reference structure by designing new objectives and constraints and also addressing *loop* and *beta-sheet* regions. Independent of this, sequences found could already be used as starting points for other exact protein design methods and possibly generate successful designs with a very low sequence identity compared to the reference. Additional possible applications could be generating meaningful decoy sets for other studies or finding bridges in sequence space between known proteins of the same structural classes.

Cross-References

- ▸ Diversity and Equity Models
- ▸ Evolutionary Algorithms
- ▸ Genetic Algorithms
- ▸ Multi-objective Optimization

Acknowledgments Work was funded by the National Research Fund of Luxembourg (FNR) as part of the EVOPERF project at the University of Luxembourg with the AFR contract no. 1356145. Experiments were carried out using the HPC facility of the University of Luxembourg [31].

References

1. Alba E, Dorronsoro B (2005) The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. IEEE Trans Evol Comput 9(2):126–142
2. Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P (2002) Molecular biology of the cell. Garland Science, New York
3. Bellows ML, Fung HK, Taylor MS, Floudas CA, Lopez de Victoria A, Morikis D (2010) New compstatin variants through two de novo protein design frameworks. Biophys J 98(10):2337–2346
4. Bellows ML, Taylor MS, Cole PA, Shen L, Siliciano RF, Fung HK, Floudas CA (2010) Discovery of entry inhibitors for HIV-1 via a new de novo protein design framework. Biophys J 99(10):3445–3453
5. Bowie JU, Lüthy R, Eisenberg D (1991) A method to identify protein sequences that fold into a known three-dimensional structure. Science (New York, N.Y.) 253(5016):164–170
6. Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M (1983) Charmm – a program for macromolecular energy, minimization, and dynamics calculations. J Comput Chem 4(2):187–217
7. Chen W, Brühlmann F, Richins RD, Mulchandani A (1999) Engineering of improved microbes and enzymes for bioremediation. Curr Opin Biotechnol 10(2):137–141
8. De Jong AK (1975) Analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Ann Arbor. Dissertation Abstracts International 36(10):5140B, University Microfilms Number 76–9381
9. Deb K, Saha A (2010) Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach. In: Proceedings of the 12th annual conference on genetic and evolutionary computation. ACM, pp 447–454
10. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
11. Drexler KE (1981) Molecular engineering: an approach to the development of general capabilities for molecular manipulation. Proc Natl Acad Sci 78(9):5275–5278
12. Fung HK, Floudas CA, Taylor MS, Zhang L, Morikis D (2008) Toward full-sequence de novo protein design with flexible templates for human beta-defensin-2. Biophys J 94(2):584–599
13. Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Grefenstette JJ (ed) Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms. Lawrence Erlbaum, Hillsdale, pp 41–49
14. Gutte B, Däumigen M, Wittschieber E (1979) Design, synthesis and characterisation of a 34-residue polypeptide that interacts with nucleic acids. Nature 281(5733):650–655
15. Harbury PB, Plecs JJ, Tidor B, Alber T, Kim PS (1998) High-resolution protein design with backbone freedom. Science 282(5393):1462–1467
16. Isogai Y, Ota M, Fujisawa T, Izuno H, Mukai M, Nakamura H, Iizuka T, Nishikawa K (1999) Design and synthesis of a globin fold. Biochemistry 38(23):7431–7443
17. Jones DT (1994) De novo protein design using pairwise potentials and a genetic algorithm. Protein Sci 3:567–574
18. Kabsch W, Sander C (1983) Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22(12):2577–2637
19. Klein F, Mouquet H, Dosenovic P, Scheid JF, Scharf L, Nussenzweig CM (2013) Antibodies in HIV-1 vaccine development and therapy. Science (New York, N.Y.) 341(6151):1199–204

20. Klepeis JL, Floudas CA, Morikis D, Tsokos CG, Lambris JD (2004) Design of peptide analogues with improved activity using a novel de novo protein design approach. Ind Eng Chem Res 43(14):3817–3826
21. Kuhlman B, Baker D (2000) Native protein sequences are close to optimal for their structures. Proc Natl Acad Sci 97(19):10383–10388
22. Laredo JLJ, Nielsen SS, Danoy G, Bouvry P, Fernandes CM (2014) Cooperative selection: improving tournament selection via altruism. Accepted for publication in EvoCOP14 – 14th European conference on evolutionary computation in combinatorial optimisation
23. Mitra P, Shultis D, Brender JR, Czajka J, Marsh D, Gray F, Cierpicki T, Zhang Y (2013) An evolution-based approach to de novo protein design and case study on mycobacterium tuberculosis. PLoS Comput Biol 9(10):e1003298
24. Pabo C (1983) Molecular technology. Designing proteins and peptides. Nature 301(5897):200
25. Ponder JW, Richards FM (1987) Tertiary templates for proteins: use of packing criteria in the enumeration of allowed sequences for different structural classes. J Mol Biol 193(4):775–791
26. Rost B, Sander C (1994) Combining evolutionary information and neural networks to predict protein secondary structure. Proteins 19(1):55–72
27. Shimodaira H (1997) Dcga: a diversity control oriented genetic algorithm. In: ICTAI, pp 367–374
28. Smadbeck J, Peterson MB, Khoury GA, Taylor MS, Floudas CA (2013) Protein wisdom: a workbench for in silico de novo design of biomolecules. J Vis Exp n77:50476
29. Su A, Mayo SL (1997) Coupling backbone flexibility and amino acid sequence selection in protein design. Protein Sci 6(8):1701–1707
30. Toffolo A, Benini E (2003) Genetic diversity as an objective in multi-objective evolutionary algorithms. Evol Comput 11(2):151–167
31. Varrette S, Bouvry P, Cartiaux H, Georgatos F (2014) Management of an academic HPC cluster: the UL experience. In: Proceedings of the 2014 international conference on high performance computing & simulation (HPCS 2014), Bologna
32. Voigt CA, Mayo SL, Arnold FH, Wang Z-G (2001) Computational method to reduce the search space for directed protein evolution. Proc Natl Acad Sci USA 98(7):3778–3783
33. Wernisch L, Hery S, Wodak S (2000) Automatic protein design with all atom force-fields by exact and heuristic optimization. J Mol Biol 301(3):713–736
34. Wessing S, Preuss M, Rudolph G (2013) Niching by multiobjectivization with neighbor information: trade-offs and benefits. In: 2013 IEEE congress on evolutionary computation (CEC), pp 103–110
35. Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83
36. Xu J, Zhang Y (2010) How significant is a protein structure similarity with tm-score = 0.5? Bioinformatics 26(7):889–895
37. Yang J, Yan R, Roy A, Xu D, Poisson J, Zhang Y (2015) The i-TASSER suite: protein structure and function prediction. Nat Methods 12(1):7–8
38. Zemla A (2003) LGA: a method for finding 3D similarities in protein structures. Nucleic Acids Res 31(13):3370–3374
39. Zhang Y, Skolnick J (2004) Scoring function for automated assessment of protein structure template quality. Proteins Struct Funct Bioinf 57(4):702–710

Linear Layout Problems

34

Eduardo G. Pardo, Rafael Martí, and Abraham Duarte

Contents

Introduction 1026
The Cutwidth Minimization Problem 1029
The Minimum Linear Arrangement Problem 1032
The Vertex Separation Problem 1035
The SumCut/Profile Problem 1038
The Bandwidth Minimization Problem 1040
Conclusion 1043
Cross-References 1044
References 1044

Abstract

The term layout problem comes from the context of Very Large-Scale Integration (VLSI) circuit design. Graph layouts are optimization problems where the main objective is to project an original graph into a predefined host graph, usually a horizontal line. In this paper, some of the most relevant linear layout problems are reviewed, where the purpose is to minimize the objective function: the Cutwidth, the Minimum Linear Arrangement, the Vertex Separation, the SumCut, and

E. G. Pardo (✉)
Dept. of Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, Spain
e-mail: eduardo.pardo@upm.es

R. Martí
Statistics and Operations Research Department, University of Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

A. Duarte
Department of Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: abraham.duarte@urjc.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_45

the Bandwidth. Each problem is presented with its formal definition and it is illustrated with a detailed example. Additionally, the most relevant heuristic methods in the associated literature are reviewed together with the instances used in their evaluation. Since linear layouts represent a challenge for optimization methods in general and, for heuristics in particular, this review pays special attention to the strategies and methodologies which provide high-quality solutions.

Keywords
Embedding · Graph layout · Heuristics · Linear arrangement · Linear layout · Optimization

Introduction

In recent years there has been a growing interest in studying graph layout problems. From a historical perspective, the terms *layout* and *layout problem* come from their early application to the optimal layout of circuits in the context of Very Large-Scale Integration (VLSI) design. A VLSI circuit can be modeled by means of a graph, where the edges represent the wires and the vertices represent the modules. Specifically, given a set of modules, they look for placing these modules on a board in both a non-overlapping manner and wiring together the terminals on the different modules. In general, there are two stages: placement and routing. The former consists in placing the modules on a board while, the latter consists in wiring together the terminals on different modules that should be connected. In Fig. 1 an example of a circuit design with six modules (identified with a different letter, A–F) and their corresponding connections through different tracks are presented. This circuit can be modeled as a graph, as it will be seen below. Notice that the order of the modules can determine, among others, the number of tracks needed to wire the circuit and, therefore, the space needed to build it.

From a theoretical point of view, the main objective of graph layout problems is to project an original graph $G = (V_G, E_G)$ into a predefined host graph $H = (V_H, E_H)$. This projection, generally known as embedding, consists in defining

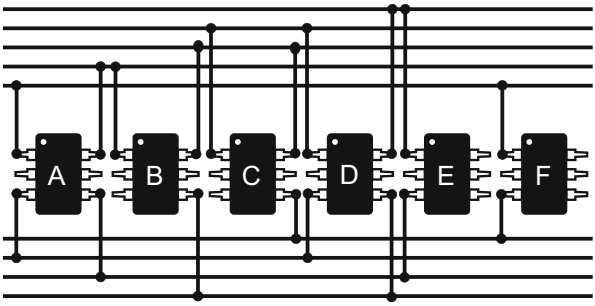


Fig. 1 Circuit design with six modules and its corresponding connections

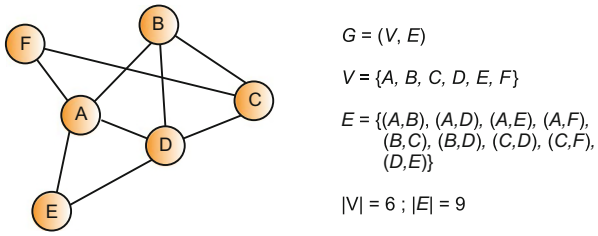


Fig. 2 The graph G with six vertices and nine edges represents the circuit in Fig. 1

a function, mapping the vertices of G into the vertices of H, and associating a path in H for each edge of G. This function assesses that $V_G \subseteq V_H$ and $E_G \subseteq E_H$. According to [26] the quality of an embedding depends on the *dilation*, the *congestion*, and the *load*. The *dilation* of an embedding is the length of the largest associated path in H. The *congestion* is the largest number of paths that share an edge of H. Finally, its *load* is the maximum number of vertices of G that are mapped to a vertex of H.

Let G be a graph (finite, undirected, and without loops), where its vertex set is denoted by V (with $|V| = n$) and its edge set by E (with $|E| = m$). The notation (u, v) stands for an undirected edge between vertices u and v. In Fig. 2 it is shown an example of a general graph G obtained as the result of modeling the circuit in Fig. 1 as a graph. The graph is formed by six vertices corresponding to each of the modules in the circuit (say the vertex set $V = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$) and nine edges corresponding to the circuit connections (say the edge set $E = \{(\mathtt{A}, \mathtt{B}), (\mathtt{A}, \mathtt{D}), (\mathtt{A}, \mathtt{E}), (\mathtt{A}, \mathtt{F}), (\mathtt{B}, \mathtt{C}), (\mathtt{B}, \mathtt{D}), (\mathtt{C}, \mathtt{D}), (\mathtt{C}, \mathtt{F}), (\mathtt{D}, \mathtt{E})\}$). Notice that vertex A in Fig. 2 corresponds with module A in Fig. 1, vertex B in Fig. 2 corresponds with module B in Fig. 1, and so on.

The case in which a graph with n vertices is projected over a path graph with *load* equal to 1 is perhaps the simplest non-trivial embedding problem. In particular, it consists in defining a bijective function, ϕ, where each vertex and edge of G has a unique correspondence in H. The function $\phi : V \rightarrow \{1, \ldots, n\}$ assigns distinct integer numbers (from 1 to n) to each of the n vertices of G. Then, the resulting graph H is usually represented by aligning its vertices on a horizontal line, where each vertex u is located in position $\phi(u)$. This graphical representation is usually known as linear layout, although it has received other names in the literature, for instance, linear ordering [2], linear arrangement [115], numbering [16], embedding in the line [96], or labeling [56]. In Fig. 3 an example of linear layout is shown where vertex A occupies the first position on the line, vertex B the second one, and so on. The horizontal dotted line represents the host graph structure. For the sake of simplicity, the linear layout is usually represented as an array $\phi = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$, where the position of each vertex $u \in V$ in this array corresponds to $\phi(u)$.

Linear layout is, probably, the most studied family of problems in the context of graph layout problems, as shown below. However, there exist other interesting embeddings on more general host graphs than a path, for instance, in closed paths,

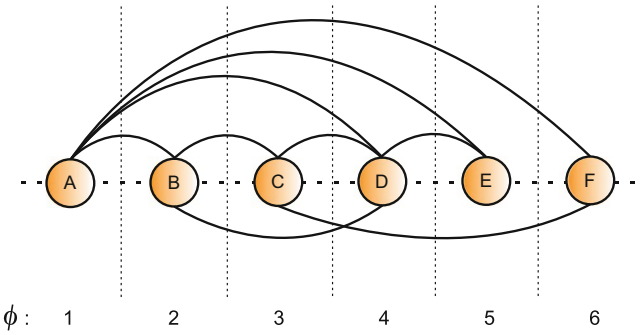
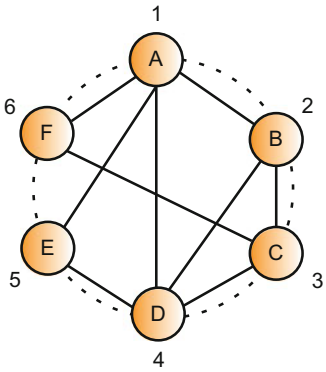


Fig. 3 A possible linear layout of graph G

Fig. 4 A possible circular layout of graph G



usually known as cycles [77, 102, 117]. A circular layout is a function $\rho : V \rightarrow \{1, \ldots, n\}$ that also assigns distinct integer numbers (from 1 to n) to each of the n vertices of the graph, but arranging the vertices of the graph in a cycle where the last vertex (the one with label n) is next to the first vertex (the one with label 1). This embedding has also a *load* equal to 1. In Fig. 4 a possible circular layout of the vertices of the graph in Fig. 2 is shown, where the dotted circle represents the corresponding circular host graph structure.

The embedding of graphs in grids has been studied in [6, 73, 85]. When dealing with grids, it is convenient to label each vertex with a pair of coordinates. A grid layout is a function $\delta : V \rightarrow \{1, \ldots, n_1\} \times \{1, \ldots, n_2\}$ where n_1 denotes the maximum horizontal number, while n_2 indicates the maximum vertical number. Notice that in optimization problems where the *load* is equal to 1, it means that $n = n_1 * n_2$. In Fig. 5 an example of embedding the graph depicted in Fig. 2 on a 3×2 grid is shown. Again, the dotted lines represent the corresponding host graph structure.

There are also other embeddings of practical importance, where the host graph is neither a line nor a cycle nor a grid. Only to mention a few, in [3, 28, 53, 54] the

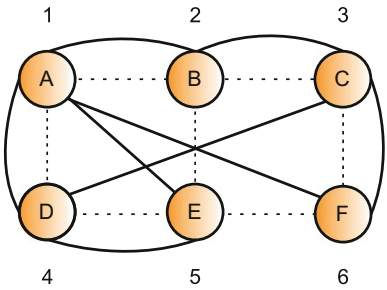


Fig. 5 A possible grid layout of graph G

embedding of graphs on trees is studied, in [55, 121] the embedding of graphs on torus, and in [65, 75] the embedding of graphs in hypercubes.

In this paper, the attention is focused on minimization linear layout problems, by reviewing some of the most relevant optimization problems that falls into this category. In particular, this revision is focused on the problems that have been intensively studied by the heuristic community. See the following sections for each problem: "The Cutwidth Minimization Problem," "The Minimum Linear Arrangement Problem," "The Vertex Separation Problem," "The SumCut/Profile Problem," and "The Bandwidth Minimization Problem," Finally, in section "Conclusion" the conclusions of the study are presented.

The Cutwidth Minimization Problem

The Cutwidth Minimization Problem (CMP) is an NP-hard min-max linear layout problem [43]. It consists in finding an ordering of the vertices of a graph on a line, in such a way that the maximum number of edges between each pair of consecutive vertices is minimized. In terms of graph embedding theory, this problem consists in finding the linear layout (*load* equals to 1) that minimizes the *congestion*.

The classical combinatorial formulation can be described as follows. Given a graph $G = (V, E)$ with $|V| = n$, a linear layout $\phi : V \rightarrow \{1, 2, \ldots, n\}$ of G assigns the integers $\{1, 2, \ldots, n\}$ to the vertices in V, in such a way that each vertex receives a different label. Therefore, the host graph H is a path with the same number of vertices and edges. The cutwidth of a vertex v with respect to ϕ, denoted as $Cut(v, \phi, G)$, is the number of edges $(u, w) \in E$ satisfying $\phi(u) \leq \phi(v) < \phi(w)$. Therefore,

$$Cut(v, \phi, G) = |\{(u, w) \in E : \phi(u) \leq \phi(v) < \phi(w)\}|$$

The cutwidth of G with respect to ϕ is defined as the maximum value of all $Cut(v, \phi, G)$ for $v \in V$. More formally:

$$Cut(\phi, G) = \max_{v \in V} Cut(v, \phi, G)$$

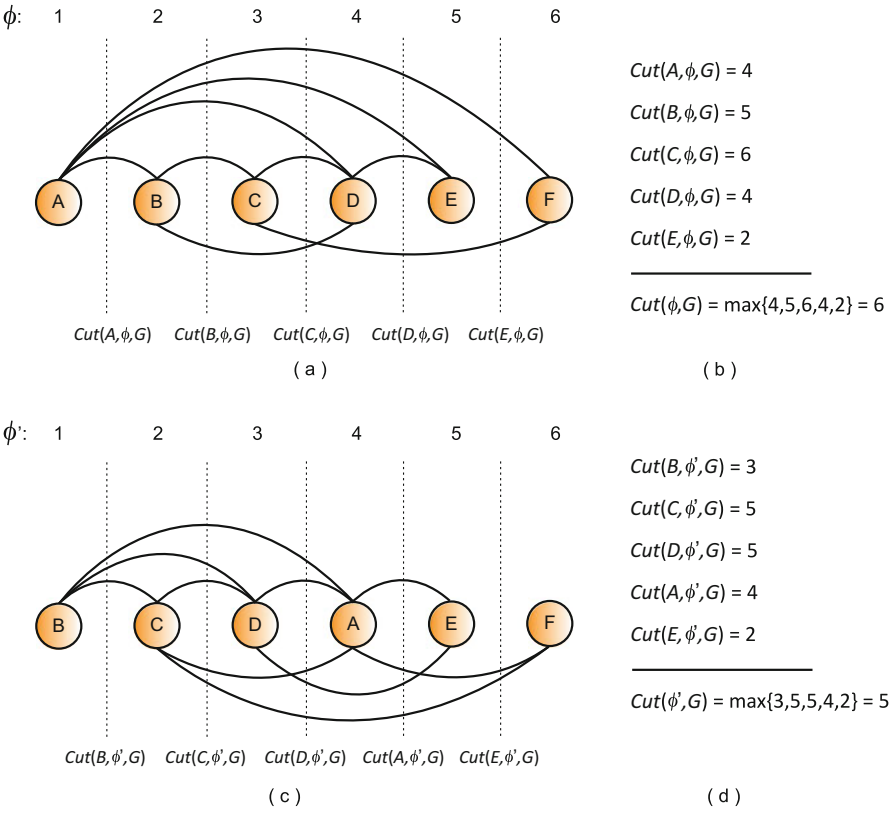


Fig. 6 (**a**) Example of linear layout ϕ. (**b**) Computation of *Cut*-values associated to ϕ. (**c**) Example of linear layout ϕ'. (**d**) Computation of *Cut*-values associated to ϕ'

The optimum cutwidth of G is then defined as the minimum $Cut(\phi, G)$ over all possible layouts Φ of G.

In Fig. 6a, it is shown a possible linear layout ϕ of the graph G depicted in Fig. 2. This layout sets the vertices in a line, as it is commonly represented in the CMP context. Specifically, since $\phi(\mathtt{A}) = 1$, vertex A comes first, followed by vertex B ($\phi(\mathtt{B}) = 2$), and so on. In Fig. 6b it is depicted the computation of the *Cut*-value for each vertex. For example, the *Cut*-value of vertex A is $Cut(\mathtt{A}, \phi, G) = 4$, because the edges $(\mathtt{A}, \mathtt{B})$, $(\mathtt{A}, \mathtt{D})$, $(\mathtt{A}, \mathtt{E})$, and $(\mathtt{A}, \mathtt{F})$ have an endpoint in A, labeled with 1, and the other endpoint is a vertex labeled with a value larger than 1. Similarly, the *Cut*-value of vertex C is $Cut(\mathtt{C}, \phi, G) = 6$, computed by counting the appropriate number of edges (i.e., $(\mathtt{A}, \mathtt{D})$, $(\mathtt{A}, \mathtt{E})$, $(\mathtt{A}, \mathtt{F})$, $(\mathtt{B}, \mathtt{D})$, $(\mathtt{C}, \mathtt{D})$, and $(\mathtt{C}, \mathtt{F})$). Then, since the cutwidth of G, $Cut(\phi, G)$, is the maximum of the $Cut(v, \phi, G)$ for all vertices $v \in V$, in this example it is obtained that $Cut(\phi, G) = Cut(\mathtt{C}, \phi, G) = 6$.

In Fig. 6c, a different linear layout ϕ' of the graph G is shown. The associated *Cut*-values are depicted on Fig. 6d. Considering that the CMP is a minimiza-

tion problem, this second linear layout ϕ' is better than ϕ since $Cut(\phi', G) = Cut(\mathtt{D}, \phi', G) = 5$ which is smaller than $Cut(\phi, G) = Cut(\mathtt{C}, \phi, G) = 6$.

The CMP has been referred to in the literature with different names, such as Minimum Cut Linear Arrangement [23, 118] or Network Migration Scheduling [4,5]. There can also be found a generalization of the CMP for hyper-graphs named Board Permutation Problem [18, 19]. Several practical applications of this problem have been reported in different areas of engineering and computer science, such as circuit design [2, 19, 79], network reliability [57], information retrieval [14], automatic graph drawing [90, 114], protein engineering [7], or network migration [104]. This last application is, as far as the authors know, the newest one. It refers to the problem where internodal traffic from an obsolete telecommunication network needs to be migrated to a new network. This problem appears nowadays in phone traffic, where it is performed a migration between 4ESS switch-based networks to IP router-based networks [104]. Nodes are migrated, one at each time period, from the old to the new network. All traffic originating or terminating at a given node in the old network is moved to a specific node in the new network, with the objective of minimizing the connections, at the same time, between the old and the new networks.

In the scientific literature, the CMP has been addressed from both, exact and heuristic approaches. Most of the exact approaches present polynomial-time algorithms for particular graphs, such as hypercubes [51], trees [17, 120, 122], or grids [107]. There are also several exact methods for general graphs: integer linear programming formulations [76, 78] and Branch and Bound algorithms [83, 91]. These approaches can only solve relatively small instances. Therefore, different heuristic procedures have been proposed to address it. The work by Cohoon and Sahni [19] was the first approach proposing heuristic algorithms for the generalized version of the problem. These authors proposed several constructive methods and local search procedures, embedded into a Simulated Annealing metaheuristic [62]. Twenty years later, Andrade and Resende [4, 5] proposed a Greedy Randomized Adaptive Search Procedure (GRASP) [40, 41] hybridized with a Path Relinking method [46]. Later, Pantrigo et al. in [92] introduced a Scatter Search algorithm [45, 66], which outperformed previous methods. More recently, Pardo et al. first in [94] and later in [95] proposed several heuristics based on the Variable Neighborhood Search (VNS) methodology [87]. In particular, in [94] the authors applied several reduced VNS and basic VNS schemas to tackle the problem. Later, in [95] they applied a new variant of VNS, called Variable Formulation Search (VFS), to the CMP. This new algorithm is specially useful to deal with min-max (or max-min) optimization problems, where it is usual that many solutions have the same value of the objective function, which makes them especially hard for a heuristic search. VFS makes use of alternative formulations of the problem to determine which solution is more promising when they have the same value of the objective function in the original formulation. The latest proposal was performed by Duarte et al. in [32] by introducing different parallel designs of VNS to solve the CMP, which differs in the VNS stages to be parallelized as well as in the communication mechanisms among the processes. Experimental results show that the parallel implementation

outperforms previous methods in the state of the art for the CMP. This fact is also confirmed by non-parametric statistical tests. A recent approach to the problem [31] has also considered the CMP as a part of a multi-objective optimization problem.

As far as the set of instances used to evaluate this problem are concerned, most of the aforementioned authors have used the instances reported in [92]. In particular, authors proposed the use of three sets of instances: *small*, *grid*, and *Harwell-Boeing*. The *small* data set consists of 84 graphs introduced in the context of the Bandwidth Reduction Problem [80]. The number of vertices of the graphs in this set ranges from 16 to 24. The exact value of the CMP for the instances in this set is known, as they were solved to optimality in [83]. The *grid* data set consists of 81 matrices constructed as the Cartesian product of two paths [102], and they were originally introduced in the context of CMP by [107]. The size of the considered instances ranges from 9 vertices to 729. In this case, the optimal value is also known by construction [102]. Finally, the *Harwell-Boeing* data set is a subset of 87 instances extracted from the Harwell-Boeing sparse matrix collection which is accessible at the public domain matrix market library [20]. This collection was originally developed by Iain Duff, Roger Grimes, and John Lewis [34], and it consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines [35]. In the context of CMP, the instances selected have a number of vertices that ranges from 30 to 700. This data set is, in fact, the hardest among the previously defined for the algorithms in the state of the art.

The Minimum Linear Arrangement Problem

The Minimum Linear Arrangement Problem, usually known as MinLA, is also based on the *Cut*-value defined in section "The Cutwidth Minimization Problem." In this case, a linear layout with *load* equal to 1 is sought that minimizes the sum of the *congestion* on each position of the layout. It is equivalent to the minimization of the sum of *Cut*-values. More formally, given graph G and a linear layout ϕ, the linear arrangement value $LA(\phi, G)$ is defined as follows:

$$LA(\phi, G) = \sum_{v \in V} cut(v, \phi, G)$$

The optimum MinLA of G is then defined as the minimum $LA(\phi, G)$ over all possible layouts Φ of G.

Considering the linear layouts depicted in Fig. 6a, c it is shown in Fig. 7a, b the computation of the LA-values, respectively. In particular, the objective function value associated with the first layout ϕ is $LA(\phi, G) = 4 + 5 + 6 + 4 + 2 = 21$. Similarly, the objective function value associated with ϕ' is $LA(\phi', G) = 3 + 5 + 5 + 4 + 2 = 19$, Therefore, ϕ' is a better solution than ϕ since the MinLA is a minimization problem.

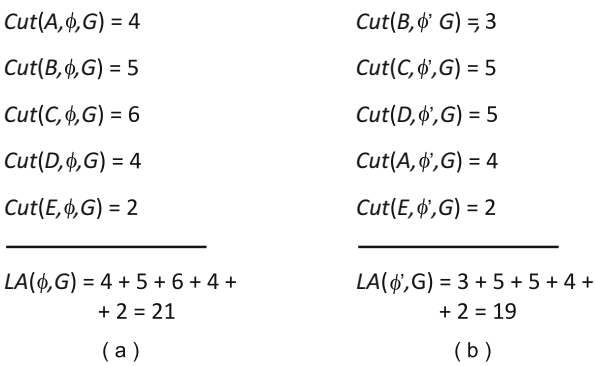


Fig. 7 (**a**) Computation of the objective function of the MinLA for the linear layout ϕ shown in Fig. 6a (**b**) Computation of the objective function of the MinLA for the linear layout ϕ' shown in Fig. 6c

The Minimum Linear Arrangement Problem is an NP-hard problem [42] and it is related to two other well-known layout problems: the Bandwidth and Profile minimization problems. However, as pointed out in [84], an optimal solution for one of these problems is not necessarily optimal for the other related problems. The Minimum Linear Arrangement Problem is also known as the Optimal Linear Ordering, Edge Sum, Minimum-1-Sum, Bandwidth Sum, or Wire-Length problem. It was originally introduced in [50], with the goal of designing error-correcting codes with minimal average absolute errors on certain classes of graphs. Twenty years later, this problem was considered in [86] as an oversimplified model of some nervous activity in the cortex. The MinLA also has applications in single machine job scheduling [1, 103] and in graph drawing [114].

Many different algorithms have been proposed for solving the MinLA problem. Juvan and Mohar, in [56], introduced the Spectral Sequencing method (SSQ). This method computes the eigenvectors of the Laplacian matrix of G. It then orders the vertices according to the second smallest eigenvector. As stated by Petit in [99], the rationale behind the SSQ heuristic is that vertices connected with an edge will tend to be assigned numbers that are close to each other, thus providing a good solution to the MinLA problem. McAllister proposed in [84] a heuristic method for the MinLA, which basically consists of a constructive procedure that labels vertices in a sequential order. Vertices are selected according to their degree with respect to previously labeled ones. This method compares favorably with previous methods for this and related problems. Later, Petit in [99] reviewed the existing lower bounds and heuristic methods, proposed new ones, and introduced a set of instances that were considered as the standard for this problem in the future, as it will be described ahead. Among the methods reviewed in [99], it is possible to find the Juvan-Mohar method [56], the Gomory-Hu tree method [2], and the edge method, which the author improved by adding a degree method. Petit concluded that the Juvan-Mohar method and the degree method provide the best lower bounds; however, their values are far from those of the best known solutions, and therefore they are of very limited interest from a practical point of view. In [99], Petit additionally introduced both a constructive and a local search procedure.

The Successive Augmentation (SAG) is a greedy heuristic that constructs a step-by-step solution, by extending a partial layout until all vertices have been enumerated. At each step, the best free label is assigned to the current vertex. Vertices are examined in the order given by a breadth-first search. Once a solution has been constructed, three different heuristics, based on local search, are considered: hill climbing, full search, and Simulated Annealing (SA). In the hill-climbing method, moves are selected at random; in the full search, the entire neighborhood of a solution is examined, at each iteration, in search of the best available move. Finally, the SA algorithm implements the temperature parameter as described in [62] for move selection. Petit, in [99], considered two neighborhoods, called *flip2* and *flip3*. The former exchanges the label of two vertices, while the latter "rotates" the label of three vertices. The experimentation in [99] shows that the neighborhood based on the exchange of two labels (*flip2*) produces better results than the rotation (*flip3*). Overall, the experimentation concludes that the SA method outperforms the others, although it employs much longer running times (not reported in the paper). Therefore, the author recommends employing the hill-climbing as well as the Spectral Sequencing methods. In [98], a more elaborate Simulated Annealing algorithm is proposed. The author introduces a new neighborhood, *flipN*, based on the normal distribution of the distances between the labels of vertices. The Simulated Annealing algorithm based on the *flipN* neighborhood (SAN) improves upon the previous SA method. Moreover, to speed up the method, the initial solution is obtained with the SSQ algorithm. The combined method, SSQ + SAN, is able to outperform previous methods.

In [105], the authors proposed a new algorithm based on the Simulated Annealing methodology. The Two-Stage Simulated Annealing (TSSA) performs two steps. In the first one, a solution is constructed with the procedure described in [84]; then, in the second step, it performs a Simulated Annealing procedure based on exchanges of labels. This method introduces two new elements to solve the MinLA: a combined neighborhood and a new evaluation function. Given a vertex v, the first neighborhood, selected with a probability of 0.9, examines the vertices u such that their label $\phi(u)$ is close to the median of the labels of the vertices adjacent to v (at a maximum distance of 2). The second neighborhood, selected with a probability 0.1, exchanges the labels of two vertices selected at random with diversification (exploration) purposes. The method evaluates the solutions with a function that is more discriminating than the original one. The authors compared their TSSA method with the best known algorithms, and they concluded that their method outperforms the previous algorithms. This method has been the reference of the state of the art for many years. More recent approaches include the one by Martí et al. [82] with an algorithm based on a Scatter Search with Path Relinking and the one proposed by Mladenovic et al. [88], with a Skewed General Variable Neighborhood Search (SGVNS), which is, as far as the authors know, the latest proposal for the problem. The method presented in [88] starts from a greedy-constructed solution as an initial point. The authors propose the use of a local search, based on a four different neighborhood structures. These neighborhoods are explored with a Variable Neighborhood Descent (VND) procedure. Then, the VND, together with

a shake procedure, based on random moves within the four neighborhoods defined, conform a General VNS (GVNS) method. This method is finally extended by the use of several ideas from Skewed VNS where the sharp move acceptance criterion is relaxed. However, the procedures proposed in [82] and [88] were not able to outperform the results by the TSSA presented in [105].

As aforementioned, there is a set of instances considered as standard as the comparison framework in the literature for the MinLA. In particular, this set was introduced by Petit et al. [99] and it consists of 21 graphs ($62 \leq n \leq 10240$) being n the number of vertices of the graph. As the authors indicate in [99], choosing a particular class of graphs to serve as benchmark for the MinLA problem is difficult, because no real large instances for this problem exist. The aim of the authors, when they formed the data set, was to measure the heuristics in sparse graphs, so, therefore, they selected different kinds of graphs belonging to the following families: general random graphs, geometric random graphs, graphs with known optima (trees, hypercubes, meshes), graphs from finite element discretizations, graphs from VLSI design, and graphs from graph drawing competitions.

The Vertex Separation Problem

The Vertex Separation Problem (VSP) was originally motivated by the idea of finding good separators for graphs [74]. In particular, the VSP consists in finding a linear layout (*load* equals to 1) where the separator is minimized. In mathematical terms, being p a position in the layout, the set $L(p, \phi, G) = \{u \in V : \phi(u) \leq p\}$ is defined, which represents the set of vertices in the linear layout placed in a position lower than or equal to p. Symmetrically, the set $R(p, \phi, G) = \{u \in V : \phi(u) > p\}$ contains those vertices placed in a position larger than p. In general, $L(p, \phi, G)$ can be simply referred to as the set of *left vertices* with respect to position p in ϕ, and $R(p, \phi, G)$ is the set of the *right vertices* with respect to p in ϕ. The separation value at position p of layout ϕ, $Sep(p, \phi, G)$, is defined as the number of vertices in $L(p, \phi, G)$ with one or more adjacent vertices in $R(p, \phi, G)$. More formally:

$$Sep(p, \phi, G) = |\{u \in L(p, \phi, G) : \exists\, v \in R(p, \phi, G) \wedge (u, v) \in E\}|.$$

The vertex separation (VS) value of a layout ϕ is the maximum of the *Sep*-values among all positions in ϕ:

$$VS(\phi, G) = \max_{1 \leq p < n} Sep(p, \phi, G)$$

The Vertex Separation Problem (VSP) then consists in finding a layout, $\phi^\star$, that minimizes the *VS*-value of the graph G. In mathematical terms:

$$\phi^\star = \arg\min_{\phi \in \Phi} VS(\phi, G)$$

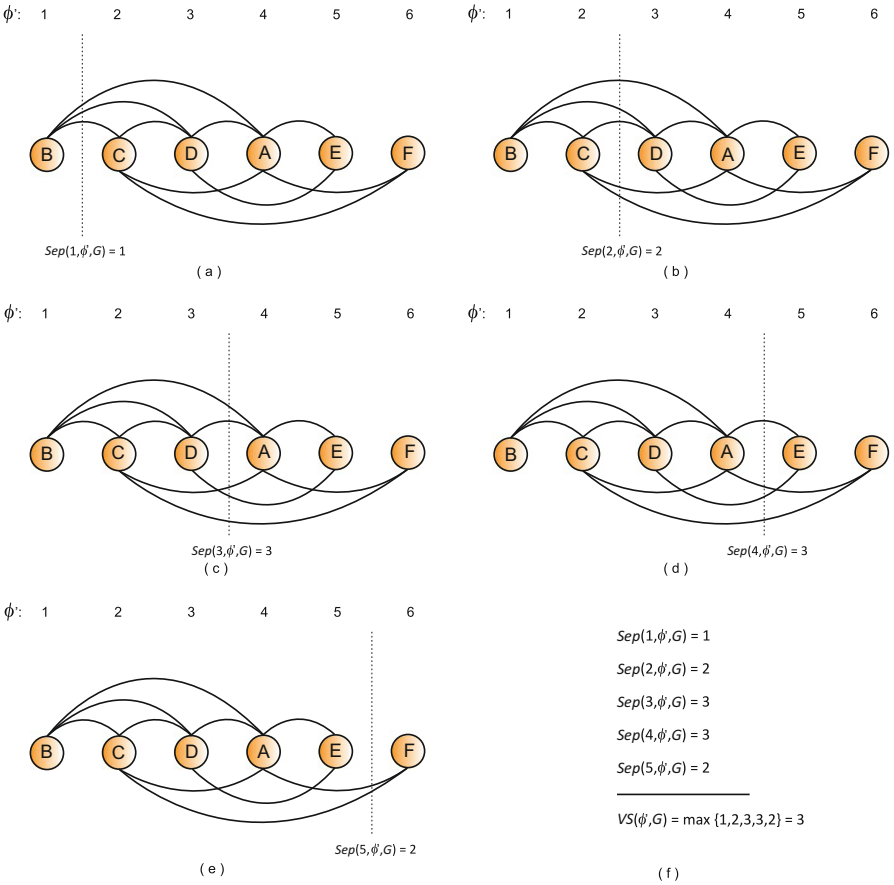


Fig. 8 (**a**)–(**e**) Computation of $Sep(p, \phi', G)$, with $p = \{1, \ldots, 5\}$, for the layout $\phi' = \{\texttt{B}, \texttt{C}, \texttt{D}, \texttt{A}, \texttt{E}, \texttt{F}\}$ introduced in Fig. 6c. (**f**) Computation of $VS(\phi', G)$

where Φ represents the set of all possible linear layouts of G.

Considering the graph example introduced in Fig. 2 and the layout ϕ' shown in Fig. 6c, in Fig. 8a–e, the *Sep*-value of each position p in layout ϕ' is depicted, denoted as $Sep(p, \phi', G)$. For instance, $Sep(1, \phi', G) = 1$ (see Fig. 8a) because $L(1, \phi', G) = \{\texttt{B}\}$ and $R(1, \phi', G) = \{\texttt{C}, \texttt{D}, \texttt{A}, \texttt{E}, \texttt{F}\}$, and there is only one vertex in L having an adjacent vertex in R. Similarly, $Sep(4, \phi', G) = 3$ (see Fig. 8d) because $L(4, \phi', G) = \{\texttt{B}, \texttt{C}, \texttt{D}, \texttt{A}\}$ and $R(4, \phi', G) = \{\texttt{E}, \texttt{F}\}$, and the vertices C, D, and A in L have an adjacent vertex in R. Notice that vertex B does not affect the Sep-value, since it does not have adjacent vertices in R. The objective function is then computed as the maximum of these *Sep*-values. In particular, this value is $VS(G, \phi') = 3$, associated with positions 3 and 4. For the sake of completeness, all the Sep-values as well as the objective function value are shown in Fig. 8f.

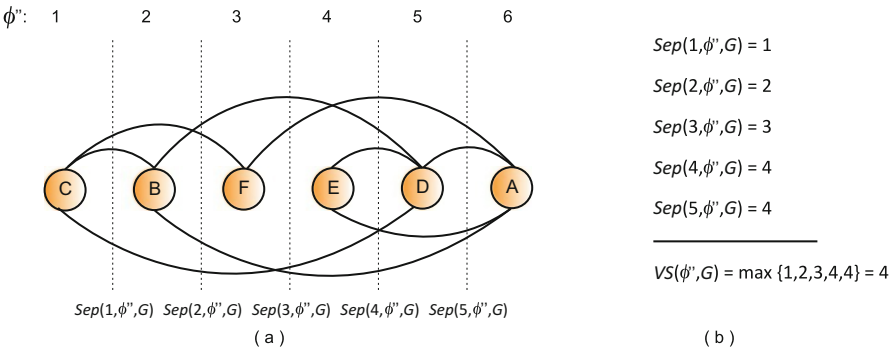


Fig. 9 (**a**) Computation of $Sep(p, \phi'', G)$ with $p = \{1, \ldots, 5\}$, for the layout $\phi'' = \{\mathtt{C}, \mathtt{B}, \mathtt{F}, \mathtt{E}, \mathtt{D}, \mathtt{A}\}$. (**b**) Computation of $VS(\phi'', G)$

As in the previous optimization problems defined above, a second example of layout ϕ'' (see Fig. 9a, b) is included. The objective function value of this solution is 4.

The VSP is strongly related to other well-known graph problems, such as the Path Width [61], the Node Search Number [64], or the Interval Thickness [63], among others. The description of the equivalences among these problems can be found in [39,61,64]. All these optimization problems are NP-hard and have practical applications in VLSI design [67], computer language compiler design [9], or graph drawing [36].

It is possible to find efficient exact approaches to solve the VSP on special classes of graphs. A linear algorithm to compute the optimal vertex separation of a tree is proposed in [37] as well as an $O(n \log n)$ algorithm for finding the corresponding optimal layout. The algorithm was improved in [116] with a linear time procedure to find the optimal layout. In [97], an alternative method to compute the vertex separation of trees was also proposed, and in [38] an $O(n \log n)$ algorithm computes the vertex separation of unicyclic graphs (i.e., trees with an extra edge). A polynomial-time algorithm to compute the Path Width (which is identical to VSP) is proposed in [11]. However, the algorithm cannot be considered from a practical point of view, since the bound on its time complexity is $\Omega(n)$; see [38]. In [13] a polynomial-time algorithm for optimally solving the VSP for n-dimensional grids is proposed. Co-graphs and permutational graphs can also be optimally solved as it was stated in [10–12]. Approximation algorithms have been also proposed for the VSP. Specifically, [8] proposes an $O(\log^2 n)$-approximation algorithm for general graphs and a $O(\log n)$-approximation algorithm for planar graphs. Similar results for binomial random graphs are presented in [26]. The first heuristic was proposed by Duarte et al. in [30] where the authors proposed a Basic VNS for the VSP. In particular, the authors presented two constructive procedures and one local search based on interchange moves. As far as the authors know, this algorithm obtains the best results in this problem. More recently, Sánchez-Oro et al. [113] proposed three

VNS algorithms: Reduced VNS (RVNS), VND, and General VNS (GVNS). RVNS mainly focuses on the diversification, VND mainly focuses on the intensification, and GVNS balances intensification and diversification. Computational experiments reveal that the best procedure (GVNS) improves the state of the art in both quality and computing time. This fact is confirmed with non-parametric statistical tests.

As far as the set of instances used to evaluate the heuristic algorithms are concerned, the first set of instances for the problem was defined in [30]. In this case, the authors selected 173 instances divided in three different subsets as a test bed. In particular, two of the subsets were formed by instances with a defined structure (trees and grids). For those instances, the optimum value is known by definition for the VSP. Additionally, they also selected an additional subset of random graphs, whose instances have previously been used in the literature to evaluate other linear layout optimization problems. The grid subset consists of 50 matrices constructed as the Cartesian product of two paths. The number of vertices of this subset ranges from 25 to 2916. The tree subset is formed by 50 instances where the number of vertices ranges from 22 to 202. Finally, the random graph subset was formed by 73 instances derived from the Harwell-Boeing sparse matrix collection [20, 34, 35] introduced at the end of section "The Cutwidth Minimization Problem." In this case, the size of the instances, in terms of the number of vertices, ranges from 24 to 960. The aforementioned set of instances has been used later in [109] and [113].

The SumCut/Profile Problem

Considering the separation value defined in the previous section, the SumCut of a graph G with respect to the linear layout ϕ, denoted as $SC(\phi, G)$, is calculated as the sum of Sep-values of each position. In mathematical terms:

$$SC(\phi, G) = \sum_{i=1}^{n} Sep(i, \phi, G)$$

The SumCut Minimization Problem then consists in minimizing the value of $SC(\phi, G)$ over all possible linear layouts Φ of G:

$$SC(G) = \min_{\phi \in \Phi} SC(\phi, G)$$

Considering the linear layouts ϕ' and ϕ'' depicted in Fig. 8a and 9a, respectively, in Fig. 10a, b the computation of the SC-values associated to each layout is shown. In particular, the objective function value associated with the first layout ϕ' is $SC(\phi', G) = 1 + 2 + 3 + 3 + 2 = 11$. Similarly, the objective function value associated with ϕ'' is $SC(\phi'', G) = 1 + 2 + 3 + 4 + 5 = 14$. Therefore, ϕ' is a better solution than ϕ'' since the SumCut is a minimization problem.

The SumCut was proved to be NP-complete for co-bipartite graphs in [123] and for general graphs in [24, 49]. This optimization problem is equivalent to the Profile

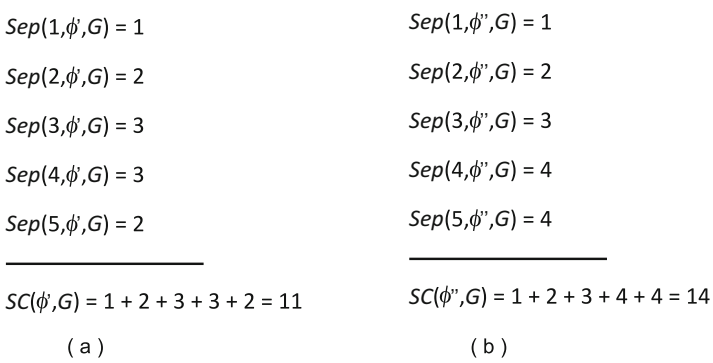


Fig. 10 (**a**) Computation of the objective function of SC for the linear layout shown in Fig. 8a. (**b**) Computation of the objective function of SC for the linear layout shown in Fig. 9a

Minimization Problem, as it was stated in [103]. Specifically, the reverse solution of the SumCut corresponds to a solution of the Profile. Both optimization problems have been extensively studied. See, for instance, [25–27]. Practical applications of these problems appear in genetics. The goal of the Human Genome Project consist in sequencing the DNA of humans, as well as other species, with the target of elucidating the genetic information contained therein. In order to construct a physical map of a large DNA molecule, it is necessary to extract clones from it. Then, a fingerprint of each clone is obtained. Finally, DNA molecule is reassembled determining how the clones overlap among them. Each clone is a sequence of nucleotides drawn from the set {adenine, cytosine, guanine, thymine}, so the reassembly process consists in permuting a linear layout of a graph. In [59] the author described an application in archeology, where it is necessary to serialize different artifacts (fossil, hardware, jewels, etc.). The serialization is known in archeology as "seriation" and consists in placing in chronological order different artifacts in the same culture using a relative dating method. Specifically, the practical application is based on the rearrangement of a matrix, which can be translated on the reordering of a linear layout of a graph.

Reducing the profile of a matrix is a relevant problem in mathematics since it leads to a reduction of the amount of space needed for some storage scheme. On the whole, it achieves an improvement of the performance of several operations such as Choleski factorization of non-singular systems of equations [108]. Recently the profile reduction has been used in new areas like information retrieval to browse hypertext [14]. The Profile Minimization Problem was originally proposed as a way to reduce the storage space needed to save a sparse matrix [119], but it was proved that it is equivalent to the SumCut Minimization Problem [103]. An important application of the Profile Minimization Problem arises in clone fingerprinting [58].

As far as the heuristics are concerned, Cuthill and McKee proposed in [21] the reverse Cuthill-McKee (RCM) algorithm, in order to get the minimum profile of a graph. In order to obtain a solution for the SumCut Minimization Problem, it is only

needed to reverse the solution generated. Gibbs et al. in [44] solved the SumCut using a new algorithm based on the RCM. The paper describes three problems of the RCM and presents a new algorithm that solves the described problems. Lewis described a method to reorder sparse matrices in order to reduce their profile [68], using the Gibbs-King algorithm to improve the results from the RCM algorithm. The previous algorithms were used not only in Profile and SumCut problems but also in the Bandwidth Minimization Problem, making a small adjustment in the last step: the numbering of the nodes. Lewis, in [69] presented a Simulated Annealing to reduce the profile of a matrix. The algorithm starts with a previously calculated solution and it improves the solution by using the Simulated Annealing technique. The original solution is calculated using either the RCM algorithm or the Gibbs-King algorithm, and the instances used are a subset of the Harwell-Boeing sparse matrix collection [20, 34, 35]. In [111] a new heuristic method based on the GRASP methodology is introduced. As it is shown in the computational experience, the proposed method is competitive with respect to previous approaches. This algorithm is further improved in [110] by coupling it with a Path Relinking post-optimization stage. The best heuristic procedure identified in the state of the art is presented by Sánchez-Oro et al. in [112]. This method is based on the Scatter Search methodology. Among several mechanisms, this procedure includes Path Relinking as the basis for combining solutions to generate new ones. Extensive computational experiments show that this method clearly outperforms previous approaches in terms of both solution quality and computing time.

The current reference instances to evaluate the performance of the heuristic algorithms for this problem were proposed in [112] which is, as far as we know, the most complete data set of instances for the problem. In particular they propose the use of the three different subsets of instances: Harwell-Boeing, bipartite, and tree. The Harwell-Boeing subset was formed by 73 sparse matrices whose number of vertices ranges from 24 to 960, extracted from the well-known collection Harwell-Boeing sparse matrix collection [20, 34, 35]. Notice that instances belonging to this family had been used before in [21] as a test bed for the problem. The bipartite subset is formed by 98 bipartite graphs with a number of vertices ranging from 4 to 142. A complete bipartite graph is such that the set of vertices V can be divided into two subsets V_1 and V_2 in such a way that there exists an edge between every pair of vertices belonging to different subsets. Finally, the tree subset consists of 91 instances based on trees with diameter 4 and a number of vertices ranging from 10 to 100. Optimal objective function values are known by construction for the instances in bipartite and tree subsets.

The Bandwidth Minimization Problem

The Bandwidth Minimization Problem (BMP), also known as Bandwidth Reduction Problem or Matrix Bandwidth Minimization, is an NP-hard min-max linear layout problem [93]. It consists in finding an ordering of the vertices of a graph on a line, in such a way that the largest distance between two adjacent vertices in the

corresponding line is minimized. In terms of graph embedding theory, this problem consists in finding the linear layout (*load* equals to 1) that minimizes the *dilation*. In the terms of circuit design, the BMP asks for a layout that minimizes the maximum edge length between each pair of connected modules.

Given a graph G and a linear layout ϕ, the bandwidth of a vertex v, $B(v, \phi, G)$, is the maximum of the differences between $\phi(v)$ and the labels of its adjacent vertices. That is:

$$B(v, \phi, G) = \max_{(u,v)\in E} \{|\phi(v) - \phi(u)|\}$$

The bandwidth of a graph G with respect to a layout ϕ is then

$$B(\phi, G) = \max_{v\in V} B(v, \phi, G)$$

The optimal bandwidth of graph G is thus the minimum $B(\phi, G)$ value over all possible layouts Φ of G.

Considering the linear layouts ϕ and ϕ' depicted in Fig. 6a, c respectively, in Fig. 11a, b the computation of their bandwidth value is shown. In particular, the objective function value associated with the first layout ϕ is $B(\phi, G) = \max\{5, 2, 3, 3, 4, 5\} = 5$. Similarly, the objective function value associated with ϕ' is $B(\phi', G) = \max\{3, 3, 4, 2, 2, 4\} = 4$. Therefore, ϕ' is a better solution than ϕ since the BMP is a minimization problem.

This problem has been also defined in terms of matrices. In particular, given the incidence matrix of graph G, the problem consists in finding a permutation of the rows and the columns of this matrix that keeps all the non-zero elements in a band

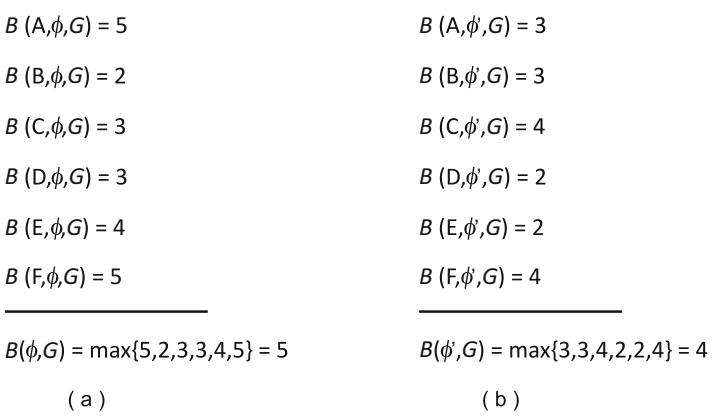


Fig. 11 (**a**) Computation of the BMP objective function for the linear layout ϕ shown in Fig. 6a. (**b**) Computation of the BMP objective function for the linear layout ϕ' shown in Fig. 6c

that is as close as possible to the main diagonal. For that reason it is possible to find references to this problem as the Matrix Bandwidth Minimization problem.

The main application of the BMP is to solve non-singular systems of linear algebraic equations. The preprocessing of the coefficient matrix to reduce its bandwidth results in substantial savings in the computational effort associated with solving the system of equations. The context of these applications includes aircraft structures, liquid nitrogen gas tanks, propel blades, and submarines. In order to tackle the problem, Del Corso and Manzini in [22] proposed two exact algorithms to solve randomly generated graphs up to 100 nodes. Similarly, Martí et al. proposed an exact method in [80] based on a branch and bound algorithm, together with several lower bounds, to compute the optimal solution for small and medium-sized instances. However, the complexity of the problem makes that most of the efforts performed by researchers have been heuristic approaches. In this sense, for many years, researchers were only interested in designing relatively simple heuristic procedures and sacrificed solution quality for speed. This is the case of the reverse method by Cuthill and McKee [21] and the GPS procedure by Gibbs et al. [44]. These two methods yield similar results in terms of solution quality; however, GPS is considerably faster, with an average speed that is about eight times faster than the reverse Cuthill-McKee procedure. Recently, metaheuristics have been adapted to this problem. A Simulated Annealing procedure was introduced by Dueck and Jeffs in [33], which is based on an insertion mechanism and does not take advantage of the graph structure as the GPS method does. Martí et al. in [81] propose a Tabu Search method [47] for this problem, which is likewise based on swap moves that exchange the labels of a pair of vertices; however, it incorporates memory structures that prove to be remarkably effective. Later, Piñana et al. in [100] proposed an algorithm based on GRASP for the problem. In this proposal, the constructive step is based on GPS and the local search is based on exchanges. The GRASP method is coupled with a Path Relinking phase for improved outcomes. This algorithm clearly outperforms all the previous heuristic approaches. More recently, Lim proposed several algorithms for the problem. In [70] they proposed an Ant Colony Optimization (ACO) [29] method with hill climbing. The same year, in [72] they introduced the node-shift heuristic which computes the desired label of each vertex according to the label of its adjacent vertices and then orders all the vertices in the graph with respect to these desired labels; finally all the vertices are relabeled following this ordering. This innovative method is repeated until no vertex changes its label and it is coupled with a local search hill climbing. These authors also proposed a Genetic Algorithm [48, 52] that generates the solutions in the initial population with a level structure procedure (as the GPS does) and implements a classic midpoint crossover as a combination operator. Finally, among their proposals, in [71], Lim et al. proposed a Particle Swarm Optimization (PSO) [60] and hill climbing method. Later, Rodríguez-Tello et al. in [106] presented a Simulated Annealing method based on a new neighborhood definition. Instead of swapping the labels of two vertices, they introduce a more elaborated move definition that leads to an efficient search as shown in their computational results. Recent approaches to the problem include several strategies based on Variable

Neighborhood Search proposed by Mladenovic et al. in [89]; an adaptation of Tabu Search and Scatter Search performed by Campos et al. in [15], where the authors studied the influence of adaptive memory; and the Genetic Algorithm proposed by Pop and Matei in [101].

The test data set of instances used as a reference for the problem is again derived from the Harwell-Boeing sparse matrix collection [20, 34, 35]. The use of instances derived from this family can be traced back to the first papers in the literature with heuristic approaches. In particular, the current data set is composed of 113 instances and it is divided into two subsets. The first subset consists of 33 instances with a number of vertices ranging from 30 to 199, and the second subset includes 80 instances with a number of vertices ranging from 200 to 1000. This data set was first introduced in [81], where the authors selected 126 instances and then it was reduced to its current state in [100] by considering just 113 instances. The data set has been used in almost all the papers in the latter literature of the BMP.

Conclusion

In this chapter a family of graph layout problems is reviewed. Graph layout problems are optimization problems where the main objective is to project an original graph into a predefined host graph. The term layout problem comes from its application to VLSI design. In particular, five linear layout problems are reviewed, where the host graph is a horizontal line and the aim is to minimize an objective function. This family is probably the most studied family of problems by the heuristic literature within the context of graph layout problems. Specifically, in this chapter the following are reviewed: the Cutwidth Minimization Problem, the Minimum Linear Arrangement Problem, the Vertex Separation Problem, the SumCut Problem, and the Bandwidth Minimization Problem. Each problem is illustrated with its formal definition and with a detailed example. Additionally, the most relevant heuristic methods in the literature and the sets of instances used to evaluate them are also reviewed. It is worth mentioning that, among other data sets, including those with graphs where the optimum value is known for each problem, all the studied problems have been tested over instances from the Harwell-Boeing sparse matrix collection. This collection of graphs can be considered as a reference test bed for linear layout problems, since the hardness of the instances has made them suitable to find differences among the algorithms.

The wide variety of heuristic algorithms presented in the literature for the reviewed problems set a deep understanding of strategies to reach high-quality solutions to minimization linear layout problems. This knowledge can be then applied to other linear layout problems. Despite of the fact that there are many algorithms able to find remarkable upper bounds for the studied problems, the gap between the lower and the upper bounds for general graphs is still large. Most of the exact approaches are not able to handle graphs larger than 100 vertices, while current heuristics are successfully handling graphs up to 1000 vertices. Therefore,

among the open questions, the development of efficient algorithms to get better lower bounds would be interesting, as well as testing the current heuristics on larger graphs. Finally, the attention in the future should be also derived to extend the current knowledge on the embedding of graphs into a linear structure, to other structures such as grids, trees, or cycles.

Cross-References

▸ Network Optimization
▸ Scheduling Heuristics

Acknowledgments This research has been partially supported by Fondo Europeo de Desarrollo Regional (FEDER) and Ministerio de Economía y Competitividad (MINECO) of Spain. Grant Ref. TIN2015-65460-C2 (MINECO/FEDER).

References

1. Adolphson DL (1977) Single machine job sequencing with precedence constraints. SIAM J Comput 6:50–54
2. Adolphson DL, Hu TC (1973) Optimal linear ordering. SIAM J Appl Math 25(3):403–423
3. Álvarez C, Cases R, Díaz J, Petit J, Serna M (2000) Approximation and randomized algorithms in communication networks. In: Routing trees problems on random graphs. ICALP workshops 2000. Carleton Scientific, p 99111
4. Andrade DV, Resende MGC (2007) GRASP with evolutionary path-relinking. In: Seventh metaheuristics international conference (MIC), Montreal
5. Andrade DV, Resende MGC (2007) GRASP with path-relinking for network migration scheduling. In: Proceedings of international network optimization conference (INOC), Spa
6. Bezrukov SL, Chavez JD, Harper LH, öttger MR, Schroeder UP (2000) The congestion of n-cube layout on a rectangular grid. Discret Math 213:13–19
7. Blin G, Fertin G, Hermelin D, Vialette S (2008) Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. J Discret Algorithms 6:618–626
8. Bodlaender HL, Gilbert JR, Hafsteinsson H, Kloks T (1995) Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J Algorithms 18(2):238–255
9. Bodlaender HL, Gustedt J, Telle JA (1998) Linear-time register allocation for a fixed number of registers. In: Proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms (SODA'98). Society for Industrial and Applied Mathematics, Philadelphia, pp 574–583
10. Bodlaender HL, Kloks T, Kratsch D (1993) Treewidth and pathwidth of permutation graphs. In: Lingas A, Karlsson R, Carlsson S (eds) Automata, languages and programming. Lecture notes in computer science, vol 700. Springer, Berlin/Heidelberg, pp 114–125
11. Bodlaender HL, Kloks T, Kratsch D (1995) Treewidth and pathwidth of permutation graphs. SIAM J Discret Math 8(4):606–616
12. Bodlaender HL, Möhring RH (1990) The pathwidth and treewidth of cographs. SIAM J Discret Math 6(6):181–188
13. Bollobás B, Leader I (1991) Edge-isoperimetric inequalities in the grid. Combinatorica 11(4):299–314
14. Botafogo RA (1993) Cluster analysis for hypertext systems. In: 16th annual international ACM-SIGIR conference on research and development in information retrieval, Pittsburgh, PA, USA, pp 116–125

15. Campos V, Piñana E, Martí R (2011) Adaptive memory programming for matrix bandwidth minimization. Ann Oper Res 183(1):7–23
16. Chinn PZ, Chvátalová J, Dewdney AK, Gibbs NE (1982) The bandwidth problem for graphs and matricesa survey. J Graph Theory 6(3):223–254
17. Chung MJ, Makedon F, Sudborough IH, Turner J (1982) Polynomial time algorithms for the min cut problem on degree restricted trees. In: Proceedings of the 23rd annual symposium on foundations of computer science (SFCS'82). IEEE Computer Society, Washington, DC, pp 262–271
18. Cohoon J, Sahni S (1983) Exact algorithms for special cases of the board permutation problem. In: Proceedings of the allerton conference on communication, control and computing, Monticello, pp 246–255
19. Cohoon J, Sahni S (1987) Heuristics for backplane ordering. J VLSI Comput Syst 2:37–61
20. Harwell-Boeing Sparse Matrix Collection. Public domain matrix market (2016). http://math.nist.gov/matrixmarket/data/harwell-boeing/
21. Cuthill E, McKee J (1969) Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th national conference. ACM, New York, pp 157–172
22. Corso GMD, Manzini G (1999) Finding exact solutions to the bandwidth minimization problem. Computing 62(3):189–203
23. Díaz J, Gibbons A, Pantziou GE, Serna MJ, Spirakis PG, Torán J (1997) Parallel algorithms for the minimum cut and the minimum length tree layout problems. Theor Comput Sci 181:267–287
24. Díaz J, Gibbons A, Paterson M, Torán J (1991) The minsumcut problem. In: Dehne F, Sack J-R, Santoro N (eds) WADS. Lecture notes in computer science, vol 519. Springer, Berlin/Heidelberg, pp 65–89
25. Díaz J, Penrose MD, Petit J, Serna MJ (2000) Convergence theorems for some layout measures on random lattice and random geometric graphs. Comb Probab Comput 9: 489–511
26. Díaz J, Petit J, Serna MJ (2002) A survey of graph layout problems. ACM Comput Surv (CSUR) 34(3):313–356
27. Díaz J, Petit J, Serna MJ, Trevisan L (1998) Approximating layout problems on random sparse graphs. Technical report LSI-98-44-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (Presented in the fifth Czech-Slovak international symposium on combinatorics, graph theory, algorithms and applications)
28. Ding G, Oporowski B (1995) Some results on tree decomposition of graphs. J Graph Theory 20(4):481–499
29. Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Politecnico di Milano, Italia
30. Duarte A, Escudero LF, Martí R, Mladenović N, Pantrigo JJ, Sánchez-Oro J (2012) Variable neighborhood search for the vertex separation problem. Comput Oper Res 39(12): 3247–3255
31. Duarte A, Pantrigo JJ, Pardo EG, Mladenović N (2015) Multi-objective variable neighborhood search: an application to combinatorial optimization problems. J Glob Optim 63(3):515–536
32. Duarte A, Pantrigo JJ, Pardo EG, Sánchez-Oro J (2016) Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J Manag Math 27(1):55–73
33. Dueck G, Jeffs J (1995) A heuristic bandwidth reduction algorithm. J comb math comput 18:97–108
34. Duff IS, Grimes RG, Lewis JG (1989) Sparse matrix test problems. ACM Trans Math Softw 15(1):1–14
35. Duff IS, Grimes RG, Lewis JG (1992) User's guide for the harwell-boeing sparse matrix collection (release I), Rutherford Appleton Laboratory, Chilton
36. Dujmovic V, Fellows MR, Kitching M, Liotta G, McCartin C, Nishimura N, Ragde P, Rosamond F, Whitesides S, Wood DR (2008) On the parameterized complexity of layered graph drawing. Algorithmica 52(2):267–292

37. Ellis JA, Sudborough IH, Turner JS (1994) The vertex separation and search number of a graph. Inf Comput 113(1):50–79
38. Ellis JA, Markov M (2004) Computing the vertex separation of unicyclic graphs. Inf Comput 192(2):123–161
39. Fellows MR, Langston MA (1994) On search, decision, and the efficiency of polynomial-time algorithms. J Comput Syst Sci 49(3):769–779
40. Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67–71
41. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6(2):109–133
42. Garey MR, Johnson DS (1979) Computers and intractability: a Guide to the theory of np-completeness. W.H. Freeman, San Francisco
43. Gavril F (1977) Some NP-complete problems on graphs. In: Proceedings of the eleventh conference on information sciences and systems, Baltimore, pp 91–95
44. Gibbs NE, Poole WG Jr, Stockmeyer PK (1976) An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM J Numer Anal 13(2):236–250
45. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(1):156–166
46. Glover F (1997) Tabu search and adaptive memory programming—advances, applications and challenges. Springer, Boston, pp 1–75
47. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Norwell
48. Golberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addion wesley 1989:102
49. Golovach P (2009) The total vertex separation number of a graph. Discret Math Appl 6(7):631–636
50. Harper LH (1964) Optimal assignments of numbers to vertices. J Soc Ind Appl Math 12(1):131–135
51. Harper LH (1966) Optimal numberings and isoperimetric problems on graphs. J Comb Theory 1:385–393
52. Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Ann Arbor
53. Hu TC (1974) Optimum communication spanning trees. SIAM J Comput 3(3):188–195
54. Johnson DS, Lenstra JK, Kan AHGR (1978) The complexity of the network design problem. Networks 8(4):279–285
55. Juvan M, Marincek J, Mohar B (1995) Embedding a graph into the torus in linear time. Springer, Berlin/Heidelberg, pp 360–363
56. Juvan M, Mohar B 1992 Optimal linear labelings and eigenvalues of graphs. Discret Appl Math 36:153–168
57. Karger DR (1999) A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. SIAM J Comput 29(2):492–514
58. Karp RM (1993) Mapping the genome: some combinatorial problems arising in molecular biology. In: Proceedings of the twenty-fifth annual ACM symposium on theory of computing. ACM, New York, pp 278–285
59. Kendall D (1969) Incidence matrices, interval graphs and seriation in archeology. Pac J math 28(3):565–570
60. Kennedy J Eberhart R (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks, Yokohama, vol 4, pp 1942–1948
61. Kinnersley NG (1992) The vertex separation number of a graph equals its path-width. Inf Process Lett 42(6):345–350
62. Kirkpatrick S, Gelatt CD, Vecchi MP 1983 Optimization by simulated annealing. Science 220(4598):671–680
63. Kirousis LM, Papadimitriou CH (1985) Interval graphs and searching. Discret Math 55(2):181–184

64. Kirousis LM, Papadimitriou CH (1986) Searching and pebbling. Theor Comput Sci 47(0):205–218
65. Klugerman M, Russell A, Sundaram R (1998) On embedding complete graphs into hypercubes. Discret Math 186(13):289–293
66. Laguna M, Martí R (2012) Scatter search: methodology and implementations in C, vol 24. Springer Science & Business Media, New York
67. Leiserson CE (1980) Area-efficient graph layouts (for vlsi). In: IEEE symposium on foundations of computer science, Syracuse, pp 270–281
68. Lewis JG (1982) The gibbs-poole-stockmeyer and gibbs-king algorithms for reordering sparse matrices. ACM Trans Math Softw (TOMS) 8(2):190–194
69. Lewis RR (1994) Simulated annealing for profile and fill reduction of sparse matrices. Int J Numer Methods Eng 37(6):905–925
70. Lim A, Lin J, Rodrigues B, Xiao F (2006) Ant colony optimization with hill climbing for the bandwidth minimization problem. Appl Soft Comput 6(2):180–188
71. Lim A, Lin J, Xiao F (2007) Particle swarm optimization and hill climbing for the bandwidth minimization problem. Appl Intell 26(3):175–182
72. Lim A, Rodrigues B, Xiao F (2006) Heuristics for matrix bandwidth reduction. Eur J Oper Res 174(1):69–91
73. Lin YX (1994) Two-dimensional bandwidth problem. In: Combinatorics, graph theory, algorithms and applications 1993, Beijing, pp 223–232
74. Lipton RJ, Tarjan RE (1979) A separator theorem for planar graphs. SIAM J Appl Math 36(2):177–189
75. Livingston M, Stout QF (1988) Embeddings in hypercubes. Math Comput Model 11(0): 222–227
76. López-Locés MC, Castillo-García N, Huacuja HJF, Bouvry P, Pecero JE, Rangel RAP, Barbosa JJG, Valdez F (2014) A new integer linear programming model for the cutwidth minimization problem of a connected undirected graph. In: Recent advances on hybrid approaches for designing intelligent systems. Springer, Cham, pp 509–517
77. Lozano M, Duarte A, Gortázar F, Martí R (2013) A hybrid metaheuristic for the cyclic antibandwidth problem. Knowl-Based Syst 54:103–113
78. Luttamaguzi J, Pelsmajer M, Shen Z, Yang B (2005) Integer programming solutions for several optimization problems in graph theory. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, DIMACS
79. Makedon F Sudborough IH (1989) On minimizing width in linear layouts. Discret Appl Math 23(3):243–265
80. Martí R, Campos V, Piñana E (2008) A branch and bound algorithm for the matrix bandwidth minimization. Eur J Oper Res 186(2):513–528
81. Martí R, Laguna M, Glover F, Campos V (2001) Reducing the bandwidth of a sparse matrix with tabu search. Eur J Oper Res 135(2):450–459
82. Martí R, Pantrigo JJ, Duarte A, Campos V, Glover F (2011) Scatter search and path relinking: a tutorial on the linear arrangement problem. Int J Swarm Intell Res (IJSIR) 2(2): 1–21
83. Martí R, Pantrigo JJ, Duarte A, Pardo EG (2013) Branch and bound for the cutwidth minimization problem. Comput Oper Res 40(1):137–149
84. Mcallister AJ (1999) A new heuristic algorithm for the linear arrangement problem. Technical report, University of New Brunswick
85. Miller Z, Sudborough IH 1991 A polynomial algorithm for recognizing bounded cutwidth in hypergraphs. Theory Comput Syst 24:11–40
86. Mitchison G, Durbin R (1986) Optimal numberings of an n × n array. Algebr Discret Methods 7(4):571–582
87. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
88. Mladenović N, Urošević D, Pérez-Brito D (2014) Variable neighborhood search for minimum linear arrangement problem. Yugosl J Oper Res 24(2):2334–6043. ISSN:0354-0243

89. Mladenović N, Urošević D, Pérez-Brito D, García-González CG (2010) Variable neighbourhood search for bandwidth reduction. Eur J Oper Res 200(1):14–27
90. Mutzel P (1995) A polyhedral approach to planar augmentation and related problems. In: Proceedings of the third annual European symposium on algorithms, ESA'95. Springer, London, pp 494–507
91. Palubeckis G, Rubliauskas D (2012) A branch-and-bound algorithm for the minimum cut linear arrangement problem. J comb optim 24(4):540–563
92. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199:285–304
93. Papadimitriou CH (1976) The NP-completeness of the bandwidth minimization problem. Computing 16:263–270
94. Pardo EG, Mladenović N, Pantrigo JJ, Duarte A (2012) A variable neighbourhood search approach to the cutwidth minimization problem. Electron Notes Discret Math 39(0):67–74
95. Pardo EG, Mladenović N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252
96. Pardo EG, Soto M, Thraves C (2015) Embedding signed graphs in the line. J Comb Optim 29(2):451–471
97. Peng SL, Ho CW, Hsu TS, Ko MT, Tang C (1998) A linear-time algorithm for constructing an optimal node-search strategy of a tree. In: Hsu W-L, Kao M-Y (eds) Computing and combinatorics. Lecture notes in computer science, vol 1449. Springer, Berlin/Heidelberg, pp 279–289
98. Petit J (2003) Combining spectral sequencing and parallel simulated annealing for the minla problem. Parallel Process Lett 13(1):77–91
99. Petit J (2003) Experiments on the minimum linear arrangement problem. ACM J Exp Algorithm 8:2–3
100. Piñana E, Plana I, Campos V, Martí R (2004) GRASP and path relinking for the matrix bandwidth minimization. Eur J Oper Res 153(1):200–210
101. Pop PC, Matei O (2011) An improved heuristic for the bandwidth minimization based on genetic programming. In: Hybrid artificial intelligent systems. Springer, Berlin, pp 67–74
102. Raspaud A, Schröder H, Sỳkora O, Torok L, Vrto I (2009) Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discret Math 309(11):3541–3552
103. Ravi R, Agrawal A, Klein P (1991) Ordering problems approximated: single-processor scheduling and interval graph completion. In: Proceedings of the ICALP. Springer, Berlin, pp 751–762
104. Resende MGC, Andrade DV (2009) Method and system for network migration scheduling. United States Patent Application Publication. US2009/0168665
105. Rodriguez-Tello E, Hao J-K, Torres-Jimenez J (2008) An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Comput Oper Res 35(10):3331–3346
106. Rodriguez-Tello E, Hao J-K, Torres-Jimenez J (2008) An improved simulated annealing algorithm for bandwidth minimization. Eur J Oper Res 185(3):1319–1335
107. Rolim J, Sýkora O, Vrt'o I (1995) Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes. In: Graph-theoretic concepts in computer science. Lecture notes in computer science, vol 1017. Springer, Berlin/Heidelberg, pp 252–264
108. Saad Y (2003) Iterative methods for sparse linear systems. SIAM, Philadelphia
109. Sánchez-Oro J, Duarte A (2012) An experimental comparison of variable neighborhood search variants for the minimization of the vertex-cut in layout problems. Electron Notes Discret Math 39:59–66
110. Sánchez-Oro J, Duarte A (2012) Grasp with path relinking for the sumcut problem. Int J Comb Optim Probl Inf 3:3–11
111. Sánchez-Oro J, Duarte A Grasp for the sumcut problem. In: XVII international congress on computer science research, Morelia (México), 26–28/10/2011
112. Sánchez-Oro J, Laguna M, Duarte A, Martí R (2015) Scatter search for the profile minimization problem. Networks 65(1):10–21

113. Sánchez-Oro J, Pantrigo JJ, Duarte A (2014) Combining intensification and diversification strategies in VNS. An application to the vertex separation problem. Comput Oper Res 52(Part B):209–219
114. Shahrokhi F, Sýkora O, Székely LA, Vrt'o I (2001) On bipartite drawings and the linear arrangement problem. SIAM J Comput 30(6):1773–1789
115. Shiloach Y (1979) A minimum linear arrangement algorithm for undirected trees. SIAM J Comput 8(1):15–32
116. Skodinis K (2000) Computing optimal linear layouts of trees in linear time. In: Proceedings of the 8th annual European symposium on algorithms, ESA'00. Springer, London, pp 403–414
117. Sýkora O, Torok L, Vrt'o I (2005) The cyclic antibandwidth problem. Electron Notes Discret Math 7th Int Colloq Graph Theory 22:223–227
118. Takagi K, Takagi N (1999) Minimum cut linear arrangement of p-q dags for VLSI layout of adder trees. IEICE Trans Fundam Electron Commun Comput Sci E82-A(5):767–774
119. Tewarson RP (1973) Sparse matrices, vol 69. Academic Press, New York
120. Thilikos DM, Serna MJ, Bodlaender HL (2005) Cutwidth II: algorithms for partial w-trees of bounded degree. J Algorithms 56(1):25–49
121. Woodcock JR (2006) A faster algorithm for torus embedding. PhD thesis, University of Victoria
122. Yannakakis M (1985) A polynomial algorithm for the min-cut linear arrangement of trees. J ACM (JACM) 32:950–988
123. Yuan J, Lin Y, Liu Y, Wang S (1998) NP-completeness of the profile problem and the fill-in problem on cobipartite graphs. J Math Study 31(3):239–243

Maritime Container Terminal Problems

35

Christopher Expósito-Izquierdo, Eduardo Lalla-Ruiz, Jesica de Armas, Belén Melián-Batista, and J. Marcos Moreno-Vega

Contents

Introduction 1052
Optimization Problems 1054
Berth Allocation 1055
Quay Crane Scheduling Problem 1061
Internal Vehicle Scheduling 1065
Container Storage 1070
Conclusion 1075
References 1076

Abstract

Maritime container terminals are essential infrastructures in global supply chains. Their high management complexity and heterogeneous processes make them an interesting field to apply heuristics. A brief overview of the main optimization problems found at maritime container terminals and a review of the way they are related to each other are firstly introduced. In order to solve these problems, several heuristics are presented and analyzed. The computational results reveal that they are suitable to be applied in practical scenarios due to the fact that they provide high-quality solutions in short computational times.

Keywords

Container · Heuristics · Logistics · Maritime container terminal

C. Expósito-Izquierdo (✉) · E. Lalla-Ruiz · J. de Armas · B. Melián-Batista · J. Marcos Moreno-Vega
Department of Computer Engineering and Systems, Universidad de La Laguna, San Cristóbal de La Laguna, Spain
e-mail: cexposit@ull.es; elalla@ull.es; jdearmas@ull.es; mbmelian@ull.es; jmmoreno@ull.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_57

Introduction

Containerization is, without doubts, the engine of economic globalization, which enables to move goods around the world within global intermodal supply chains by means of standardized containers. The containers are usually built of weathering steel and have locking systems and well-known international dimensions. The capacity unit of measure in transport is the twenty-foot equivalent unit, which refers to a container with a length of 20 feet. Due to their physical structure, containers can be stacked one on top of another, which allows to reduce the surface dedicated to their storage and be easily transported by a multitude of transport means.

The United Nations Conference on Trade And Development (UNCTAD) examines trends in seaborne trade and analyzes the comparative performance of different geographic regions and countries around the world. In this regard, the Review of Maritime Transport is an annual publication edited by the UNCTAD secretariat since 1968 with the goal of presenting a global maritime market analysis. One of the main assertions included in the last edition of the review in 2014 [25] remarks that the merchandise trade and seaborne shipments have been increased in tandem over the last years. Particularly, the world merchandise trade has grown about twice as fast as the world gross domestic product. Simultaneously, the seaborne trade has increased 4.3% in 2012, allowing to move about 9.2 billion tons of goods in ports worldwide. However, containerized trade is the fastest-growing market segment, giving rise to that 1.6 billion tons which are being nowadays transported by containers around the world.

In order to cope with the huge aforementioned seaborne trade, over the last decades, public institutions and freight shipping companies have funded the building of maritime container terminals in strategic regions as engines of economic development and investment elements. A maritime container terminal is a large infrastructure located in a seaport and aimed at joining together maritime and land transport means. The main transport means that can be usually found at a maritime container terminal are vessels, trucks, inland waterway systems, and trains. The main goal of a maritime container terminal is, hence, to perform the transshipment of freights between different means of transport efficiently. With this goal in mind, a maritime container terminal is responsible for unloading and loading the incoming vessels. The freights are temporarily stored by the terminal until their final means of transport pick them up and retrieve them from the terminal.

As depicted from left to right in Fig. 1, a maritime container terminal is typically split into the following main areas:

- *Seaside*. It is the part of the maritime container terminal aimed at serving the incoming vessels. It contains the most expensive machinery found at the maritime container terminal, the quay cranes, which perform the transshipment operations of containers between the terminal and the vessels. Roughly speaking, a quay crane grabs the containers from the vessels when they are berthed through its spreader during unloading operations. Then, the containers are lifted and moved ashore. Finally, the containers are placed on trucks in order to be moved toward

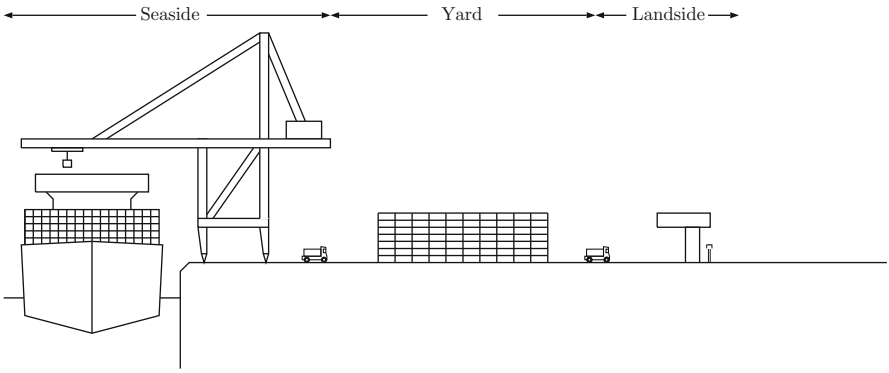


Fig. 1 Overview of a maritime container terminal

the yard or outside the terminal directly. This sequence of movements is reversed during loading operations.

- *Yard*. It is a temporal storage that usually takes up more than 50% of the terminal surface and in which those containers arrived to the terminal are stacked until their subsequent retrieval. The yard is split into several blocks, that is, sets of container bays arranged in parallel. A bay is a delimited two-dimensional stock disposed in the vertical direction, where containers can be placed one on top of another. The blocks are served by specialized machinery, such as, reach stackers, rubber-tyred gantry cranes, or rail-mounted gantry cranes.
- *Landside*. It is an interface to communicate the maritime container terminal with external trucks and trains. Several gates are available in order to supervise the movement of containers.

Some particular facilities that can be found in maritime container terminals are the following: container freight station, interchange area, and railhead. These are, respectively, dedicated to manage the consignments, allow road vehicles to deliver and retrieve cargoes from the terminal, and carry out inspection of the containers arriving or leaving by train.

Nowadays, the large volume of freights and the unceasing competition among maritime container terminals give rise to increasingly vessel operator's demand for reliable services. A suitable performance of a maritime container terminal is obtained by harmonizing the technical equipment and staff. However, maritime container terminals are complex to manage due to their large number of heterogeneous processes that take place in them. In this environment, heuristics stand as promising optimization techniques to be taken into consideration. The reason is found in that, as discussed along this chapter, most of the optimization problems that are addressed in a maritime container terminal belong to the $\mathcal{NP}$-hard class of problems. In this regard, even in small-size scenarios of some of these optimization

problems, efficient exact approaches do not exist. Nevertheless, heuristics report high-quality solutions through affordable computational times.

The remainder of this chapter introduces the main optimization problems found at a maritime container terminal, the way they are interconnected, and analyzes heuristics aimed at addressing them. Some conclusions and further lines of study are presented at the end of the chapter.

Optimization Problems

As mentioned before, maritime container terminals are huge infrastructures aimed at managing increasing transshipment flows of containers within transport multimodal networks. The large number of logistical problems taking place in them and their interrelations constitute a challenge for practitioners and researches. Henesey et al. [16] divide the problems taking place at container terminals in four large subsystems or categories, namely, *ship to shore*, *transfer*, *storage*, and *delivery*. In this regard, due to the similarity of the operations between transfer and delivery, they are here considered as a single category. Moreover, in the following, some of the most highlighted problems within these categories are briefly described:

- *Ship to shore*. Movements of loaded/unloaded containers from/to the sea to the container terminal:
  - *Ship routing* (Korsvik and Fagerholt [19]). These are a particular kind of routing problems where the vessels have to be routed along a set of ports in such a way that their cargoes are picked up and delivered from/to them.
  - *Stowage planning* (Monaco et al. [22]). In these problems, the goal is to determine the positions to be occupied by each container within a container vessel along its shipping route.
  - *Berth allocation* (Bierwirth and Meisel [3]). The problems under this category seek to determine the berthing position and berthing time of those container vessels arriving to the terminal.
  - *Quay crane assignment* (Meisel and Bierwirth [21]). Once a vessel is berthed, these problems seek to determine the fixed number of quay cranes to be assigned to the vessel according to its workload.
  - *Quay crane scheduling* (Meisel and Bierwirth [2]). Given a vessel and a subset of the available quay cranes, the aim of these problems is to determine the quay crane schedules for performing the vessel loading/unloading tasks.
- *Transfer and delivery*. Bidirectional movement of containers from the quay to storage area, from the storage area to the hinterland, or vice versa:
  - *Vehicle dispatching* (Angeloudis et al. [1]). Containers within transshipment flows are usually moved from the quay to the yard and vice versa by internal vehicles. In these problems, the management of those vehicles is addressed.
  - *Gate operations planning* (Chen et al. [8]). The management of vehicles with containers from the yard to the gate and vice versa is dealt by these problems.

- *Storage*. Movement and warehousing of containers until their retrieval for continuing their routes by vessels, trucks, or trains:
  - *Yard crane scheduling* (Gharehgozli et al. [15]). These problems are related to the schedule of the cranes within the yard at a container terminal in such a way that the loading, relocation, and unloading operations are performed.
  - *Container storage* (Caserta et al. [6]). These problems are aimed at determining the movements to be performed by the yard cranes in order to store/retrieve the containers on/from the yard.

In the remainder of this chapter, the focus is put on four particular optimization problems, namely, berth allocation problem, quay crane scheduling problem, vehicle dispatching, and container storage.

Berth Allocation

The berth allocation problem (BAP) is an $\mathcal{NP}$-hard optimization problem that seeks to determine the berthing position and berthing time of each incoming vessel that arrives to the port within a given planning horizon. The main constraint involved in the BAP concerns that the berthing positions of the incoming vessels cannot be overlapped in the same time step.

According to the arrival of the incoming vessels, two variants of the BAP can be identified: static BAP and dynamic BAP. Firstly, in the static BAP all the incoming vessels are already waiting for being berthed before the planning horizon starts. Secondly, the dynamic BAP considers that the vessels arrive to the port at any moment during the planning horizon. In both cases, each vessel could have a certain time window in which the vessel must be served by the maritime container terminal.

Moreover, the berth layout of the maritime container terminal gives rise to the following variants of the BAP:

- *Continuous BAP*. In this layout, the quay is not divided, thus the vessels can berth at any position along it.
- *Discrete BAP*. The quay is physically divided into segments called *berths* in such a way that each berth can serve a certain number of vessels at each time step.
- *Hybrid BAP*. There are also options to discretize the continuous quay to have intermediate options from classical berths up to almost continuous ones. This is known as hybrid layout. The continuous quay is divided into a set of berths, and a vessel can occupy more than one berth at a time or share its assigned berth with other container vessels.

In addition, according to the planning level to consider, the BAP can be classified as follows:

- *Operational*. It covers decisions ranging from one up to several days. Thus, the BAP within this level is aimed at optimizing the delays and waiting times of

container vessels. This variant of BAP has received larger attention than the other ones.
- *Tactical*. At this level, the decisions cover operations ranging from one week up to several months. Some of the objectives here are to optimize the transshipment flows among vessels, cycling visiting of the vessels, fulfilment of contracts among shipping companies and terminal managers, route design, etc.
- *Strategic*. The decisions covered at this level range from one up to some years. They seek to establish specific and dedicated berths, strategic cooperation agreements between terminal and shipping companies, etc.

In the related literature, the majority of works address the BAP at operational level. To a lesser extent, some works consider the tactical level and a few of them the strategic one. Due to this, in the following, how heuristics can be used to tackle the BAP at operational level is analyzed. In this regard, one of the most used variants of BAP is the dynamic BAP (DBAP) proposed by Imai et al. [17], who address the problem over a discrete layout. Then, Cordeau et al. [9] formulate it as multi-depot vehicle routing problem (MD-VRP). Additionally, they also include berth and vessel time windows. The authors propose a benchmark suite based on real data from the container terminal of Gioia Tauro (Italy).

The input data for the DBAP consists of a set of n incoming vessels to serve, denoted as V, a set of m berths, denoted as B, and a planning horizon, H. Each container vessel $v \in V$ has a certain time window, $[a_v, d_v]$, which must be served by the maritime container terminal, whereas its service priority is p_v. The handling time of vessel $v \in V$ in berth $b \in B$ is denoted as s_{vb}. This way, the turnaround time of vessel is the sum of its waiting time to be berthed and its handling time. Furthermore, each berth $b \in B$ has a known time window in which vessels can be served, $[s_b, e_b]$. The goal of this problem is to minimize the sum of (weighted) turnaround times of the vessels.

Figure 2 shows a solution example of the DBAP. In this figure, a schedule and an assignment plan are shown for $n = 6$ vessels in $m = 3$ berths. The rectangles represent the vessels, and inside each vessel its service priority is provided. The time windows of the vessels are represented by the lines at the bottom of the figure. In this case, for example, vessel 1 arrives at time step 4, and it should be served before time step 14. Moreover, the time window of each berth is limited by the non-hatched areas. Table 1 reports the different handling times of each vessel depending on the assigned berth. For example, the handling time of vessel 1 at berth 1 would be 7, whereas its handling time would be 8 at berth 2.

As can be seen in the previous example, vessels 5 and 6 would have to wait for berthing in their respective assigned berths. In this regard, since their service priorities are $p_5 = 2$ and $p_6 = 1$, their waiting times will have less impact on the objective function value than delaying other vessels, for example, vessels 3 and 4, for which their service priorities are $p_3 = 6$ and $p_4 = 4$, respectively. That is, if their berthing time is delayed, the waiting time of each vessel is multiplied by 6 and 4, respectively. In this example, the weighted turnaround time of the six vessels is

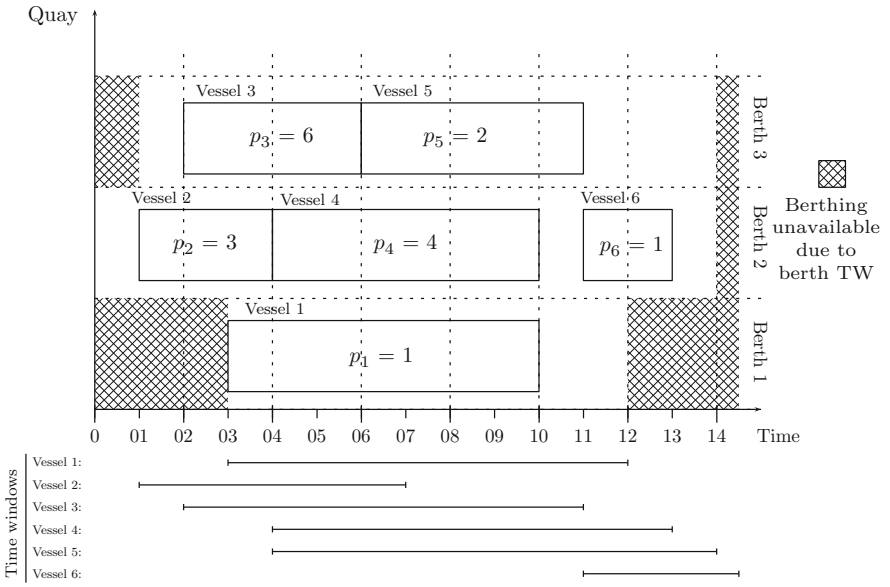


Fig. 2 Solution example of the Dynamic Berth Allocation Problem composed of 6 vessels and 3 berths

Table 1 Vessel handling times depending on the berth

Vessel	Berth 1	Berth 2	Berth 3
1	7	8	5
2	2	3	4
3	5	5	4
4	4	6	5
5	5	8	5
6	4	2	3

calculated as follows. Starting from vessel 1 and increasing a unit until vessel 6, the objective function value is $1 \cdot 7 + 3 \cdot 3 + 6 \cdot 4 + 6 \cdot 4 + 5 \cdot 2 + 2 \cdot 1 = 76$.

At container terminals, there are some queue rules used when scheduling incoming vessels to berthing positions. The vessels are sorted according to a specific rule. Some examples are described in the following:

- *Random (R)*. The incoming container vessels are randomly selected to be served by the terminal.
- *First come, first served (FCFS)*. This rule is based on serving vessels according to their arrival times. That is, the earliest vessel is the first to be served.
- *Shortest processing time (SPT)*. The vessels are sorted according to their required handling times. In this case, that vessel requiring the shortest handling time berths first.

- *Earliest due date (EDD)*. It serves the vessels on the basis of the end of their time windows. This way, that vessel whose time window ends earliest is served firstly.

The previous rules allow to obtain a permutation of the incoming container vessels to serve. There are four heuristics that can be obtained by using these rules in such a way that the vessels are iteratively assigned to the best possible berth. This berth is selected by minimizing the impact on the objective function value. The names of the heuristics are derived from the names of the rules. That is, *random greedy method* (R-G, Cordeau et al. [9]), *first come, first served greedy* (FCFS-G, Cordeau et al. [9]), *shortest processing time greedy* (SPT-G), and *earliest due date greedy* (EDD-G).

Algorithm 1 presents the general framework of the heuristics proposed in the literature (FCFS-G and R-G). It should be noted that the heuristics proposed in this chapter, namely, EDD-G and SPT-G, are also included within this general heuristic framework.

As indicated in Algorithm 1, the initial solution, S, is empty (line 1). Its associated objective function value, $f_{obj}(S)$, is set to 0 (line 2). Once this initialization is done, depending on the given rule, a permutation of vessels R is obtained (line 3). So, for example, in FCFS-G, the first element of the permutation, $R(1)$, is the first vessel that arrives at port. With the permutation already defined, the vessels are assigned one at a time (line 5) to the best possible berth following the order given by the sorted list (lines 4–8). The best possible berth is determined according to the impact on the objective function value of the solution (line 6). That is, each vessel is assigned to the berth that least increases the objective function value of the solution built until the moment.

Furthermore, a tabu search (see the ▸ Chap. 25, "Tabu Search") for solving this problem is proposed by Cordeau et al. [9]. Their proposed approach, $\mathtt{T^2S}$, [9] uses R-G and FCFS-G for restarting the search. In this regard, for improving the performance of their solution approach, Lalla-Ruiz et al. propose [20], on the one hand, the use of one additional neighborhood structure $\mathtt{T^2S^*}$ and, on the other hand, the use of a path-relinking procedure (see the ▸ Chap. 16, "GRASP"), $\mathtt{T^2S^*}$ + PR, for restating the search. In the path-reliking procedure, in order to create new starting

Algorithm 1: General heuristic framework for the Berth Allocation Problem

1: $S \leftarrow \emptyset$
2: $f_{obj}(S) = 0$
3: $R \leftarrow$ Create list of vessels according to selected rule
4: **for** $i \leftarrow 1$ to n **do**
5: $\quad v \leftarrow$ Select vessel $R(i)$
6: $\quad b \leftarrow$ Select berth that allows the minimum impact on $f_{obj}(S)$
7: $\quad$ Assign v to b in S
8: **end for**
9: Return S

Algorithm 2: Tabu search with path relinking for the dynamic berth allocation problem

1: s_{R-G} = R-G()
2: s_{FCFS-G} = FCFS-G()
3: $ES = \mathrm{T}^2\mathrm{S}^*_{min}(s_{R-G})$
4: $ES = ES \cup \mathrm{T}^2\mathrm{S}^*_{min}(s_{FCFS-G})$
5: **while** !stopping condition **do**
6: $s_{initial}$ = R-G()
7: $s_{elite} = Select(ES)$
8: $s = \mathrm{PR}(s_{initial}, s_{elite})$
9: $ES = ES \cup \mathrm{T}^2\mathrm{S}^*_{\mathrm{min}}(s)$
10: **end while**

points for the tabu search, an elite set of solutions, ES, consisting of a subset of the local optima encountered during the process, is used. Then, a path-relinking from one solution randomly picked from ES and one generated by R-G is performed.

Algorithm 2 shows the pseudo-code of $\mathrm{T}^2\mathrm{S}^* + \mathrm{PR}$. First of all, the elite set, ES, is initialized with the sets of local optima $\mathrm{T}^2\mathrm{S}^*_{min}(s_{R-G})$ (line 3) and $\mathrm{T}^2\mathrm{S}^*_{min}(s_{FCFS-G})$ (line 4). Then, a random solution, $s_{initial}$, is generated by running R-G (line 6) and a solution $s_{elite} \in ES$ is randomly selected (line 7). Once that, a path relinking connecting $s_{initial}$ with s_{elite} is performed (line 8). Finally, the midpoint solution of this path, s, is selected as starting point to run the tabu search $\mathrm{T}^2\mathrm{S}^*$ (line 9). The local optima reached along this search are also used to update the elite set of solutions (line 9). These steps (lines 5–9) are repeated until a stopping condition is met.

Comparative Analysis

This subsection is devoted to compare the heuristics to solve the DBAP. The problem instances used to evaluate the proposed heuristics are a representative set of the problem instances proposed by Cordeau et al. [9]. These instances were generated by taking into account a statistical analysis of the traffic and berth allocation data at the maritime container terminal of Gioia Tauro (Italy) [10]. Moreover, a representative set of instances from those proposed by Lalla-Ruiz et al. [20] are used. All the heuristics have been implemented in ANSI C and executed on a computer equipped with an Intel 3.16 GHz and 4 GB of RAM.

Table 2 illustrates the results obtained for a representative set of instances mentioned above. The first three columns correspond to the problem instance description, namely, the number of vessels, n, and, number of berths, m, and the instance identifier, *Id*. In columns *R-G*, *FCFS-G*, *SPT-G*, and *EDD-G*, the objective function values provided by the different heuristics for the DBAP are reported. That is, the objective function value, $Obj.$, and the computational time measured in seconds, *t(s.)*. In the last row, the average values are provided.

Table 2 Comparison of the heuristics for the Discrete Berth Allocation Problem. Best values in bold

Instance			R-G		FCFS-G		SPT-G		EDD-G	
n	*m*	*Id*	Obj.	t (s.)	Obj.	t (s.)	Obj.	t (s.)	Obj.	t (s.)
30	3	1	3237	<0.001	**2039**	<0.001	2667	<0.001	4702	<0.001
		2	3523	<0.001	**2829**	<0.001	4246	<0.001	3903	<0.001
40	7	1	2397	<0.001	**1612**	<0.001	3102	<0.001	4185	<0.001
		2	2987	<0.001	**1633**	<0.001	2755	<0.001	3961	<0.001
55	7	1	5874	<0.001	**3591**	<0.001	5695	<0.001	7576	<0.001
		2	5761	<0.001	**3565**	<0.001	5084	<0.001	7246	<0.001
60	13	1	2802	<0.001	**1437**	<0.001	2776	<0.001	2987	<0.001
		2	2861	<0.001	**1278**	<0.001	2761	<0.001	3361	<0.001
Average:			3680.25	<0.001	2248	<0.001	3635.75	<0.001	4740.13	<0.001

Furthermore, as indicated by Lalla-Ruiz et al. [20], the GSPP model implemented in CPLEX runs out of memory for some problem instance sets where other characteristics are taken into account. Therefore, in Table 3, the results provided by $\mathtt{T^2S}$, $\mathtt{T^2S^*}$, and $\mathtt{T^2S^*}$ + PR are reported for a set of instances where CPLEX is not able to provide a feasible solution for any instance within it. The table is divided, in column $Instance$, containing the number of vessels, n, number of berths, m, and the instance identifier, Id. In columns $\mathtt{T^2S}$, $\mathtt{T^2S^*}$, and $\mathtt{T^2S^*}$ + PR the objective function value, $Obj.$, and the computational time measured in seconds, *t(s.)*, are reported. Moreover, the relative error, $Gap_1(\%)$, with respect to $\mathtt{T^2S}$ is also reported. In the case of $\mathtt{T^2S^*}$ + PR, the relative error, $Gap_2(\%)$, with respect to $\mathtt{T^2S^*}$ is shown.

As can be seen in Table 2, the best heuristic for the DBAP irrespective of the size of the instance is the FCFS-G. On the other hand, R-G and SPT-G exhibit, on average, a similar performance in terms of objective function value, while the policy of serving the vessels according to the due date reports the worst values for these instances. It should be noted that the computational time required is less than 0.001 s, as expected for these heuristics. This advises their use as initialization methods. In this regard, FCFS-G and R-G are the heuristics most used in the related literature, which at the light of the results, are the best options in terms of quality of the solutions. It is important to highlight that R-G is not a deterministic heuristic, therefore, it provides different solutions that may be beneficial in multi-start methods.

The results shown in Table 3 indicate that among the three algorithms, the use of a restarting strategy based on path-relinking between a solution from an elite set and one generated by the R-G heuristic improves the quality of the solutions in terms of objective function value. In this regard, $\mathtt{T^2S^*}$, and $\mathtt{T^2S^*}$ + PR are able to provide a feasible solution in less than 2 s.

The results shown in this subsection highlight that the use of heuristics either alone or within metaheuristics for solving this problem is suitable and desirable.

Table 3 Comparison among $\mathrm{T}^2\mathrm{S}$, $\mathrm{T}^2\mathrm{S}^*$, and $\mathrm{T}^2\mathrm{S}^* + \mathrm{PR}$ for the instances proposed by Lalla-Ruiz et al. [20]. Best values in bold

Instance			$\mathrm{T}^2\mathrm{S}$		$\mathrm{T}^2\mathrm{S}^*$			$\mathrm{T}^2\mathrm{S}^* + \mathrm{PR}$			
n	m	Id	Obj.	t (s.)	Obj.	$\mathrm{Gap}_1(\%)$	t (s.)	Obj.	$\mathrm{Gap}_1(\%)$	$\mathrm{Gap}_2(\%)$	t (s.)
40	7	1	1489	8.16	1467	−1.48	1.20	**1460**	−1.95	−0.48	1.11
		2	1423	8.26	1381	−2.95	1.01	**1375**	−3.37	−0.43	1.32
		3	2149	8.13	**2119**	−1.40	0.84	**2119**	−1.40	0.00	1.17
		4	1618	7.99	1600	−1.11	1.18	**1597**	−1.30	−0.19	1.78
		5	1885	8.37	1849	−1.91	1.11	**1847**	−2.02	−0.11	1.45
		6	2104	8.11	**2080**	−1.14	0.86	**2080**	−1.14	0.00	1.37
		7	1863	7.91	1845	−0.97	1.25	**1841**	−1.18	−0.22	1.56
		8	2040	8.16	**2026**	−0.69	1.18	**2026**	−0.69	0.00	1.70
		9	1901	8.01	1888	−0.68	1.06	**1880**	−1.10	−0.42	1.48
		10	1922	7.89	1905	−0.88	0.71	**1892**	−1.56	−0.68	1.59

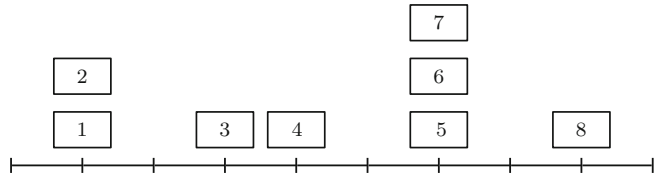
This is of particular significance when, as shown before, the exact approaches are not able to provide a feasible solution due to lack of memory or require high amounts of computational time.

Quay Crane Scheduling Problem

The quay crane scheduling problem (QCSP) can be defined as the problem of determining the finishing times of the tasks performed by each quay crane allocated to a container vessel in order to minimize its turnaround time (Meisel and Bierwirth [2]). Input data for the QCSP consist of a set of n tasks, denoted as Ω, and a set of m quay cranes, denoted as Q. Each $t \in \Omega$ is composed of a set of containers with similar characteristics (e.g., weight, destination port, dimensions, etc.) that are placed together in the same cross section, l_t, into the container vessel. The processing time of t is the time required by a quay crane to load or unload its containers and is denoted as p_t. Furthermore, each $q \in Q$ is available after its earliest ready time, r^q; is initially located on a cross section, l_0^q; and can travel between two adjacent positions with a travel time, $\hat{t}$.

The QCSP has a set of particular constraints. In first place, within each cross section of the container vessel, tasks are sorted according to known precedence relationships. Thus, tasks have to be performed orderly: unloading operations before loading operations, loading operations into a hold before loading operations on the deck, and so forth. Moreover, the quay cranes are rail mounted and, then, they cannot cross each other. Additionally, they have to keep a safety distance, δ, between them in order to prevent possible collisions (Kim and Park [18]).

Figure 3 illustrates an example of the QCSP and a feasible schedule for it where quay cranes are available from the starting of the service time (e.g.,



Task, t	1	2	3	4	5	6	7	8
Position, l_t	1	1	3	4	6	6	6	8
Processing Time, p_t	10	8	10	15	7	6	5	10

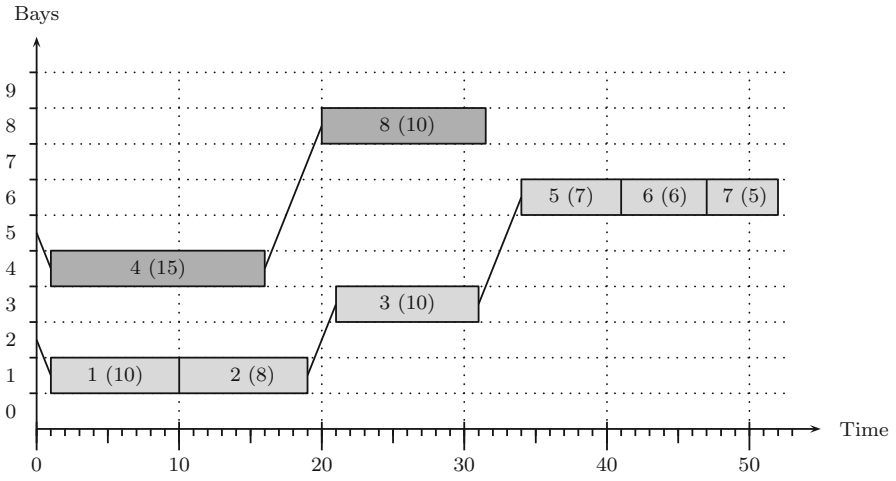


Fig. 3 Example of Quay Crane Scheduling Problem composed of $n = 8$ tasks and $m = 2$ quay cranes

$r^q = 0, \forall q \in Q)$, $\delta = 1$, and $\hat{t} = 1$. Within each cross section of the vessel, tasks are sorted. For instance, task 5 has to be performed before task 6 and task 6 before task 7. Additionally, in this example, tasks 1, 2, 3, 5, 6, and 7 are performed by quay crane 1 and tasks 4 and 8 by quay crane 2. The turnaround time of this container vessel is 52 time units.

An interesting heuristic approach to solve the QCSP is to explore the search space of unidirectional schedules (Meisel and Bierwirth [2]). A schedule is referred to as unidirectional if all quay cranes move with similar sense of movement along their service times. In the following, a hybrid estimation of distribution algorithm with local search (EDA/LS, Expósito-Izquierdo et al. [12]) is described to solve the QCSP from a unidirectional point of view. This approach uses a priori knowledge

Algorithm 3: Hybrid estimation of distribution algorithm with local search for the quay crane scheduling problem

1: $g = 0$
2: Initialize the probabilistic learning model, $\mathcal{P}(g)$
3: $\mathcal{S}(g) \leftarrow$ Generate the initial population from $\mathcal{P}(g)$
4: **repeat**
5: $\quad \mathcal{S}_e(g) \leftarrow$ Select the subset of elite solutions from $S(g)$
6: $\quad \mathcal{S}(g+1) \leftarrow$ Create population composed of solutions in $\mathcal{S}_e(g)$ and solutions from $\mathcal{P}(g)$
7: $\quad$ **if** λ generations without improvement **then**
8: $\quad\quad$ Apply Local Search to best solutions from $\mathcal{S}(g)$
9: $\quad$ **end if**
10: $\quad \mathcal{P}(g+1) \leftarrow$ Create next probabilistic learning model
11: $\quad g = g + 1$
12: **until** Stopping criterion is met

about the problem with the goal of achieving high-quality solutions. Additionally, a novel restart strategy (see the ▸ Chap. 8, "Restart Strategies") is introduced to prevent the premature convergence of the search and guide it to insufficiently explored regions. Finally, a proper finishing time of the search is determined by means of an adaptive stopping criterion.

Algorithm 3 depicts the pseudocode of the EDA/LS. The proposed scheme keeps, at each generation g, a population $\mathcal{S}(g)$ with N solutions (line 3). Additionally, a probabilistic learning model $\mathcal{P}(g)$ based on a probability matrix with m rows and n columns is considered (line 2). Each value $p_{qt}(g)$ defines the probability of assigning task $t \in \Omega$ to quay crane $q \in Q$ in the model sampling step, as follows:

$$\mathcal{P}(g) = \begin{pmatrix} p_{11}(g) & p_{12}(g) & \cdots & p_{1n}(g) \\ p_{21}(g) & p_{22}(g) & \cdots & p_{2n}(g) \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1}(g) & p_{m2}(g) & \cdots & p_{mn}(g) \end{pmatrix}$$

In most EDAs, the initial population is randomly created according to the uniform distribution over all feasible solutions. The QCSP structure allows to include a priori knowledge in the scheme of EDA/LS in a straightforward manner. Intuitively, the probability that a certain task is performed by a quay crane in a promising solution is highly influenced by the distance between them. The initialization step here is based on a Gaussian function in a discrete way to compute initial assignment probabilities. Its use is based on the fact that it lets to assign a higher weight to those quay cranes initially located close to the corresponding task, and it is decremented as the distance between them increases. At each new generation, a new population is obtained. The new population is composed of the subset of solutions with the lowest

objective function value, $\mathcal{S}_e(g)$, (defined by a percentage α, whose value is set by the user) from the previous population (elitism criterion) and new unidirectional ones sampled from the probabilistic learning model (line 6). In the sampling step, the tasks are assigned to a specific quay crane considering the cross sections in which they are located (from the leftmost up to the rightmost) and following the precedence relationships. At each step, one of the available quay cranes is selected according to the roulette wheel selection based on a pseudo-random generator. The subset of elite solutions is also used to update the probabilistic learning model in such a way that the probability of a task can be assigned to a quay crane and is defined by the number of times that assignment has been previously performed. This is formally expressed as follows:

$$p_{qt}(g) = \frac{c_{qt}(g)}{\sum_{k \in Q} c_{kt}(g)}, \forall q \in Q, t \in \Omega, \tag{1}$$

where $c(g)$ is a matrix composed of m rows and n columns such that $c_{qt}(g)$ is the number of times task $t \in \Omega$ has been performed by quay crane $q \in Q$ in a high-quality solution found during the search.

In order to overcome the lack of intensification of EDAs, this scheme incorporates a local search. This local search is based on a neighborhood structure composed by movements of reassignment and interchange of tasks between quay cranes. The next current schedule in the local search is selected according to the first improvement strategy. Additionally, the local search is applied to a subset of solutions with the lowest objective function value (line 8) defined by the percentage β (whose value is set by the user) after $\lambda > 0$ generations without improvement (lines 7–9). After the application of the local search, a subset of problem variables, $\mathcal{V}$, is selected. $\mathcal{V}$ is defined by the set of tasks performed by different quay cranes in the improved solutions. The effectiveness of the proposed local search is used as an indicator of the search process convergence, so that the algorithm execution is stopped when it is not possible to improve any solution from the selected subset of solutions from the population.

Computational Analysis

This section presents a comparison between the EDA/LS described above and the most competitive algorithm from the related literature: unidirectional scheduling, UDS (Bierwirth and Meisel [2]). This comparison is carried out on a benchmark suite proposed by Kim and Park [18] and subsequently extended by Bierwirth and Meisel [2] (90 instances). For each instance, the EDA/LS has been executed on a computer equipped with an Intel 3.16 GHz and 4 GB of RAM, whereas the computational results presented for UDS are those reported in the corresponding original paper. Computational times for those instances that cannot be solved within the maximum computational time of 1 h by the UDS are not reported.

Table 4 shows an extract of the computational results in the largest instances ($k83$–$k102$). The objective function values (f_{UDS} and f_{EDA}) and computational

Table 4 Comparison between EDA/LS and UDS. Largest instances

	UDS		EDA/LS		
Instance	f_{UDS}	$t_{UDS}(m.)$	f_{EDA}	$t_{EDA}(m.)$	Gap (%)
k83	948	6,37	948	0,63	0,00
k84	897	3,29	897	0,68	0,00
k85	972	5,82	972	0,68	0,00
k86	816	–	819	0,63	0,37
k87	**867**	–	867	0,80	0,00
k88	768	43,73	768	0,81	0,00
k89	843	10,96	843	0,66	0,00
k90	**1053**	24,95	1056	0.69	0,29
k91	**837**	10,74	840	0,70	0,36
k92	**897**	34,61	903	0,62	0,67
Avg.:	889,80	>60,00	891,30	0,69	0,17
k93	**816**	–	822	0,86	0,74
k94	**786**	–	795	0,92	1,15
k95	834	–	834	0,99	0,00
k96	819	–	**804**	0,89	−1,83
k97	720	–	**717**	**0,93**	−0,42
k98	**735**	23,97	741	0,89	0,82
k99	852	–	855	1,03	0,35
k100	900	–	**891**	**1,07**	−1,00
k101	**813**	–	816	0,87	0,37
k102	903	–	**900**	**0,86**	**−0,33**
Avg.:	817,80	>60,00	817,50	0,93	−0,04

times ($t_{UDS}(m.)$ and $t_{EDA}(m.)$) are shown for each approach. The results demonstrate the suitability of the EDA/LS when solving the QCSP. It has a great effectiveness and efficiency when solving the largest problem instances, since it presents short gaps and best solutions for four problem instances have been improved. In addition, the computational times are lesser than 1 min in all cases.

Internal Vehicle Scheduling

The internal vehicle scheduling problem (IVSP) is an $\mathcal{NP}$-hard optimization problem that consists of assigning jobs to the internal vehicles of the container terminal, in such a way that the time the incoming vessels spend at the port (i.e., their turnaround times) is minimized. In the related literature, this problem has been treated either as vehicle routing problem (VRP) or a scheduling problem (the reader is referred to ▸ Chap. 40, "Particle Swarm Optimization for the Vehicle Routing Problem: A Survey and a Comparative Analysis" for VRP and ▸ Chap. 43, "Supply Chain Management" for further details). In the IVSP, a set of n vehicles, denoted as

V, and a set of m jobs, denoted as $W = \{W_1, W_2, \ldots, W_m\}$, are known. In addition, a set of locations, denoted as L, in which the jobs can start and end is given. A job $j \in W$ corresponds to the transportation of a container from a pick-up location to a delivery location that must be performed from a certain time, t_j. There are two types of jobs:

- *Loading job*. It is the transportation of a container from the yard toward that vessel in which it is going to be loaded into. Thus, the pick-up location is a stack on the yard and the delivery location is at the corresponding quay crane.
- *Discharging job*. It is the transportation of a certain container from its source vessel toward the yard. Therefore, the pick-up location is at the corresponding quay crane and the delivery location is a known stack on the yard.

Thereby, each job has to be assigned to one of the vehicles. This way, each vehicle can perform a sequence of jobs, so that after completing a job, a vehicle can start another one. It should be noted that each job consists of (i) an empty drive of the corresponding vehicle from its last location $i \in L$ to the pick-up location, $k \in L$, whose time is t_{ik}, (ii) a hand-over time at the pick-up location, h_p, (iii) a drive to the delivery location, $l \in L$, and (iv) a hand-over time at the delivery location, h_d. Figure 4 illustrates these times involved in a job, where $departure$ corresponds to location i, $Source\ quay\ crane/stack$ corresponds to location k of the quay crane or a stack depending on the type of job, and finally, $destination\ quay\ crane/stack$ corresponds to location l of the quay crane or stack depending on the kind of job.

Once a vessel arrives at the terminal, its containers are first discharged from the vessel onto the internal vehicles by the available quay cranes. Next, the internal vehicles transport the containers to their pre-specified stacks on the yard. When a vehicle arrives to its destination, a container is unloaded from it by a yard crane. Typically, after most, or all, containers have been discharged from the vessel; the outgoing containers from the yard are transported to the quay by those internal vehicles and are loaded into the vessel by the quay cranes.

A fast transshipment of containers associated with the vessels is important to both the shipping companies, since they can reduce their operational costs, and the terminal, which can serve a large number of vessels per day. However, in order to achieve this reduction of the turnaround time, different objective functions can be

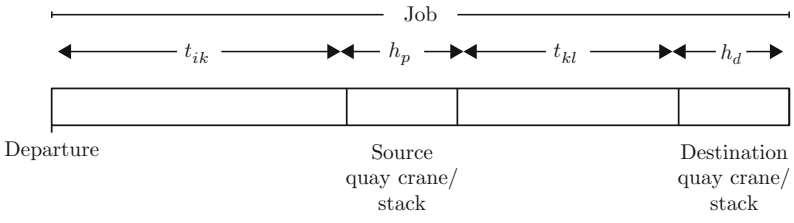


Fig. 4 Breakdown of the times involved in a job

defined such as minimizing the quay crane waiting times for vehicles, minimizing of the vehicles waiting times at quay cranes, minimizing of the empty travel times, and minimizing of the cost associated with the jobs.

There are different dispatching approaches to establish the relationship between vehicles and quay cranes. Traditionally, in the static dispatching approach, one vehicle serves only one quay crane. When it finishes an unloading job in the corresponding block on the yard, it must travel back to the bounding quay crane for another unloading job. An empty travel always happens in this dispatching approach. Furthermore, a dynamic dispatching approach means the vehicles can serve any quay crane. When a vehicle finished its unloading job in the blocks, it can choose either a loading job near it or an unloading job at the quay side. A dynamic dispatching model will help to reduce the empty travels.

Moreover, there is a variety of transporters used in container terminals, such as yard trucks (YTs), multi-trailers, straddle carriers, automated guided vehicles (AGVs), and automated lift vehicles (ALVs). Their heterogeneous technical characteristics allow to cover a wide range of practical scenarios. In this regard, selecting and dispatching technical machinery around a given maritime container terminal is a strategic decision to take into consideration according to the features of the environment, the volume of cargoes, etc.

Therefore, when addressing the IVSP, the terminal decision-maker must consider the following: firstly, how many and what type of vehicles are deployed at the terminal; secondly, what kind of relationship is established between vehicles and quay cranes; and finally, what will be the objective to achieve.

Due to the fact that IVSP is an operational problem (i.e., it has to be solved daily) and is usually integrated with other closely related problems in container terminals (e.g., quay crane scheduling, container storage, etc.), it is important to solve it quickly and provide results of the highest possible quality. For this reason, heuristics are appropriate methods to tackle this problem. However, most of the heuristics proposed in the literature are based on particular formulations of the problem with different objective function which hinders the comparison among them (e.g. Angeloudis and Bel [1] maximize benefits; Bish et al. [4] minimize travel distances and the transportation time; Briskorn, et al. [5] minimizes the weighted sum of earliness, tardiness and empty travel time; Shang [24] minimize a weighted sum of cost; etc.). In the following, we describe some basic heuristic strategies to solve the problem:

- *First come, first served (FCFS)*. Each idle vehicle selects the job with the earliest emergence time. In this case, the assignment concept is reversed. This is the conventional scheduling rule followed in container terminals to dispatch vehicles, and its results are taken as reference when a new method is proposed (Shang [24]):
- *First*. A due job will be assigned to the first available vehicle (Angeloudis and Bell [1]).
- *Fastest*. A due job will be assigned to the vehicle that needs less time to complete the job when it finishes its previous one.

- *Earliest*. A due job will be assigned to that vehicle that spends less time in be available and complete the job.
- *Closest*. A newly idle vehicle will be assigned to the job with the shortest setup operation. In this case, the assignment concept is reversed (Angeloudis and Bell [1]).
- *Greedy*. A vehicle will be assigned to the least costly job. The cost of performing job $j \in W$ once the job $i \in W$ has been already performed is denoted as C_{ij} and formally expressed as follows (Angeloudis and Bell [1]):

$$C_{ij} = \alpha L_{ij} + \beta T_{ij} + \gamma U_{ij} + \delta D_i \tag{2}$$

where α, β, γ, and δ are weights, L_{ij} is the distance between the location of the vehicle when finishing job i and the location of the vehicle when starting job j, T_{ij} is the travel time (expected) from the location of the vehicle when finishing of job i to the location of the vehicle when starting of job j, U_{ij} is the uncertainty index associated with transition from job i to job j, and, finally, D_i is equal to 0 if i is not a current vehicle task (assigned job, maintenance, or idle state) or equal to the expected remaining time until the completion of job i, otherwise.

Figure 5 illustrates an example of solution, for instance, with four vehicles and ten jobs using $First$ heuristic. The times from which the jobs can be performed (i.e., t_j, where $j \in W$) are marked by dashed line. Firstly, four jobs appear at time zero, and they are assigned to the available vehicles. Then, a new job W_5 arises at time instant five, and it is assigned to $Vehicle$ 4, which is the first available vehicle. Next job W_6 appears at time instant eight, and it is assigned to $Vehicle$ 3, which is available just at this time. The process continues following this strategy, so that it ends at time 29. Whenever a job is not assigned to a vehicle, this vehicle remains idle.

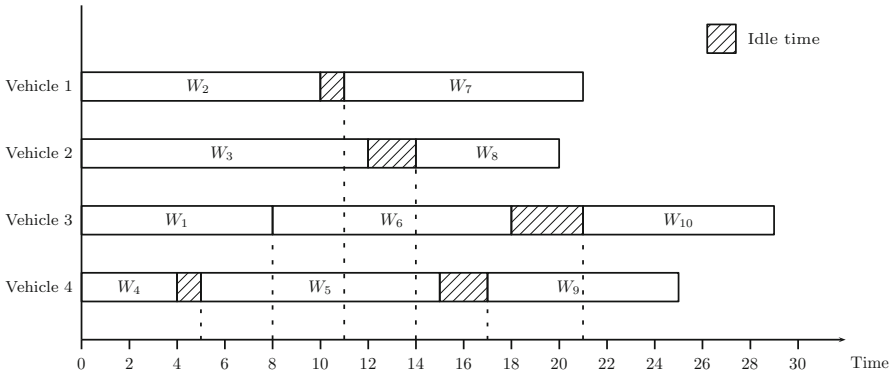


Fig. 5 Example of Vehicle Scheduling Problem using $First$ heuristic

Algorithm 4: General heuristic framework for the Vehicle Scheduling Problem

1: $S \leftarrow \emptyset$
2: $f_{obj}(S) = 0$
3: $\Omega \leftarrow$ List of jobs ordered by appearing time
4: $V \leftarrow$ Set of vehicles
5: **for** $i = 1 \rightarrow m$ **do**
6: $v \leftarrow$ Select vehicle from V servicing $\Omega(i)$ with the minimum impact on $f_{obj}(S)$
7: Assign v to $\Omega(i)$ in S
8: **end for**
9: Return S

Finally, Algorithm 4 shows a heuristic framework for this problem. A minimization function is supposed to be used. Firstly, an empty solution is established and the objective function is set to zero (lines 1–2). Then, for each job that appears during the planning horizon, a vehicle $v \in V$ is chosen to make it, trying to minimize the impact on the objective function value (line 6). These partial solutions are included in the final one (line 7). Finally, when all the jobs included into Ω have been already assigned to one of the vehicles, the process finishes and the solution is returned (line 9).

Computational Analysis

This subsection is devoted to compare some heuristics for the VSP. As mentioned before, the objective function of this problem may vary from one work to another, due to the different ways of reducing the turnaround. For this reason, some performance indicators have been here used to compare three of the heuristics explained above: $first$, $closest$, and $greedy$. In the following, several indicators are presented to evaluate the performance of the heuristics by means of simulation (Angeloudis and Bell [1]). That is:

1. Vehicle productivity (moves/h): It examines the amount of container moves that can be accomplished within an hour by a single vehicle in the fleet, so that, unlike the remaining indicators, the higher, the better.
2. Average job delay (s.): It measures the average period between the emergence of a new job and its time of execution.
3. Fleet utilization (%): It reflects the percentage of the fleet that is likely to be involved in a handling operation.
4. Average distance not in-service (m./job): It measures the average travelled distance between the location of the vehicle when finishing a job and the location of the vehicle when starting another.
5. CPU time per instance (ms.): It is the CPU time for solving an instance.

Table 5 Comparison of the heuristics for the Vehicle Scheduling (Angeloudis and Bell [1]). Best heuristic values in bold

	BKS	First		Closest		Greedy	
	Value	Value	Gap(%)	Value	Gap(%)	Value	Gap(%)
1	8.87	5.43	38.78	**5.66**	36.19	5.58	37.09
2	304.00	329.00	8.22	329.00	8.22	**325.00**	6.91
3	93.00	**99.00**	6.45	**99.00**	6.45	**99.00**	6.45
4	355.92	743.11	108.79	**713.71**	100.53	744.78	109.25
5	669.00	**0.00**	−100.00	1.00	−99.85	2.00	−99.70

Table 5 summarizes the computational results for $first$, $closest$, and $greedy$ heuristics. The first column shows the number associated with each performance indicator. The second column shows the best-known solutions (BKS). The remaining columns show the results obtained by each heuristic and the gap regarding BKS values. As can be seen, the heuristic that provides the best results for more indicators is $closest$, which presents the lowest gap for indicators 1, 3, and 4, as well as a very short CPU time. This highlights that serving a job according to the closeness in terms of setup operations benefits, on the overall, the quality of the solution in the majority of the indicators.

Container Storage

Container storage is a three-level (strategic, tactical, and operational) problem that arises at maritime container terminals. Defining the yard layout as well as selecting handling machinery to use (e.g., rubber-tyred gantry cranes, rail-mounted gantry cranes, and overhead bridge cranes) are decisions to take at strategic level. At a tactical level, decisions concerning the storage capacity of the yard and the handling machinery deployment are involved. Lastly, operational decisions are those related to the movement of containers on the yard in a short-term (i.e., from some hours to a few days).

Figure 6 illustrates a traditional layout of yard in a maritime container terminal in which the top and horizontal views are shown. It is composed of a set of similar three-dimensional storage areas, termed yard blocks. In this case, five yard blocks are depicted. The yard blocks can be placed in perpendicular (as Fig. 6) or parallel direction to the quay. Each yard block is composed of a set of bays (e.g., ten bays in Fig. 6). A bay is a two-dimensional storage composed of S homogeneous stacks made up of T tiers each one (e.g., bays in Fig. 6 are composed of $S = 8$ stacks and $T = 5$ tiers). The storage capacity of a terminal is expressed in terms of number of containers that can be stored on it and derived from the dimensions of its yard blocks. In this regard, the capacity of a given bay depends upon its stacks and tiers, whereas the capacity of a yard block is the sum of capacities of its individual bays.

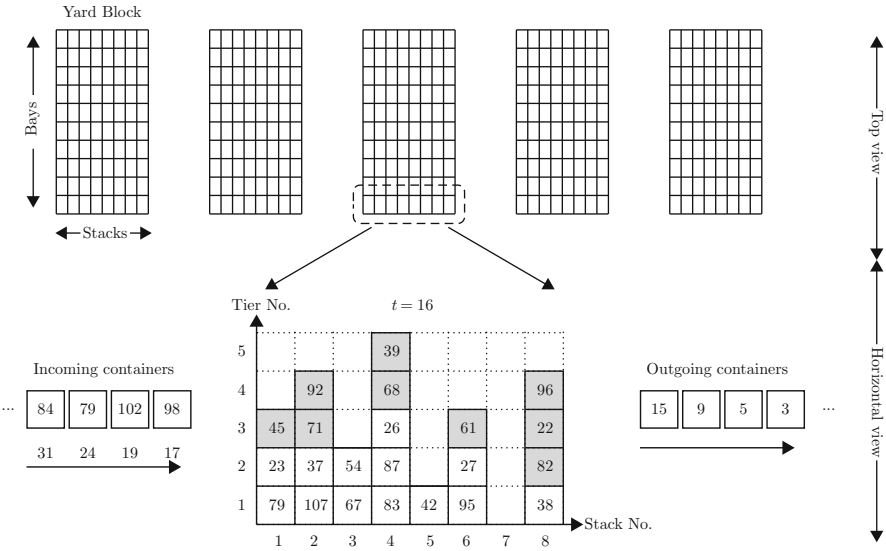


Fig. 6 Layout of a yard in a maritime container terminal

The yard blocks in a maritime container terminal are managed by yard cranes. Traditionally, one crane is deployed in each yard block. These handling machineries move along the yard blocks through a pair of rail tracks placed on both sides. In order to access a certain container, the handling machinery is positioned over the relevant bay in which the interested container is found. The trolley is moved toward its stack and the spreader is lowered to achieve the container. Finally, the container is hooked up and picked up to be removed from its current location.

During a certain planning horizon, H, containers arrive (incoming containers) and leave the yard (outgoing containers). The set of containers is denoted as C. In this regard, each container $c \in C$ has a known arrival time, denoted as $a(c)$, and a retrieval time, denoted as $r(c)$, where $r(c) > a(c)$. The objective of the yard cranes is to store and retrieve the containers according to their arrival and retrieval times. With this goal in mind, the feasible movements to be performed by a crane are defined on the basis of the intrinsic Last In First Out (LIFO) structure of the stacks. That is:

- *Storage movement*. The next incoming container is placed at the top of a stack with at least one empty slot.
- *Retrieval movement*. The next container to retrieve is taken out of its bay whenever it is currently placed at the top of a stack.
- *Relocation movement*. A container is moved from the top of a stack to the top of another one with at least one empty slot.

The incoming and outgoing containers in a bay give rise to the definition of the following closely related $\mathcal{NP}$-hard optimization problems:

- *Stacking Problem*. It is aimed at determining the shortest sequence of movements to be performed by the crane in order to store and retrieve the containers in/from the bay [13].
- *Container Relocation Problem*. It seeks to determine the shortest sequence of relocation movements to retrieve a subset of containers. It is assumed that all the containers are already stored in the bay, and no new incoming containers arrive [14].
- *Pre-Marshalling Problem*. Its objective is to find the shortest sequence of movements to arrange the containers within a given bay, such that any container is placed above other container with earlier retrieval time in the same stack. In this case, neither incoming containers nor outgoing containers are considered [11].

The main operational decision derived from the definitions of the previous optimization problems is how to evaluate the attractiveness of a certain stack when storing or relocating a container within the bay. Consider the example depicted in Fig. 6. The next incoming container arrives at time period 17 and must be retrieved from the bay at time period 98. However, its target stack must be selected adequately with the aim of minimizing the number of future relocation movements. Similarly, target stacks for those containers currently placed above the next to retrieve must be selected. For instance, see container with retrieval time 96 placed on top of container with retrieval time 22 in Fig. 6.

Algorithm 5 depicts a general heuristic framework to perform the storage and retrieval of containers. It is composed of two main steps: retrieving and storing containers. With this goal in mind, at each time period, $t \in [1, 2, \ldots, H]$, the sets of containers to retrieve and store from/in the bay are identified. On the one hand, each container c_{out} to retrieve at time period t is checked (lines 4–8). This way, the set of containers currently placed above it, denoted as $O(c_{out})$ (line 5), must be relocated in alternative stacks due to the fact that they are avoiding the retrieval of c_{out} (line 6). How the target positions of these containers are determined constitutes the major difference among the proposals developed by different authors so far. Some examples are described in section "Container Target Positions". Lastly, c_{out} can be directly retrieved from the bay because it would be placed at the top in its stack (line 7). On the other hand, once all the containers to retrieve from the bay at time t have been already retrieved, the storage of incoming containers is carried out. In this regard, suitable target stack and tier are firstly determined (lines 10–11) for each container to store and, then, moved to them (line 12). It should be noted that in Algorithm 5, retrieval operations (lines 4–8) are performed before storage operations (lines 9–13) in the same time period; however, according to the preferences of the decision-maker, the priority of storage and retrieval operations can be easily exchanged.

Algorithm 5: General heuristic framework to store and retrieve containers

```
for t = 1 → H do
    R_C(t) ← Set of containers to retrieve at time t
    S_C(t) ← Set of containers to store at time t
    for c_out ∈ R_C(t) do
        O(c_out) ← Set of containers currently placed above c_out
        Relocate containers from O(c_out) in other stacks
        Retrieve c_out from the top of its stack
    end for
    for c_in ∈ S_C(t) do
        s ← Select target stack for c_in
        t ← Select target tier for c_in
        Relocate c_in in slot (s, t)
    end for
end for
```

Container Target Positions

The main decision to take in the previous heuristic framework is where the containers must be stored or relocated in. This is the case of the incoming containers and those currently placed above the next to retrieve from the bay. In all cases, the problem is, given a certain container, to determine its most suitable target slot (i.e., stack and tier).

Determining the target slot of a container can be split into two parts: identifying the target stack and establishing a tier in it. In order to select a target stack, most of the authors have proposed attractiveness measures, which indicate the suitability of placing a given container in a stack. This way, the attractiveness of stack s to place container c is given by a scoring function with the shape, $f(c, s)$. In all cases, we define the set of feasible stacks of c, denoted as $S(c)$. It is composed of those stacks in which c is not currently placed and with, at least, one empty slot. Thus, it is assumed that $s \in S(c)$. Some examples of scoring functions are shown in the following:

1. *Random*. The target stack is selected at random among those feasible stacks for c. That is:

$$f(c, s) = \frac{1}{|S(c)|} \tag{3}$$

2. *Conflict Minimization* (Rei and Pedroso [23]). It seeks to place c in a stack in which it is not going to require future relocations. The set of stacks that allows this fact is denoted as $S(c)'$ and, therefore, the scoring function is defined as follows:

$$f(c,s) = \begin{cases} \frac{1}{|S(c)'|} & \text{if} S(c)' \neq \emptyset \\ \frac{1}{|S(c)|} & \text{otherwise} \end{cases} \tag{4}$$

3. *Flexibility Optimization* (Rei and Pedroso [23]). It aims to place containers with close retrieval times together, whereas in those scenarios in which future relocation movements are unavoidable, these are delayed as much as possible. This is expressed as follows:

$$f(c,s) = \begin{cases} K & \text{if } s \text{ is empty} \\ M_s & \text{if } M_s \geq r(c) \\ 2 \cdot K - M_s & \text{if } M_s < r(c) \end{cases}, \tag{5}$$

   where K is a constant representing a value larger than the latest retrieval time of a container in the bay, and M_s represents the earliest retrieval time of a container in the stack s. These are formally defined as follows: $K = 1 + \max\{r(c')|c' \in C\}$ and $M_s = \min\{r(c')|c' \in C \wedge c' \text{ in } s\}$.
4. *Parameterized Flexibility Optimization* (Rei and Pedroso [23]). It is an extension of the FO in which the best stack is selected with probability $1 - \rho$ (where ρ is a parameter whose value is set by the user), whereas one of the remaining stacks is selected with probability ρ.

In the previous functions, the best stack is selected to be target of c. Moreover, other authors have proposed explicit expressions to select the target stack of container c, denoted as s^*. This is the case, for instance, of Caserta et al. [7], who propose to select the target stack, as follows:

$$s^* = \begin{cases} \arg\min_{s \in S(c)}\{\min(s) | \min(s) > r(c)\} & \text{if } \exists s : \min(s) > r(c) \\ \arg\max_{s \in S(c)}\{\min(s)\} & \text{otherwise} \end{cases}, \tag{6}$$

where

$$\min(s) = \begin{cases} 1 + \max\{r(c')|c' \in C\} & \text{if } s \text{ is empty} \\ \min\{r(c')|c' \in C \wedge c' \text{ in } s\} & \text{otherwise} \end{cases}.$$

In this case, Eq. (6) seeks to firstly place container c in one of the stacks in which it is not going to require additional relocations. Particularly, that stack storing a container with the earliest retrieval time is chosen. However, when this is not possible, c is placed in that stack in which the next retrieval movement must be performed as later as possible.

Computational Analysis

In the following, the performances of the previous heuristics are assessed. With this goal in mind, a benchmark suite has been considered. It is composed of a wide range of problem instances with a number of containers ranging from 50 up to 300. In each

Table 6 Performance of the heuristics on a wide range of practical-size scenarios

Instance	Random	CM	FO	PFO ($\rho = 0.1$)	CSV
50	73.6	32.9	3.3	5.3	3.5
100	308.4	263.3	29.4	40.5	31.1
150	714.3	680.9	114.0	134.6	114.2
200	1270.2	1329.3	246.3	289.9	247.0
250	1949.4	2222.3	438.6	496.3	443.5
300	2879.4	3501.8	624.1	720.3	635.1

case, the containers must be stored in and retrieved from a bay composed of $S = 10$ stacks and without maximum stacking limit.

Table 6 reports the computational results obtained by the heuristics under analysis over the problem instances from the aforementioned benchmark suite. In this regard, all the heuristics have been implemented in Java and executed on a computer equipped with an Intel 3.16 GHz and 4 GB of RAM. In each case, ten problem instances with a certain number of containers have been used. The columns *Containers* shows the number of containers. Furthermore, columns *Random*, *CM*, *FO*, *PFO* ($\rho = 0.001$), and *CSV* present the average number of relocation movements of the solutions reported by the heuristics. The computational times required by the heuristics are not included into the comparison due to the fact that all of them are, even in large-scale scenarios, too negligible to draw any relevant conclusions.

As can be checked in the computational results, FO reports solutions with the lowest number of relocation movements of those heuristics in the comparison. The reason is found in that this heuristic allows to minimize the differences of retrieval times of consecutive containers in the same stack of the bay. It should be noted that placing a container at the top of a stack by minimizing the differences of retrieval times reduces the probability of requiring additional relocation movements in the future. Analogously, CSV exploits this idea and, consequently, it also reports high-quality solutions.

Conclusion

Maritime container terminals are large infrastructures in which heterogeneous logistic processes are brought together. This fact, together with the increasing volume of containers moved around the world, gives rise to their management and is extremely complex.

Up to now, exact approaches have not demonstrated to be efficient due to the $\mathcal{NP}$-hard nature of most of the optimization problems found at maritime container terminals. In this regard, heuristics have stood as the most competitive alternatives because they report (near) optimal solutions within short computational times. Along this chapter, several heuristics aimed at solving several relevant optimization

problems are presented and discussed. The analysis of the performance of the heuristics here assessed indicates that some of them are appropriate when addressing realistic size scenarios.

Acknowledgments This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (projects TIN2012-32608 and TIN2015-70226-R) and with FEDER funds.

References

1. Angeloudis P, Bell MGH (2010) An uncertainty-aware AGV assignment algorithm for automated container terminals. Transp Res E Logist Transp Rev 46(3):354–366
2. Bierwirth C, Meisel F (2009) A fast heuristic for quay crane scheduling with interference constraints. J Sched 12(4):345–360
3. Bierwirth C, Meisel F (2010) A survey of berth allocation and quay crane scheduling problems in container terminals. Eur J Oper Res 202(3):615–627
4. Bish EK, Chen FY, Leong YT, Nelson BL, Ng JWC, Simchi-Levi D (2005) Dispatching vehicles in a mega container terminal. OR Spectr 27(4):491–506
5. Briskorn D, Drexl A, Hartmann S (2006) Inventory-based dispatching of automated guided vehicles on container terminals. OR Spectr 28(4):611–630
6. Caserta M, Schwarze S, VoßS (2011) Container rehandling at maritime container terminals. In: Böse JW (ed) Handbook of terminal planning. Volume 49 of operations research/computer science interfaces series. Springer, New York, pp 247–269
7. Caserta M, Schwarze S, VoßS (2012) A mathematical formulation and complexity considerations for the blocks relocation problem. Eur J Oper Res 219(1):96–104
8. Chen G, Govindan K, Yang Z (2013) Managing truck arrivals with time windows to alleviate gate congestion at container terminals. Int J Prod Econ 141(1):179–188
9. Cordeau JF, Laporte G, Legato P, Moccia L (2005) Models and tabu search heuristics for the berth-allocation problem. Transp Sci 39(4):526–538
10. Cordeau JF, Gaudioso M, Laporte G, Moccia L (2007) The service allocation problem at the Gioia Tauro maritime terminal. Eur J Oper Res 176(2):1167–1184
11. Expósito-Izquierdo C, Melián-Batista B, Moreno-Vega JM (2012) Pre-marshalling problem: heuristic solution method and instances generator. Expert Syst Appl 39(9):8337–8349
12. Expósito-Izquierdo C, González-Velarde JL, Melián-Batista B, Moreno-Vega JM (2013) Hybrid estimation of distribution algorithm for the quay crane scheduling problem. Appl Soft Comput 13(10):4063–4076
13. Expósito-Izquierdo C, Lalla-Ruiz E, de Armas J, Melián-Batista B, Moreno-Vega JM (2015) A heuristic algorithm based on an improvement strategy to exploit idle time periods for the stacking problem. Comput Ind Eng 87:410–424
14. Expósito-Izquierdo C, Melián-Batista B, Moreno-Vega JM (2015) An exact approach for the blocks relocation problem. Expert Syst Appl 42(17–18):6408–6422
15. Gharehgozli AH, Yu Y, de Koster R, Udding JT (2014) An exact method for scheduling a yard crane. Eur J Oper Res 235(2):431–447. Maritime Logistics.
16. Henesey L (2006) Overview of transshipment operations and simulation. In: MedTrade conference, pp 6–7
17. Imai A, Nishimura E, Papadimitriou S (2001) The dynamic berth allocation problem for a container port. Transp Res Part B Methodol 35(4):401–407
18. Kim KH, Park YM (2004) A crane scheduling method for port container terminals. Eur J Oper Res 156(3):752–768
19. Korsvik JE, Fagerholt K (2010) A tabu search heuristic for ship routing and scheduling with flexible cargo quantities. J Heuristics 16(2):117–137

20. Lalla-Ruiz E, Melián-Batista B, Moreno-Vega JM (2012) Artificial intelligence hybrid heuristic based on tabu search for the dynamic berth allocation problem. Eng Appl Artif Intell 25(6):1132–1141
21. Meisel F, Bierwirth C (2013) A framework for integrated berth allocation and crane operations planning in seaport container terminals. Transp Sci 47(2):131–147
22. Monaco MF, Sammarra M, Sorrentino G (2014) The terminal-oriented ship stowage planning problem. Eur J Oper Res 239(1):256–265
23. Rei RJ, Pedroso JP (2012) Heuristic search for the stacking problem. Int Trans Oper Res 19(3):379–395
24. Shang J (2010) A heuristic algorithm for the integrated yard truck scheduling in container terminal with twin 40-foot quay crane. In: 2010 international conference on computer, mechatronics, control and electronic engineering, Piscataway, vol 2, pp 386–389
25. United Nations Conference on Trade and Development. Review of maritime transport (2014)

Metaheuristics for Medical Image Registration

36

Andrea Valsecchi, Enrique Bermejo, Sergio Damas, and Oscar Cordón

Contents

Introduction 1080
Image Registration 1082
Problem Statement 1082
Classic IR Algorithms 1085
Metaheuristics-Based Image Registration 1086
Suitability of MHs in Image Registration 1086
Early Evolutionary Image Registration Methods 1087
Outstanding IR Methods Based on MHs 1089
Feature-Based Techniques 1089
Intensity-Based Techniques 1091
Experimental Study 1092
Feature-Based Comparison: Registration of Brain Magnetic Resonance Images 1092
Second Experiment: Atlas-Based Segmentation of Brain Deep Nuclei 1095
Conclusion 1096
Cross-References 1098
References 1098

A. Valsecchi (✉) · E. Bermejo · O. Cordón
Department of Computer Science and Artificial Intelligence, Unviersity of Granada, Granada, Spain
e-mail: andreavalsecchi@ugr.es; andrea.valsecchi@softcomputing.es; enric2186@ugr.es; enric2186@gmail.com; ocordon@decsai.ugr.es

S. Damas
Department of Software Engineering, University of Granada, Granada, Spain
e-mail: sdamas@ugr.es; sergio.damas@softcomputing.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_56

Abstract

In the last few decades, image registration (IR) has been a very active research area in computer vision. Applications of IR cover a broad range of real-world problems, including remote sensing, medical imaging, artificial vision, and computer-aided design. In particular, medical IR is a mature research field with theoretical support and two decades of practical experience. Formulated as either a continuous or combinatorial optimization problem, medical IR has been traditionally tackled by iterative numerical optimization methods, which are likely to get stuck in local optima and deliver suboptimal solutions. Recently, a large number of medical IR methods based on different metaheuristics, mostly belonging to evolutionary computation, have been proposed. In this chapter, we review the most recognized of these algorithms and develop an experimental comparison over real-world IR scenarios.

Keywords

Medical imaging · Image registration · Image segmentation

Introduction

In its most general formulation, image registration [60] is the task of aligning two or more images in order to establish a spatial correspondence of their common content. These images usually have the same or a similar subject but have been acquired under different conditions, such as time and viewpoint, or by multiple sensors. In medical image analysis, IR is a key technology that allows to "fuse" visual information from different sources [56]. Applications include combining images of the same subject from different modalities, detecting changes before/after treatment, aligning temporal sequences of images to compensate for motion between scans, image guidance during interventions, and aligning images from multiple subjects in cohort studies. The remarkable developments in medical imaging technology over the last few decades determine a constant demand for better image processing and analysis techniques. Dealing with novel, more diverse, and increasingly accurate sources of imaging data is the main challenge in IR, and it explains why it is still a very active research field.

The alignment between two images is specified as a spatial transformation, mapping the content of one image to the corresponding area of the other through a composition of translation, rotation, shear, and other kind of operations. A popular strategy among IR methods is to perform the alignment based on only salient and distinctive parts of the image, such as lines, corners, and contours, called *features*, ignoring the rest of the image content. This approach, called *feature based* [56], has the advantage of greatly reducing the complexity of the problem but relies

on the ability to precisely detect the features, either manually or automatically. Any error during the feature extraction stage will propagate into the registration and can hardly be recovered at a later stage. Moreover, this approach is limited to the cases in which features provide enough information to characterize the image content. To avoid these drawbacks, it is possible to use the image intensities directly without any feature extraction, an approach called *intensity based* (or voxel-based) [60]. Though more expensive in computational terms, intensity-based methods can achieve a superior level of accuracy and robustness.

Regardless of the division, the core of every IR technique is an *optimization process* that explores the space of geometrical transformations. Two strategies are available. In *parameters*-based approaches, the search is directly performed in the space of the registration transformation parameters, turning the registration into a continuous optimization problem. In *matching*-based approaches, instead, features or regions of the image are matched through a search in the space of possible correspondences. In this case, IR is formulated as a combinatorial optimization problem. Once a suitable matching has been found, the transformation parameters are derived accordingly through numerical methods. In both cases the search is guided by a *similarity metric*, a function that measures the degree of resemblance between the input images after the alignment. This can be done either by comparing the whole images or just their corresponding features. Traditional parameters-based methods use classic numerical optimization algorithms [32], while matching-based methods use matching algorithms like iterative closest point (ICP) [5].

Many characteristics of the IR problem, such as noise, discretization, and large differences in the order of magnitude of the transformation parameters, still pose a challenge to traditional optimization methods. A number of alternative approaches based on *metaheuristics* (MHs) [19] are often used to deal with complex real-world problems in computer vision and image processing. In the literature, despite their common structure, feature- and intensity-based methods are usually regarded as different categories of algorithms, therefore most publications deal with either one group or the other. Besides, feature-based approaches based on MHs are much more common than intensity-based ones, as the larger computational cost makes them less suitable for MHs. Nevertheless, both groups have been recently reviewed [13, 52], although separately.

This chapter is structured as follows. Section "Image Registration" provides a general introduction to medical IR and the traditional optimization algorithms used to solve it. Section "Metaheuristics-Based Image Registration" deals with the application of MHs to IR, while section "Outstanding IR Methods Based on MHs" reviews the most important medical IR methods of this kind. Section "Experimental Study" develops two experimental studies, comparing traditional and MH-based methods of the two different families, feature- and intensity-based, over real-world medical applications. Finally, conclusions are provided in section "Conclusion".

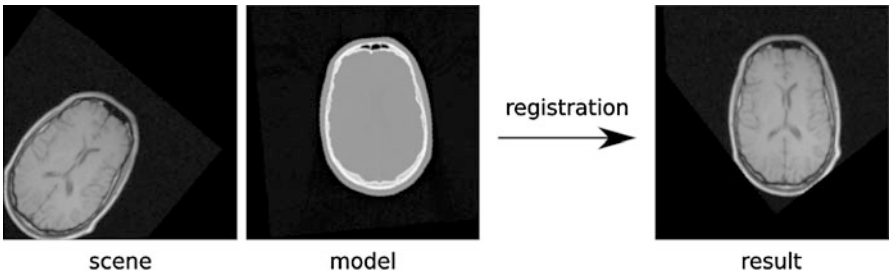


Fig. 1 An example of image registration. A MRI brain scan (*left image*) is aligned to the pose of the CT scan (*middle image*), resulting in the right image

Image Registration

Problem Statement

A typical IR problem involves two images, conventionally called *model* (I_M) and *scene* (I_S), with different roles in the registration process. The model is the reference (or target) image, while the scene is the image that is transformed to reach the geometry of the other. Figure 1 illustrates such a process. The registration aims to find a geometric transformation T that aligns the scene to the model. In other words, T is the transformation that makes the model I_M and the transformed scene $T(I_S)$ as similar as possible according to the similarity metric of choice. In other terms, IR is a maximization problem over transformations, formally stated as

$$\underset{T \in \text{Transformations}}{\arg\max} \quad \text{Similarity}\,(I_M, T(I_S))$$

A number of components characterize an IR method, but the main ones are just three: the kind of transformation used to relate the images, the similarity metric that measures the quality of the alignment, and the optimization procedure that performs the search for a suitable transformation. The optimization is usually an iterative process. The optimizer computes a candidate transformation, which is then applied to the scene image. Next, the similarity metric compares the model with the transformed scene image and returns a quality value that is sent back to the optimizer. Figure 2 shows a flow chart of the process. The loop ends when a suitable transformation has been found or the algorithm has performed a certain number of iterations.

Transformation Model

The transformation model determines which kind of geometrical transformation can be applied to the scene image to reach the model. This also controls which geometrical properties (e.g., size, shape, position, orientation, etc.) are preserved through the process. Figure 3 shows the effect of three common transformation models.

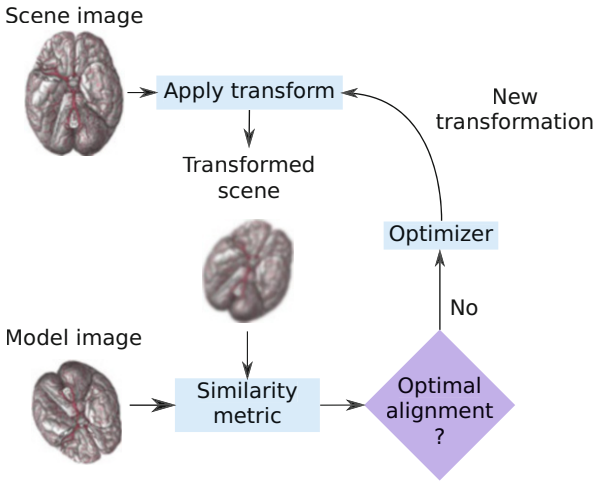


Fig. 2 The interactions among the components of a registration technique

A rigid transformation involves arbitrary translation and rotation operations, thus it has six degrees of freedom for 3D images. A similarity transform also admits a scaling operation, specified by a seventh parameter. In more complex scenarios, it may also be necessary to correct for shears, for example, those caused by the gantry tilt of CT scanners, or for scaling on a per axis basis. Thus, an affine transformation (nine parameters) is required.

Rigid, similarity, and affine transformations are frequently used for the registration of anatomical structures like the brain or bones; they are not applicable in the case where a significant local deformation is expected, e.g., in soft tissues like the breast. In these cases, deformable or non rigid transforms are required. In a typical approach, an object is deformed by manipulating an underlying mesh of control points. The resulting deformation controls the shape of the 3D object and produces a smooth and continuous transformation between control points.

The choice of the transformation model depends entirely on the needs of the application at hand. A too flexible transformation model is not just more complex and computationally expensive to apply but can also lead to results that are undesired or implausible with respect to the phenomenon under study. Bones being bent and tissues growing at an unrealistic rate are examples of such unacceptable outcomes in the medical context.

Similarity Metric

The performance of any IR method depends on an accurate estimation of the degree of alignment between the images, therefore the similarity metric is considered a crucial component [48]. Technically, a similarity metric is a real-valued function $F(I_A, I_B)$ that measures the resemblance of two images I_A, I_b. The quality of a transformation T is assessed by computing the similarity metric over the model

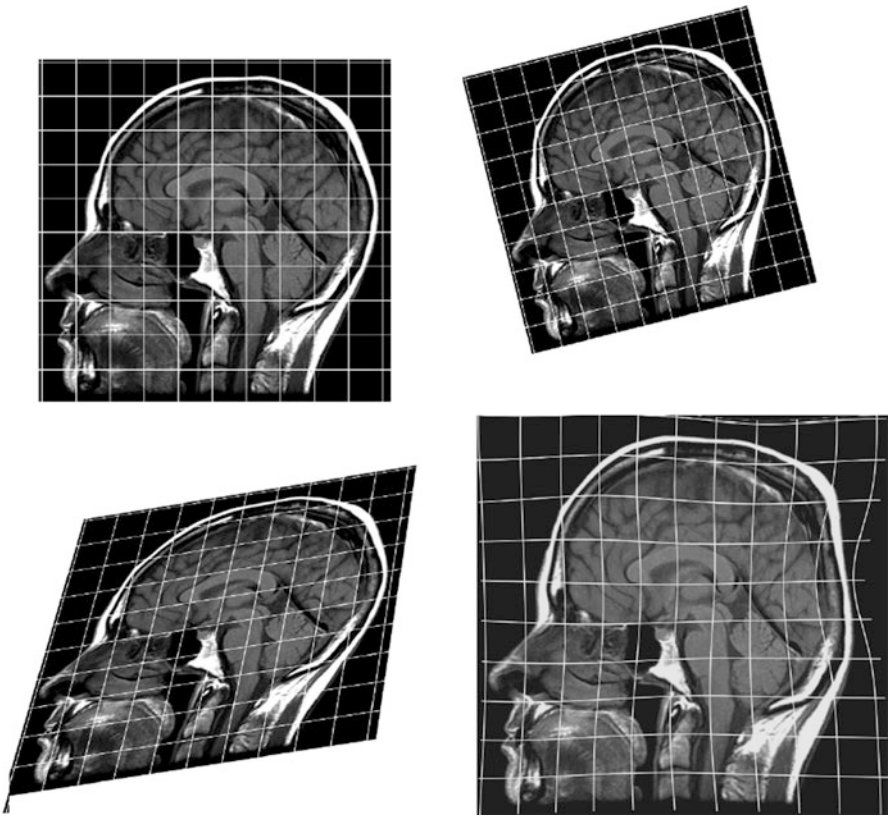


Fig. 3 Images obtained from the same scene (*top left*) by applying different transformations: similarity (*top right*), affine (*bottom left*) and B-spline (*bottom right*)

I_M and the transformed scene $T(I_S)$, i.e. $F(I_M, T(I_S))$. The actual evaluation mechanism depends on the nature of the registration approach. In feature-based methods, the similarity metric usually measures the distance between corresponding features [1]. For instance, if the features are points, the alignment can be evaluated using the mean square error (MSE) between the position of a point in the model and that of the corresponding (or closest) point in the transformed scene, i.e.,

$$\text{MSE} = \frac{1}{r}\sum_{i=1}^{r} \|x_i - c(T(x_i))\|^2$$

where r is the number of points and c is the correspondence criterion.

In intensity-based approaches, similarity metrics are usually based on the resemblance of the intensity values in the two images. The subject of the images along with their modality determine what kind of relationship is established between their intensity distributions. For instance, if one assumes this relationship to be linear, the similarity between the images can be assessed by computing the linear correlation

coefficient. When two images have been acquired using different sensors, a scenario called *multi modal* registration, the relationship between their intensity values can be strongly nonlinear. Metrics based on information theory, such as mutual information (MI) [39], are better suited for this scenario. MI is defined as

$$\mathrm{MI}(I_A, I_B) = \sum_{a \in I_A} \sum_{b \in I_B} p_{AB}(a,b) \log \frac{p_{AB}(a,b)}{p_A(a)\ p_B(b)}$$

where p_{AB} and p_A, p_B are, respectively, the joint and marginal probability distributions of the intensity values of the images.

Optimization Procedure

The third main component of an IR method is the *optimizer*. It is responsible for finding the best transformation, in terms of similarity metric, among the candidate transformations in the transformation model. Each optimizer has a different search strategy, which also depends on the nature of the algorithm. One approach is to perform the search directly in the space of the transformation parameters. This turns the registration into a continuous optimization problem; therefore classic numerical optimization algorithms can be used. Gradient descent, conjugated gradient descent, Newton's and quasi-Newton methods, Powell's method, and discrete optimization [25, 32] are among the most common choices along with approaches based on evolutionary computation and other metaheuristics [8, 43, 57]. IR algorithms that follow this approach are called *parameters based*. An alternative approach consists in searching for a matching between features, in feature-based methods, and areas of the image, in intensity-based ones. From the match, one can derive the parameters of the corresponding transformation using least-squares estimation or other more robust model fitting techniques [58]. This class of algorithms is called *matching based*. The ICP algorithm is the most representative example of this second approach [29, 44, 59].

Classic IR Algorithms

Iterative Closest Point

ICP [5] is a well-known feature-based algorithm in computer-aided design, originally proposed to recover the 3D transformation of pairs of range images. We are given a point set P of N_p points $\mathbf{p_i}$ from the data shape (the scene) and the model image X of N_x. In the original contribution, the approach dealt with 3D rigid transformations stored in a solution vector $\mathbf{q} = [q_1, q_2, q_3, q_4, t_1, t_2, t_3]$. The first four parameters correspond to a quaternion, determining the 3D rotation component, while the remaining three parameters store the translation vector. The procedure is initialized by setting $P_0 = P$, the initial registration transformation to $\mathbf{q_0} = [1, 0, 0, 0, 0, 0, 0]$ and the iteration counter k to zero. The next four steps are applied until reaching convergence within a tolerance threshold $\tau > 0$:

1. Compute the matching between the scene and the model points using the closest assignment rule. $Y_k = C(P_k, X)$
2. Estimate the registration by least squares. $f_k = \rho(P_o, Y_k)$
3. Apply the registration transformation to the scene image: $P_{k+1} = f_k(P_0)$
4. Terminate the procedure if the change in MSE falls below τ. Otherwise increase k and go to step 1.

Note that ICP is not directly guided by a similarity metric but rather by the computed matching, as other matching-based IR methods. In this strategy, the function MSE only plays the role of the stopping criterion. Moreover, the transformation estimator is a numerical method that depends on the good outcomes of the matching step. Thus, the better the choice of the correspondences that is performed, the more precise the estimation of the transformation f. Consequently, the value of the similarity metric will be more accurate, leading to a proper convergence.

Gradient Descent

In intensity-based IR, gradient-based continuous optimization algorithms are the reference methods [25]. These algorithms consist of an iterative optimization process $\mu_{k+1} = \mu_k + a_k d_k$ where d_k is the search direction at iteration k, and a_k is a gain factor that controls the step size along the search direction. The search directions and the gain factors are chosen such that the sequence μ_k converges to a local minimum of the similarity metric.

The difference between gradient-based optimizers lies in the way the search direction and the gain factor are computed. The gradient descent method (GD) takes steps in the direction of the negative gradient of the cost function, i.e., $\mu_{k+1} = \mu_k - a_k g(\mu_k)$ where g is the derivative of the cost function, and the gain factor is the decaying function $a_k = a/(k + A)^\alpha$ with $a > 0$, $A \geq 1$ and $0 \leq \alpha \leq 1$. The quasi-Newton method (QN) also moves along the negative gradient direction. The gain factor is an approximation of the inverse Hessian matrix $[H(\mu_k)]^{-1}$, computed using the Broyden–Fletcher–Goldfarb–Shanno method. The adaptive stochastic gradient descent (ASGD) [26] is based on the Robbinsâ "Monro stochastic optimization procedure, which is designed to deal with noisy observation of the objective function. ASGD considers the solutions $\mu_{k+1} = \mu_k - \gamma_k(t_k) g_k$ where $t_{k+1} = \max(0,\ t_k + \text{sigm}(-g_k \cdot g_{k-1}))$ and $\gamma_k = \frac{a}{t_k + A}$. The "time" t_k is adapted depending on the inner product between the current and the previous gradients. If the gradients have the same direction, the time is reduced, leading to a larger step size.

Metaheuristics-Based Image Registration

Suitability of MHs in Image Registration

There are different strengths and limitations that have been stated either to justify or to avoid the use of these methods when tackling IR. Some of the advantages are:

- MHs do not depend on the starting solution, thus being more robust approaches, and provide specific strategies to escape from local optima
- MHs have been used in a wide variety of optimization tasks within IR including numerical optimization and combinatorial optimization problems, i.e., facing both the transformation parameters and the matching-based IR approaches
- MHs can easily handle arbitrary kinds of constraints and objective functions
- MHs offer a framework in which including prior knowledge about the problem is easy. Thus, the search process is more appropriate, yielding a more efficient exploration of the search space. For instance, some feature-based IR approaches [7, 9, 12] improved the design of the objective function to exploit information related to the geometry of the images.
- MHs can also be easily combined with more traditional optimization techniques such as gradient-based methods [23, 45]. An outstanding approach to exploit the benefits of both strategies is their hybridization in the memetic computation paradigm [36, 37].

The most important shortcomings related to the use of MHs are:

- They require a tuning of the control parameters, which is often a manual, error-prone, expert-based procedure. This issue has been addressed using approaches based on automatic parameter tuning [51] or MHs with an adaptive behavior [36]
- Most MHs are time consuming; therefore they are usually avoided in real-time applications. Parallel and GPU implementations are increasingly more common [18, 42]
- Some MHs lack a formal proof of convergence to the global optimum, and there is hardly any theoretical result on the performance of MH. However, there is a very large amount of empirical results to support their effectiveness

Early Evolutionary Image Registration Methods

The application of MHs to medical IR enjoyed a growing interest in the scientific community over the last 15 years, as shown in Fig. 4. The first attempts at IR can be found in the early eighties. The size of data as well as the number of parameters that are looked for prevent from an exhaustive search of the solutions. An approach based on a genetic algorithm (GA) [21] was proposed in 1984 for the 2D case and applied to angiographic images [17]. Later, in 1989, Mandava et al. [33] used a 64-bit structure to represent a possible solution when trying to find the eight parameters of a bilinear transformation through a binary GA. Brunnström and Stoddart [7] proposed a new method based on the manual prealignment of range images followed by an automatic IR process using a novel GA that searches for solutions following the matching-based approach. Tsang [49] used 48-bit chromosomes to encode three test points as a base for the estimation of the 2D affine registration function by means of a binary-coded GA. In the case of Yamany et al. [57] and Chalermwat et al. [8] proposals, the same binary coding is found when dealing with 3D and

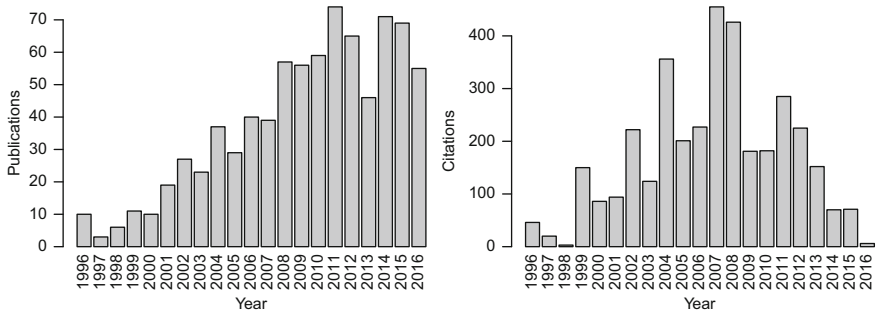


Fig. 4 The number of publications (*left*) and the number of citations (*right*) of literature contributions proposing MHs to solve medical image registration (The data shown in the graph was obtained from Thomson Reuter's Web of Science on November 29, 2016, using the query (Title OR Abstract OR Keywords) = ("image registration" OR "image alignment" OR "image matching") AND ("evolutionary algorithm" OR "evolutionary computation" OR "genetic programming" OR "genetic algorithm" OR "evolutionary programming" OR "evolution strategy" OR "differential evolution" OR "swarm intelligence" OR "bacterial foraging"))

2D rigid transformations, respectively. Yamany et al. enforced a range of $\pm 31°$ over the angles of rotation and ± 127 units in displacement by defining a 42-bit chromosome with eight bits for each translation parameter and six bits for each rotation angle. Meanwhile, Chalermwat et al. used 12 bits for the coding of the 2D rotation parameter to get a search scope of $\pm 20.48°$, therefore allowing the use of a precision factor for the discretization of the continuous rotation angle interval. Other ten bits stored each of the two translation parameters (± 512 pixels).

All the latter approaches showed several pitfalls from an evolutionary computation perspective. On the one hand, they make use of the basic binary coding to solve inherently real-coded problems, when it is well known that binary coding suffers from discretization flaws (as problem solutions of search space never visited) and requires transformations to real values for each solution evaluation. Moreover, the kind of GA considered is usually based on the old-fashioned original proposal by Holland [22]. In this way, a selection strategy based on fitness-proportionate selection probability assignment and the stochastic sampling with replacement, as well as the classical one-point crossover and simple bit flipping mutation, are used. On the one hand, it is well known that such selection strategy causes a strong selective pressure, thus having a high risk of premature convergence of the algorithm. On the other hand, it has also been demonstrated that it is difficult for the single-point crossover to create useful descendants as it is excessively disruptive with respect to the building blocks.

Outstanding IR Methods Based on MHs

This section reviews the state of the art in MH-based proposals for medical IR, grouping the algorithms according to their nature.

Feature-Based Techniques

De Falco et al.'s Differential Evolution

In [14], the authors proposed an IR method based on the differential evolution (DE) [47]. DE is a parallel direct search method that has proved to be a promising candidate to solve real-valued optimization problems. DE combines simple arithmetic operators with the classical crossover, mutation, and selection genetic operators within an easy to implement scheme. It shows the advantage of considering few control parameters, namely, mutation factor and recombination rate. The fundamental idea of DE is a new scheme for generating trial solutions by adding the weighted differenced vector between two population members to a third one. The proposed method is applied to two 2D IR problems: mosaicking and changes in time of satellite images. Registration is carried out from the transformation parameters-based approach searching for the most suitable affine transformation (given by 11 real-coded parameters) in terms of maximization of the MI similarity metric.

Lomonosov et al.'s Genetic Algorithm

In [30], the authors tackle pairwise IR of range images, facing three real-world noisy measured datasets provided by their REPLICA laser range scanner and other two from the SAMPL database. They considered the transformation parameters-based approach using rigid transformations. The main novelties of this contribution are the inclusion of a degree of overlapping parameter in the solution vector and the utilization of the trimmed squares metric as objective function to be minimized. They constitute a different schematic approach for the IR problem that offers correct coarse IR results at overlaps under 50%. A random sampling procedure is performed in order to speed up the performance of the method. According to the evolutionary design of their method, a generational GA performing search in the seven-dimensional space formed by three translation parameters, three rotation parameters, and the newly added degree of overlapping parameter is considered. They used an integer coding representation of solutions which should be properly normalized onto the corresponding real-value range. Simple one-point crossover was employed, and two mutation operators were introduced. Shift mutation alters one parameter randomly by a value not exceeding 10% of the parameter range.

Meanwhile, replacement mutation substitutes a parameter with a random value. Tournament and elitism were also employed.

Wachowiak et al.'s Particle Swarm Optimization

The authors contributed with a broad study of the performance of particle swarm optimization (PSO) algorithms [24] for solving the IR problem in biomedical applications [55]. In particular, they consider registering single slices (2D images) of 3D volumes to whole 3D volumes of medical images. In contrast to usual EAs which exploit the competitive characteristics of biological evolution (e.g., survival of the fittest), PSO exploits cooperative and social aspects, such as fish schooling, birds flocking, and insects swarming. However, both EAs and PSO approaches are considered population-based schemes. In particular, PSO algorithms start with a random population (swarm) of individuals (particles) which iteratively change their search space location by performing movements based on a velocity vector. The authors addressed the IR problem from the transformation parameters-based approach, considering a rigid transformation and the MI similarity metric as the objective function to be maximized. The variant called *PSO7* is the one achieving the best performance. It refers to a basic PSO with the following formulation for the velocity vector update $v_i(t) = \chi[v_i(t-1) + \varphi_1\mu_1(p_i - x_i(t-1)) + \varphi_2\mu_2(g - x_i(t-1))]$ with $\varphi_1 = 2.1$, $\varphi_2 = 1.3$, and the constriction coefficient $\chi = 0.7298$. However, the performance of the method is dependent on the initial orientation of the images to be registered that should be provided by the user.

Cordón et al.'s Scatter Search

The main idea behind scatter search (SS) [28] is a systematic combination between solutions (as opposed to a randomized one, usually employed in GAs) taken from a considerably reduced pool of solutions, named reference set. The fact that the mechanisms within SS are not restricted to a single uniform design allowed the authors to explore and design different strategies that demonstrated to be effective tackling point matching IR problems [12]. Furthermore, new designs for three of the five SS components – the generator of diverse solutions, the improvement, and the combination methods – were proposed to develop a method outperforming the state-of-the-art point matching approaches. In particular, the authors adopted the same feature-based approach and the same representation of solutions based on permutations previously proposed in [9]. Similarity transformations present in 3D MRIs of human brains and 3D CTs of human wrists were also considered in this work. In particular, they succeeded at dealing with significant transformations between the two registered images, one of the ICP's pitfalls.

Santamaría et al.'s Scatter Search

In [45], the authors specifically performed a broad study about the performance capabilities of different memetic IR algorithms. The authors combined three existing MH-based methods [10, 11, 14] with several local search algorithms: XLS [6], Solis and Wets [46], and Powell [41]. Two different criteria (deterministic and probabilistic) were taken for the application of the local search, and different search intensity

levels were tested. Thus, 57 different memetic IR methods were designed overall. The obtained experimental results in the 3D reconstruction of different human skull models, supported by a complementary nonparametric statistical test, revealed that the SS variant that made use of the deterministic local search application criterion and the XLS local search algorithm offered the best performance among all the developed memetic-based IR methods.

Intensity-Based Techniques

Valsecchi et al.'s Genetic Algorithm

r-GA$^+$ is an evolutionary IR method for medical imaging [50]. The optimization component of r-GA$^+$ is based on a GA with a real-coded design. A solution is encoded as a real vector, storing the transformation parameters, and the variation operators are common choices for real-coded genetic algorithms: blend crossover (BLX-α) [16] and random mutation [2]. The fitness value of a solution t is simply the similarity metric between the two images when aligned according to t. No changes are required to handle different transformation models or similarity metrics. A distinctive feature of r-GA$^+$ is the use of multiple resolutions combined with a *restart* and a search space adaptation mechanism. The key idea is that if a low-quality solution is carried over to the second resolution, the process is unlikely to recover and produce a good final solution. Therefore, restart is used at the end of the first resolution until a suitable solution if found. This process is computationally cheap, because in the first resolution the algorithm is using a small version of the imaging data, and most of the total effort is spent on the further resolutions. In addition, as the second resolution is meant to be a refinement phase, the search is focused around the best solution by restricting the range of the transformation parameters.

Valsecchi et al.'s Scatter Search

The SS$^+$ algorithm [53] is based on a two-tier SS design, meaning there are two reference sets containing, respectively, the most high-quality and diverse solutions. The methods of the SS template have been specifically chosen for IR. The diversification generation method is based on frequency memory [20] to ensure the search space is explored in a uniform manner. The solution combination method is the BLX-α crossover, while the improvement method is a local search based on a the "parent-centric" version of BLX-α called PMX-α [31]. Last, the reference set update method maintain the highest-quality solutions in the quality reference set and the most diverse solutions in the diversity one, where the diversity value of a solution x measures the distance between the solution and the quality reference set S. SS$^+$ also includes a duplication control method, in order to prevent the reference set to contain only almost identical copies of the same solution. Moreover, SS$^+$ implements the same multi-resolution strategy as r-GA$^+$, including the restart and the dynamic boundary mechanism.

Bermejo et al.'s Bacterial Foraging Optimization Algorithm

r-BFOA is an evolutionary IR method for complex 3D scenarios [3]. The global optimization process of the original BFOA proposal [38] has been adapted to the IR problem, using a hybrid approach. Likewise r-GA$^+$, a solution is encoded as a real vector, which allows the use of a recombination mechanism between two solutions by means of the crossover operator BLX-α after each chemotactic step. In addition, after each reproduction step, XLS is applied to the best bacterium in the population. A comparative study of some design variants modifying the original BFOA scheme when applied to the IR problem is presented in [4]. Even though the BFOA algorithm provides an elimination-dispersal step which reintroduces random bacteria in the population, a restart mechanism has been applied to prevent the stagnation of the bacteria. This mechanism simulates the death of all the population, reintroducing new bacteria with the aim of providing a good solution to the second resolution.

Experimental Study

The aim of the experimentation is to carry out an objective, quantitative comparison of the methods described in section "Outstanding IR Methods Based on MHs". Two separated experiments have been developed considering, respectively, feature- and intensity-based methods. We also included the traditional optimization algorithms presented in section "Metaheuristics-Based Image Registration" as baseline. See [13, 53] for a more detailed description of these comparisons.

Each experiment uses a different dataset, from which we have created a number of registration scenarios. As most of the algorithms involved are of non-deterministic nature, we carried out a number of independent runs on each scenario. The analysis measures the performance of the algorithms on each scenario by computing their mean and standard deviation values and ranking the algorithms accordingly.

Feature-Based Comparison: Registration of Brain Magnetic Resonance Images

We use a dataset from the BrainWeb public repository of the McConnell Brain Imaging Centre [27]. The BrainWeb repository is a Simulated Brain Database (SBD) providing synthetic MRI data computationally generated. Full 3D data volumes have been simulated for both models using three sequences (T1-, T2-, or proton density- (PD) weighted) and a variety of slice thickness, noise levels, and levels of intensity nonuniformity (RF). Table 1 describes the particular settings considered to generate every BrainWeb image of our study (named BW1, BW2, and BW3).

To obtain a suitable set of features, we extracted the isosurface and selected the crest-line points with relevant curvature information from the original images, using a 3D crest-line edge detector [35] (Fig. 5). The resulting datasets have between 300 and 600 points per image (see Table 1).

Table 1 Detailed description of the BrainWeb image dataset generated by the on-line SBD system

Image	Model	Modality	Noise	Protocol	RF (%)	Crest-line points
BW1	Normal	T1	0	ICBM	20	583
BW2	Mild MS	T1	1	AI	20	348
BW3	Mild MS	T1	5	AI	20	284

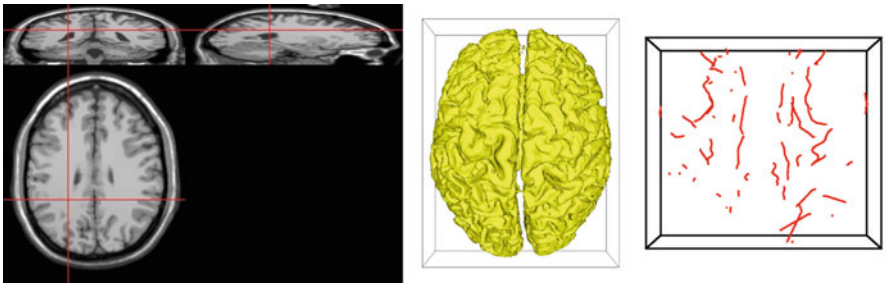


Fig. 5 From *left* to *right*: the original medical image, the corresponding extracted isosurface, and the crest line extracted from the isosurface

We considered the parameter values originally proposed by the authors in every contribution. Nevertheless, we have adapted the majority of the methods by using the same objective function in order to carry out a fair comparison. We designed several IR problem instances, taking into account similarity transformations (rotation, translation, and uniform scaling) for medical applications, thus coping with the specific characteristics of this application domain. For each problem instance tackled by the IR methods, 30 different runs are performed. Each run considers a different similarity transformation. In order to perform a fair comparison among the methods included in this study, we considered CPU time as the stop criterion. After a preliminary study, we found that 20 s was a suitable stop criterion to allow all the algorithms to converge properly.

The way a particular run is performed is as follows: a random (similarity) transformation is applied to the ground-truth image and then the IR method estimates the unknown inverse transformation. In particular, similarity transformations are randomly generated following a uniform probability distribution as follows: each of the three rotation axis parameters will be in the range $[-1, 1]$; the rotation angle will range in $[0°, 360°]$; the three translation parameters in $[-40\,\text{mm}, 40\,\text{mm}]$; the uniform scaling ranges in $[0.5, 2.0]$.

Results

The results of the first experiment are reported in Table 2. We computed the mean and standard deviation of the MSE and ranked the algorithm accordingly for each scenario. Table 3 shows the average rankings.

Table 2 Detailed results of the first experiment. For each scenario, the table reports the average MSE, standard deviation and ranking of the algorithms in the comparison

	Algorithm	MSE		Rank
		Mean	Sd	
BW1-2	Liu-ICP	2788	2364	6
	Lomonosov-GA	838	2422	5
	DeFalco-DE	132	708	2
	Wachowiak-PSO	681	1817	4
	Cordón-SS	577	1715	3
	Santamaría-SS	0.003	0.001	1
BW1-3	Liu-ICP	3009	2279	6
	Lomonosov-GA	830	2426	4
	DeFalco-DE	0.013	0.002	2
	Wachowiak-PSO	1060	2496	5
	Cordón-SS	743	1596	3
	Santamaría-SS	0.011	0.003	1
BW2-3	Liu-ICP	2929	2094	6
	Lomonosov-GA	179	857	3
	DeFalco-DE	0.026	0.001	1
	Wachowiak-PSO	645	1908	4
	Cordón-SS	1133	1785	5
	Santamaría-SS	0.028	0.004	2

Table 3 First experiment: average rankings of the algorithms

Algorithm	Mean rank
Santamaría-SS	1.33
DeFalco-DE	1.67
Cordón-SS	3.67
Lomonosov-GA	4
Wachowiak-PSO	4.33
Liu-ICP	6

The average MSE values are considerably different in one algorithm to another even when the same scenario is considered. This, together with the generally high standard deviation values, points to the fact that the algorithms can occasionally perform poorly. Nevertheless, considering the results of Liu-ICP [29] as a baseline, all algorithms performed better than the baseline on average. Indeed, Liu-ICP delivered a consistent but poor performance, ranking last on all scenarios, with MSE values just below 3000. Cordón-SS, Lomonosov-GA, and Wachowiak-PSO have slightly similar results, with MSE values in mostly the range 500–1000 and similar rankings (3.67, 4 and 4.33, respectively). Santamaría-SS and DeFalco-DE delivered much better results compared to the rest of the algorithms. Impressively, Santamaría-SS always returned MSE values below 0.01. The same applies to DeFalco-DE in BW1-3 and BW2-3, but the algorithm scored 132 on BW1-2, a much higher value but still enough to rank second on that scenario. Therefore, while the two algorithms have

quite similar rankings (1.33 and 1.67), in addition to ranking first overall, there is a clear advantage for Santamaría-SS in terms of consistency of its performance.

Second Experiment: Atlas-Based Segmentation of Brain Deep Nuclei

In the second experiment, we compare intensity-based algorithms over the registration of real brain MRI images without applying any transformation. The registration is used to perform atlas-based segmentation of deep brain structures [54]. The quality of the segmentation obtained in this phase is used to assess the effectiveness of the registration methods.

Atlas-based segmentation is a procedure that aims to automatically delineate a region of an image using an atlas (or an "average" image) of a similar subject in which the desired region has been already segmented. The first step is to register the atlas (the scene) to the input image (the model). The transformation resulting from this phase is then used to overlap the segmented region of the atlas to the scene. The region of the scene that overlaps the segmented region of the atlas is the result of the segmentation process. Figure 6 illustrates the process. Often, atlas-based segmentation is used as preliminary step in a more complex segmentation approach [34].

Thirteen T_1-weighted brain MRI were retrieved from the NMR database [40]. The deep nuclei structures in each image have been manually delineated by an expert in order to create the ground-truth data used to evaluate the registration. Figure 6 shows one of the images along with the corresponding deep nuclei. Eighteen registration scenarios were created by selecting pairs of different images at random.

The algorithms under comparison are SS^+, r-GA^+, r-BFOA, QN, and ASGD. The transformation model is affine transform, which involves rotation, translation, scaling, and shearing, and it can be represented using 12 real parameters. The stopping criterion is a time limit of 10 min. Note that no synthetic transformation is applied on the images; the registration aims to compensate for the variability in the pose of the patient during the acquisition of the images.

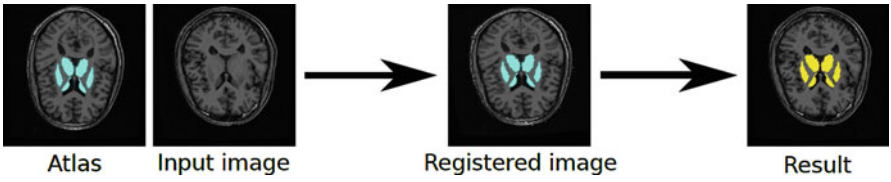


Fig. 6 The atlas-based segmentation of deep brain structures in brain MRI. First, the atlas is registered to the input image and the resulting transformation is applied to the labeled region of the atlas (*in blue*). The resulting region is overlapped on the input image (*in yellow*) to determine the output of the process

Results

The quality of atlas-based segmentation depends closely on the accuracy of the registration step, although the anatomical variability of the target region can limit its effectiveness. For each scenario we performed 32 independent runs of each algorithm. The segmented region obtained from the registration V_R is then compared with the ground-truth V_{GT}. The overlapping of the two regions is commonly measured using the Dice's coefficient [15], given by $\text{Dice}(V_R, V_{\text{GT}}) = \frac{2|V_R \cap V_{\text{GT}}|}{|V_R|+|V_{\text{GT}}|}$ where $|\cdot|$ is the number of voxels. A value of 1 means perfect overlapping, while 0 means the two regions do not overlap at all.

The results of the second experiment are reported in Table 4. We computed the mean and standard deviation of the overlap and ranked the algorithm accordingly for each scenario. Table 5 show the mean rankings for the algorithms in the comparison.

The overlap values can differ considerably across the scenarios, reflecting the fact that the effectiveness of this kind of segmentation can vary depending on the concrete anatomy of the patients. On average, r-BFOA scored the worst results in the comparison, ranking last in 10 out of 18 scenarios, with a mean rank of 4.5. High standard deviation values point to a quite uneven performance across different runs. QN ranked fourth on average, but overall the results are very inconsistent, with QN scoring in the top three in 7 out of 18 scenarios, but delivered much worse results in the remaining ones.

In general, r-GA$^+$ and ASGD have similar results, whereas SS$^+$ ranked constantly better than the others. ASGD delivered an acceptable performance, due to the low "magnitude" of the transformations involved, which is quite suitable for a local search method. However, note that the low variability of its overlap values means almost all solutions have a lower quality than the average solution found by the other two algorithms. r-GA$^+$ had a similar performance, while that of SS$^+$ is the best one of the comparison, with a mean rank of 1.61 against 2.22 and 2.94.

Conclusion

The adoption of MHs in order to solve optimization problems in the area of computer vision is steadily increasing. Medical IR is an excellent example of this trend, with feature-based being the most prevaling and mature approach. This chapter introduced the IR problem in depth and reviewed the design of some outstanding methods in the field, showing how IR have been successfully tackled using a wide spectrum of MH techniques. Despite the large interest in the community toward improving the traditional, well-known optimization methods for IR such as ICP, the experimentation in this chapter shows how algorithms based on MHs can provide superior results in terms of accuracy and robustness on challenging, real-world medical IR scenarios.

Table 4 Detailed results of the second experiment. For each scenario, the table reports the average overlap, standard deviation and ranking of the algorithms in the comparison

		MSE		
	Algorithm	Mean	Sd	Rank
1	ASGD	0.742	0.001	4
	r-BFOA	0.714	0.035	5
	r-GA$^+$	0.755	0.012	1
	QN	0.746		3
	SS$^+$	0.751	0.008	2
2	ASGD	0.679	0.003	1
	r-BFOA	0.661	0.038	5
	r-GA$^+$	0.670	0.012	4
	QN	0.677		2
	SS$^+$	0.673	0.026	3
3	ASGD	0.616	0.005	2
	r-BFOA	0.591	0.047	4
	r-GA$^+$	0.618	0.007	1
	QN	0.581		5
	SS$^+$	0.596	0.026	3
4	ASGD	0.677	0.003	3
	r-BFOA	0.622	0.064	5
	r-GA$^+$	0.679	0.008	2
	QN	0.687		1
	SS$^+$	0.677	0.008	4
5	ASGD	0.682	0.000	1
	r-BFOA	0.664	0.033	5
	r-GA$^+$	0.669	0.005	4
	QN	0.678		2
	SS$^+$	0.670	0.005	3
6	ASGD	0.691	0.001	3
	r-BFOA	0.660	0.063	5
	r-GA$^+$	0.706	0.016	2
	QN	0.685		4
	SS$^+$	0.722	0.018	1
7	ASGD	0.621	0.007	3
	r-BFOA	0.604	0.082	4
	r-GA$^+$	0.622	0.025	2
	QN	0.508		5
	SS$^+$	0.652	0.027	1
8	ASGD	0.756	0.010	3
	r-BFOA	0.715	0.048	4
	r-GA$^+$	0.760	0.009	2
	QN	0.636		5
	SS$^+$	0.773	0.012	1
9	ASGD	0.634	0.006	4
	r-BFOA	0.648	0.032	3
	r-GA$^+$	0.650	0.013	2
	QN	0.632		5
	SS$^+$	0.670	0.022	1
10	ASGD	0.738	0.003	3
	r-BFOA	0.689	0.041	5
	r-GA$^+$	0.739	0.005	2
	QN	0.717		4
	SS$^+$	0.740	0.011	1
11	ASGD	0.717	0.011	3
	r-BFOA	0.709	0.025	4
	r-GA$^+$	0.728	0.010	2
	QN	0.544		5
	SS$^+$	0.750	0.013	1
12	ASGD	0.684	0.005	4
	r-BFOA	0.683	0.066	5
	r-GA$^+$	0.688	0.008	3
	QN	0.702		2
	SS$^+$	0.716	0.022	1
13	ASGD	0.686	0.009	4
	r-BFOA	0.673	0.033	5
	r-GA$^+$	0.729	0.004	1
	QN	0.696		3
	SS$^+$	0.718	0.014	2
14	ASGD	0.680	0.005	3
	r-BFOA	0.665	0.031	4
	r-GA$^+$	0.680	0.011	2
	QN	0.629		5
	SS$^+$	0.693	0.019	1

(*continued*)

Table 4 continued

15	ASGD	0.741	0.001	4
	r-BFOA	0.690	0.041	5
	r-GA$^+$	0.750	0.007	3
	QN	0.764		2
	SS$^+$	0.769	0.023	1
16	ASGD	0.751	0.012	3
	r-BFOA	0.743	0.061	4
	r-GA$^+$	0.755	0.029	2
	QN	0.668		5
	SS$^+$	0.779	0.011	1
17	ASGD	0.754	0.004	2
	r-BFOA	0.705	0.034	5
	r-GA$^+$	0.749	0.014	3
	QN	0.736		4
	SS$^+$	0.756	0.020	1
18	ASGD	0.624	0.004	3
	r-BFOA	0.624	0.082	4
	r-GA$^+$	0.633	0.041	2
	QN	0.613		5
	SS$^+$	0.689	0.030	1

Table 5 Second experiment: the average rankings of the algorithms

Algorithm	Mean rank
SS$^+$	1.61
r-GA$^+$	2.22
ASGD	2.94
QN	3.72
r-BFOA	4.50

Cross-References

- Evolutionary Algorithms
- Genetic Algorithms
- Particle Swarm Methods
- Scatter Search

Acknowledgments This work is supported by the Spanish "Ministerio de Economía y Competitividad" under the NEWSOCO project (ref. TIN2015-53067661) and the Andalusian Department of Innovación, Ciencia y Empresa under project TIC2011-7745, both including European Regional Development Funds (ERDF).

References

1. Audette MA, Ferrie FP, Peters TM (2000) An algorithmic overview of surface registration techniques for medical imaging. Med Image Anal 4(3):201–217
2. Bäck T, Fogel DB, Michalewicz Z (1997) Handbook of evolutionary computation. IOP Publishing Ltd/Oxford University Press, Bristol
3. Bermejo E, Cordón O, Damas S, Santamaría J (2013) Quality time-of-flight range imaging for feature-based registration using bacterial foraging. Appl Soft Comput 13(6):3178–3189
4. Bermejo E, Cordón O, Damas S, Santamaría J (2015) A comparative study on the application of advanced bacterial foraging models to image registration. Inform Sci 295:160–181
5. Besl PJ, McKay ND (1992) A method for registration of 3D shapes. IEEE Trans Pattern Anal Mach Intell 14:239–256
6. Beyer HG, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. IEEE Trans Evol Comput 5(3):250–270

7. Brunnström K, Stoddart A (1996) Genetic algorithms for free-form surface matching. In: International conference of pattern recognition, Vienna, pp 689–693
8. Chalermwat P, El-Ghazawi T, LeMoigne J (2001) 2-phase GA-based image registration on parallel clusters. Future Gener Comput Syst 17:467–476
9. Cordón O, Damas S (2006) Image registration with iterated local search. J Heuristics 12:73–94
10. Cordón O, Damas S, Santamaría J (2006) A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm. Pattern Recogn Lett 27(11): 1191–1200
11. Cordón O, Damas S, Santamaría J (2006) Feature-based image registration by means of the CHC evolutionary algorithm. Image Vision Comput 22:525–533
12. Cordón O, Damas S, Santamaría J, Martí R (2008) Scatter search for the 3D point matching problem in image registration. INFORMS J Comput 20:55–68
13. Damas S, Cordón O, Santamaría J (2011) Medical image registration using evolutionary computation: an experimental survey. IEEE Comput Intell Mag 6(4):26–42
14. De Falco I, Della Cioppa A, Maisto D, Tarantino E (2008) Differential evolution as a viable tool for satellite image registration. Appl Soft Comput 8(4):1453–1462
15. Dice LR (1945) Measures of the amount of ecologic association between species. Ecology 26(3):297–302
16. Eshelman LJ (1993) Real-coded genetic algorithms and interval schemata. In: Whitley LD (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo, pp 187–202
17. Fitzpatrick J, Grefenstette J, Gucht D (1984) Image registration by genetic search. In: IEEE Southeast conference, Louisville, pp 460–464. EEUU
18. Fok K, Wong T, Wong M (2007) Evolutionary computing on consumer graphics hardware. IEEE Intell Syst 22(2):69–78
19. Glover F, Kochenberger GA (eds) (2003) Handbook of metaheuristics. Kluwer Academic Publishers, Boston
20. Glover F, Laguna M, Martí R (2003) Scatter search. In: Ghosh A, Tsutsui S (eds) Advances in evolutionary computation: theory and applications. Springer, New York, pp 519–537
21. Goldberg DE (1989) Genetic algoritms in search and optimization. Addison-Wesley, New York. EEUU
22. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
23. Jenkinson M, Smith S (2001) A global optimisation method for robust affine registration of brain images. Med Image Anal 5(2):143–156
24. Kennedy J, Eberhart R (2001) Swarm intelligence. Morgan Kaufmann, San Francisco
25. Klein S, Staring M, Pluim JPW (2007) Evaluation of optimization methods for nonrigid medical image registration using mutual information and b-splines. IEEE Trans Image Process 16(12):2879–2890
26. Klein S, Pluim J, Staring M, Viergever M (2009) Adaptive stochastic gradient descent optimisation for image registration. Int J Comput Vis 81:227–239
27. Kwan RKS, Evans AC, Pike GB (1999) MRI simulation-based evaluation of image-processing and classification methods. IEEE Trans Med Imaging 18(11):1085–1097
28. Laguna M, Martí R (2003) Scatter search: methodology and implementations in C. Kluwer Academic Publishers, Boston
29. Liu Y (2004) Improving ICP with easy implementation for free form surface matching. Pattern Recognit 37(2):211–226
30. Lomonosov E, Chetverikov D, Ekart A (2006) Pre-registration of arbitrarily oriented 3D surfaces using a genetic algorithm. Pattern Recogn Lett 27(11):1201–1208
31. Lozano M, Herrera F, Krasnogor N, Molina D (2004) Real-coded memetic algorithms with crossover hill-climbing. Evolut Comput 12(3):273–302
32. Maes F, Vandermeulen D, Suetens P (1999) Comparative evaluation of multiresolution optimization strategies for image registration by maximization of mutual information. Med Image Anal 3(4):373–386
33. Mandava VR, Fitzpatrick JM, Pickens DR (1989) Adaptive search space scaling in digital image registration. IEEE Trans Med Imaging 8(3):251–262

34. Mesejo P, Valsecchi A, Marrakchi-Kacem L, Cagnoni S, Damas S (2015) Biomedical image segmentation using geometric deformable models and metaheuristics. Comput Med Imaging Graph 43:167–178
35. Monga O, Deriche R, Malandain G, Cocquerez JP (1991) Recursive filtering and edge tracking: two primary tools for 3D edge detection. Image Vision Comput 9(4):203–214
36. Ong YS, Lim M, Zhu N, Wong K (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst Man Cybern 36(1):141–152
37. Ong YS, Lim MH, Chen X (2010) Memetic computation – past, present & future. IEEE Comput Intell Mag 5(2):24–31
38. Passino K (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst 22(3):52–67
39. Pluim JPW, Maintz JBA, Viergever MA (2003) Mutual-information-based registration of medical images: a survey. IEEE Trans Med Imaging 22(8):986–1004
40. Poupon C, Poupon F, Allirol L, Mangin JF (2006) A database dedicated to anatomo-functional study of human brain connectivity. In: 12th annual meeting of the organization for human brain mapping, Florence, vol 646
41. Powell MJD (1964) An efficient method for finding the minimum of a function of several variables without calculating derivatives. Comput J 7(2):155–162
42. Robertson C, Fisher RB (2002) Parallel evolutionary registration of range data. Comput Vis Image Underst 87:39–50
43. Rouet JM, Jacq JJ, Roux C (2000) Genetic algorithms for a robust 3-D MR-CT registration. IEEE Trans Inf Technol Biomed 4(2):126–136
44. Rusinkiewicz S, Levoy M (2001) Efficient variants of the ICP algorithm. In: Third international conference on 3D digital imaging and modeling (3DIM 2001), Quebec, pp 145–152
45. Santamaría J, Cordón O, Damas S, García-Torres J, Quirin A (2009) Performance evaluation of memetic approaches in 3D reconstruction of forensic objects. Soft Comput 13(8–9):883–904
46. Solis FJ, Wets RJB (1981) Minimization by random search techniques. Math Oper Res 6(1):19–30
47. Storn R (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359
48. Svedlow M, Mc-Gillem CD, Anuta PE (1976) Experimental examination of similarity measures and preprocessing methods used for image registration. In: Swain P, Morrison D, Parks D (eds) Symposium on machine processing of remotely sensed data, Indiana, vol 4(A), pp 9–17
49. Tsang PWM (1997) A genetic algorithm for aligning object shapes. Image Vis Comput 15: 819–831
50. Valsecchi A, Damas S, Santamaría J, Marrakchi-Kacem L (2013) Genetic algorithms for voxel-based medical image registration. In: IEEE fourth international workshop on computational intelligence in medical imaging (CIMI 2013), pp 22–29
51. Valsecchi A, Dubois-Lacoste J, Stützle T, Damas S, Santamaría J, Marrakchi-Kacem L (2013) IEEE congress on evolutionary medical image registration using automatic parameter tuning. In: Evolutionary computation (CEC 2013), pp 1326–1333
52. Valsecchi A, Damas S, Santamaría J (2014) Evolutionary intensity-based medical image registration: a review. Curr Med Imaging Rev 10:283–297
53. Valsecchi A, Damas S, Santamaría J, Marrakchi-Kacem L (2014) Intensity-based image registration using scatter search. Artif Intell Med 60(3):151–163
54. Vemuri BC, Ye J, Chen Y, Leonard CM (2003) Image registration via level-set motion: applications to atlas-based segmentation. Med Image Anal 7(1):1–20
55. Wachowiak MP, Smolikova R, Zheng Y, Zurada JM, El-Maghraby AS (2004) An approach to multimodal biomedical image registration utilizing particle swarm optimization. IEEE Trans Evol Comput 8(3):289–301
56. Wang XY, Eberl S, Fulham M, Som S, Feng DD (2008) Data registration and fusion. In: Feng DD (ed) Biomedical information technology. Academic Press, Burlingto, pp 187–210

57. Yamany SM, Ahmed MN, Farag AA (1999) A new genetic-based technique for matching 3D curves and surfaces. Pattern Recognit 32:1817–1820
58. Zambanini S, Sablatnig R, Maier H, Langs Gd (2010) Automatic image-based assessment of lesion development during hemangioma follow-up examinations. Artif Intell Med 50(2):83–94
59. Zhang Z (1994) Iterative point matching for registration of free-form curves and surfaces. Int J Comput Vis 13(2):119–152
60. Zitová B, Flusser J (2003) Image registration methods: a survey. Image Vis Comput 21: 977–1000

Metaheuristics for Natural Gas Pipeline Networks

37

Roger Z. Ríos-Mercado

Contents

Introduction 1104
Problem Description and Modeling Framework 1105
Background 1105
Description of Basic Model 1106
Network Topology 1108
Solution Techniques: Classical Approaches 1108
Tabu Search: An Approach for NLP Models 1111
Nonsequential Dynamic Programming 1112
The Tabu Search Approach 1112
Ant Colony Optimization: An Approach for MINLP Models 1114
Metaheuristic Approaches to Related Problems 1117
Particle Swarm Optimization for Non-isothermal Systems 1117
Simulated Annealing for Time-Dependent Systems 1118
Conclusion 1118
Cross-References 1119
References 1119

Abstract

In this chapter an overview of metaheuristic algorithms that have been very successful in tackling a particular class of natural gas pipeline network optimization problems is presented. In particular, the problem of minimizing fuel consumption incurred by the compressor stations driving natural gas in pipeline networks is addressed. This problem has been studied from different angles over the past few years by virtue of its tremendous economical impact. First, a general mathemati-

R. Z. Ríos-Mercado (✉)
Graduate Program in Systems Engineering, Universidad Autónoma de Nuevo León (UANL), San Nicolás de los Garza, Mexico
e-mail: roger.rios@uanl.edu.mx

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_51

cal framework for this class of problems is presented. After establishing the most relevant model properties and fundamental network topologies, which are key factors for choosing an appropriate solution technique, current state-of-the-art metaheuristics are presented for handling different versions of this problem. This work concludes by highlighting the most relevant and important challenges of this very exciting area of research in natural gas transportation networks.

Keywords

Natural gas transmission systems · Pipeline optimization · Nonlinear programming · Mixed-integer nonlinear programming · Tabu search · Ant colony optimization · Simulated annealing · Particle swarm optimization

Introduction

There are many interesting decision-making problems in the natural gas industry that have been studied over the years. These include fields such as pipeline design, gas storage, gathering, transportation, and marketing, to name a few. An important class of these problems, referred to as the minimum fuel consumption problem (MFCP), deals with how to operate a natural gas transportation network for delivering the gas from storage facilities to local distribution companies so as to minimize the fuel consumption employed by the compressor stations moving the gas along the network. Efficient design and operation of these complex networks can substantially reduce airborne emissions, increase safety, and decrease the very high daily operating costs due to the large amounts of fuel per day needed to operate the compressor stations driving the gas.

These types of networks are very complex and highly nonlinear since the relationship between the flow variables in every arc and the pressure values at the interconnection points is represented by nonlinear equations and, in some cases, by partial differential equations. Thus, in general the class of MFCPs is very challenging due to the presence of nonlinearities and nonconvexities in the models representing such problems. These problems have been studied since the late 1960s from many different angles, most of them based on classical hierarchical control and mathematical programming approaches.

It was until very recently that metaheuristics techniques were introduced for addressing some of these problems. One of the great advantages of metaheuristic algorithms over existing approaches is that the former do not depend on gradient-based information so they can handle the nonlinear and nonconvex nature of the problems with relative ease. Furthermore, they can be combined with existing mathematical programming approaches in intelligent ways to derive hybrid metaheuristic methods.

The purpose of this chapter is to introduce the reader with an important class of challenging optimization problems in natural gas transportation networks and to give a detailed discussion on how metaheuristics have been successfully applied on addressing these problems. There are of course other important decision-making

problems in the natural gas industry for which optimization techniques have made important contributions. A recent paper by Zheng et al. [40] surveys optimization models in the natural gas industry, focusing on natural gas production, transportation, and marketing. Ríos-Mercado and Borraz-Sánchez [29] present an extensive review of classical techniques for fuel consumption minimization in transmission systems, including gathering, transmission, and local distribution. Schmidt et al. [32] present stationary nonlinear programming models of gas networks that are primarily designed to include detailed nonlinear physics in the final optimization steps for midterm planning problems.

This chapter is organized as follows. The basic mathematical framework for the steady-state case, including important model properties, is presented in the first section. Then, in the following section, the existing classical approaches for handling different versions of this problem including steady-state and transient models are briefly highlighted. This is followed by two sections where both a Tabu Search algorithm for handling a nonlinear programming and an Ant Colony Optimization algorithm for handling a mixed-integer nonlinear programming model are described. Other metaheuristic approaches for related problems in the natural gas industry are reviewed next. Final remarks and discussion of future research trends about metaheuristic techniques in natural gas optimization problems are given in the last section.

Problem Description and Modeling Framework

Background

The main purpose of a natural gas transmission network is to transport gas from storage facilities to local distribution companies. The gas is moved by pressure, and pressure is lost due to the friction of the gas flow with the inner wall of the pipelines. Thus, to keep the gas moving, compressor stations, whose primary role is to increase gas pressure, are needed. In turn, every compressor station is composed of several compressor units. These units may be identical or not and hooked up in different ways. The most typical configuration, which is assumed throughout this chapter, is that of identical compressor units hooked-up in parallel. It is well known that most of the operating costs in a pipeline network are due to the amount of fuel consumed at the compressor stations.

When operating a natural gas transmission system aiming at minimizing fuel consumption, there are two main groups of decision variables that must be considered: (i) the mass flow rate through every pipe and compressor station and (ii) gas pressure values in each interconnection point. Additionally, decisions such as how many individual compressor units to operate within stations may be taken. Hence, the objective for a transmission network is to minimize the total fuel consumption of the compressor stations while satisfying specified delivery flow rates and minimum pressure requirements at the delivery terminals. The MFCP is typically modeled as a

nonlinear or mixed-integer nonlinear network optimization problem. It is of course assumed that the network is given, that is, this is not a design problem.

Depending on how the gas flow changes with respect to time, we distinguish between systems in steady state and transient state. A system is said to be in steady state when the values characterizing the flow of gas in the system are independent of time. In this case, the system constraints, particularly the ones describing the gas flow through the pipes, can be described using algebraic nonlinear equations. In contrast, transient analysis requires the use of partial differential equations (PDEs) to describe such relationships. This makes the problem considerably harder to solve from the optimization perspective. In fact, optimization of transient models is one of the most challenging ongoing research areas. In the case of transient optimization, variables of the system, such as pressures and flows, are functions of time.

Gas transmission network problems differ from traditional network flow problems in some fundamental aspects. First, in addition to the flow variables for each arc, which in this case represent mass flow rates, a pressure variable is defined at every node. Second, besides the mass balance constraints, there exist two other types of constraints: (i) a nonlinear equality constraint on each pipe, which represents the relationship between the pressure drop and the flow, and (ii) a nonlinear nonconvex set which represents the feasible operating limits for pressure and flow within each compressor station. The objective function is given by a nonlinear function of flow rates and pressures. The problem is very difficult due to the presence of a nonconvex objective function and a nonconvex feasible region.

Description of Basic Model

Let $G = (V, A)$ be a directed graph representing a natural gas transmission network, where V is the set of nodes representing interconnection points, and A is the set of arcs representing either pipelines or compressor stations. Let V_s and V_d be the set of supply and demand nodes, respectively. Let $A = A_p \cup A_c$ be partitioned into a set of pipeline arcs A_p and a set of compressor station arcs A_c. That is, $(u, v) \in A_c$ if and only if u and v are the input and output nodes of compressor station (u, v), respectively.

Two types of decision variables are defined: let x_{uv} denote the mass flow rate at arc $(u, v) \in A$, and let p_u denote the gas pressure at node $u \in V$. The following parameters are assumed known: B_u is the net mass flow rate in node u, and P_u^L and P_u^U are the pressure limits (lower and upper) at node u. By convention, $B_u > 0$ ($B_u < 0$) if $u \in V_s$ ($u \in V_d$), and $B_u = 0$ otherwise.

The basic mathematical model of the minimum fuel cost problem (MFCP) is given by:

$$\text{Minimize} \quad g(x, p) = \sum_{(u,v)\in A_c} g_{uv}(x_{uv}, p_u, p_v) \tag{1}$$

$$\text{subject to} \quad \sum_{v:(u,v)\in A} x_{uv} - \sum_{v:(v,u)\in A} x_{vu} = B_u \qquad u \in V \tag{2}$$

$$(x_{uv}, p_u, p_v) \in D_{uv} \qquad (u,v) \in A_c \tag{3}$$

$$x_{uv}^2 = R_{uv}(p_u^2 - p_v^2) \qquad (u,v) \in A_p \tag{4}$$

$$p_u \in [P_u^L, P_u^U] \qquad u \in V \tag{5}$$

$$x_{uv} \geq 0 \qquad (u,v) \in A \tag{6}$$

The objective function (1) measures the total amount of fuel consumed in the system, where $g_{uv}(x_{uv}, p_u, p_v)$ denotes the fuel consumption cost at compressor station $(u,v) \in A_c$. For a single compressor unit, the following function is typically used:

$$g^{(1)}(x_{uv}, p_u, p_v) = \frac{\alpha x_{uv}}{\eta}\left\{\left(\frac{p_v}{p_u}\right)^m - 1\right\},$$

where α and m are assumed constant and known parameters that depend on the gas physical properties, and η is the adiabatic efficiency coefficient. This adiabatic coefficient is a function of (x_{uv}, p_u, p_v), that is, in general, a complex expression implicitly defined. A function evaluation of η requires solving a linear system of algebraic equations. In practice, though, polynomial approximation functions that fit the function relatively well and are simpler to evaluate are employed. In other cases, when the fluctuations of η are small enough, η can be assumed to be a constant.

For a compressor station (u,v) with n_{uv} identical compressor units hooked-up in parallel which is very commonly found in industry, the fuel consumption is given by:

$$g_{uv}(x_{uv}, p_u, p_v) = n_{uv} g^{(1)}(x_{uv}/n_{uv}, p_u, p_v). \tag{7}$$

When all n_{uv} units are fixed and operating, we have a nonlinear programming (NLP) model. Treating n_{uv}, as decision variables, leads to mixed-integer nonlinear programming (MINLP) models.

Constraints (2) establish the mass balance at each node. Constraints (3) denote the compressor operating limits, where D_{uv} denote the feasible operating domain for compressor $(u,v) \in A_c$. Equations (4) express the relationship between the mass flow rate through a pipe and its pressure values at the end points under isothermal and steady-state assumptions, where R_{uv} (also known as the pipeline resistance

parameter) is a parameter that depends on both the physical characteristics of the pipeline and gas physical properties. When the steady-state assumption does not hold, this relationship is a time-dependent partial differential equation which leads to transient models. Constraints (5) set the lower and upper limits of the pressure value at every node, and (6) set the non-negativity condition of the mass flow rate variables. Further details of this model can be found in Wu et al. [38] and Ríos-Mercado [28].

Network Topology

There are three different kinds of network topologies: (a) linear or gun barrel, (b) tree or branched, and (c) cyclic. Technically, the procedure for making this classification is as follows. In a given network, the compressor arcs are temporarily removed. Then each of the remaining connected components is merged into a big supernode. Finally, the compressor arcs are put back into their place. This new network is called the *associated reduced network*. Figure 1 illustrates the associated reduced network for a 12-node, 11-arc example. As can be seen, the reduced network has four supernodes (labeled S1, S2, S3, S4) and three arcs (the compressor station arcs from the original network).

Types of network topologies:

Linear topology: reduced network is a single path.
Tree topology: reduced network is a tree.
Cyclic topology: reduced network has cycles (either directed or undirected).

These different types of network topologies are shown in Fig. 2, where the original network is represented by solid line nodes and arcs and the reduced network by dotted super nodes. Note that even though networks in Fig. 2a, b are not acyclic from a strict network definition, they are considered as noncyclic pipeline network structures.

Solution Techniques: Classical Approaches

There is certainly a number of different optimization techniques that have been tried in the past to address problems in fuel cost minimization of natural gas transportation networks.

Steady-state NLP models: most of the work for nonconvex NLP models has been based on steady-state models. One can find work on dynamic programming-based techniques [3, 6, 13, 15, 17, 31, 36, 37], methods based on gradient search [12, 26], global optimization methods [13], linearization techniques [9], and model properties and lower bounding schemes [5, 30, 38].

Steady-state MINLP models: there has also been studies on developing optimization methods for addressing MINLP models. In most of these models, integer variables for deciding which individual compressor units must be operating

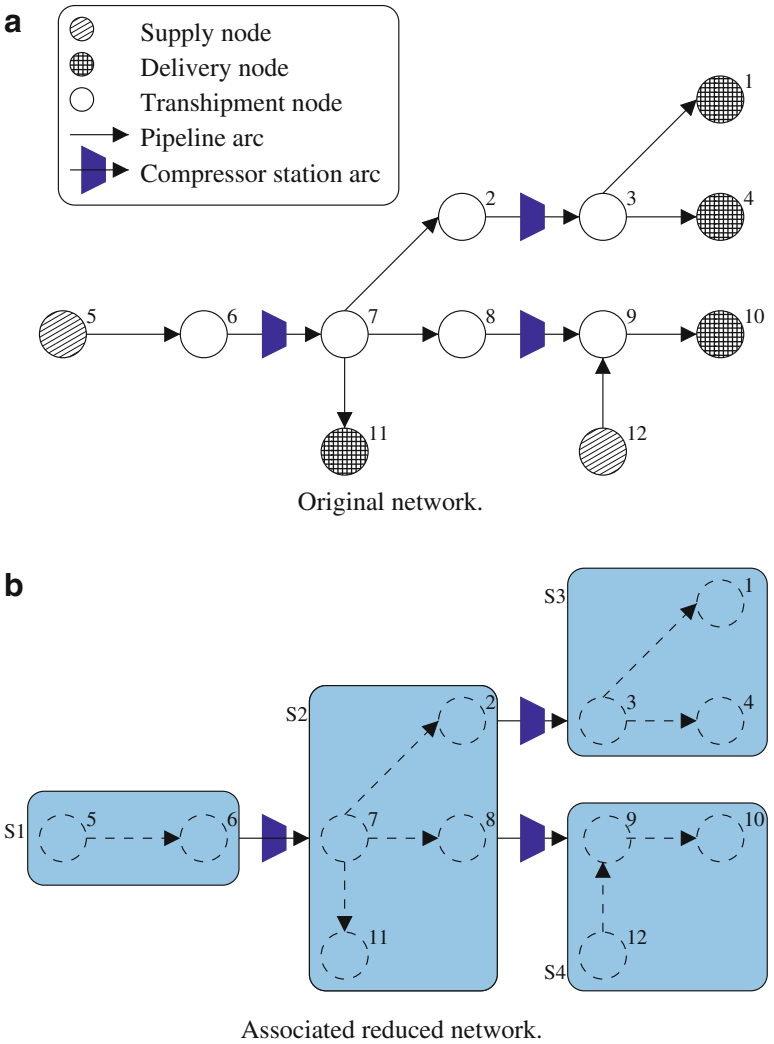


Fig. 1 Illustration of a reduced network. (**a**) Original network. (**b**) Associated reduced network

within a compressor station are introduced. Solution methodologies include mainly successive branch and bound [27, 34], outer approximation with augmented penalty [8], and linearization techniques [20].

Transient models: Transient models are more challenging as the governing PDEs associated with the dynamics of the gas system must be taken into consideration. Efforts on addressing this class of very difficult problems include hierarchical control techniques [2, 16, 21–23, 25] in the early years and mathematical programming approaches [1, 10, 11, 14, 19, 24, 35], more recently.

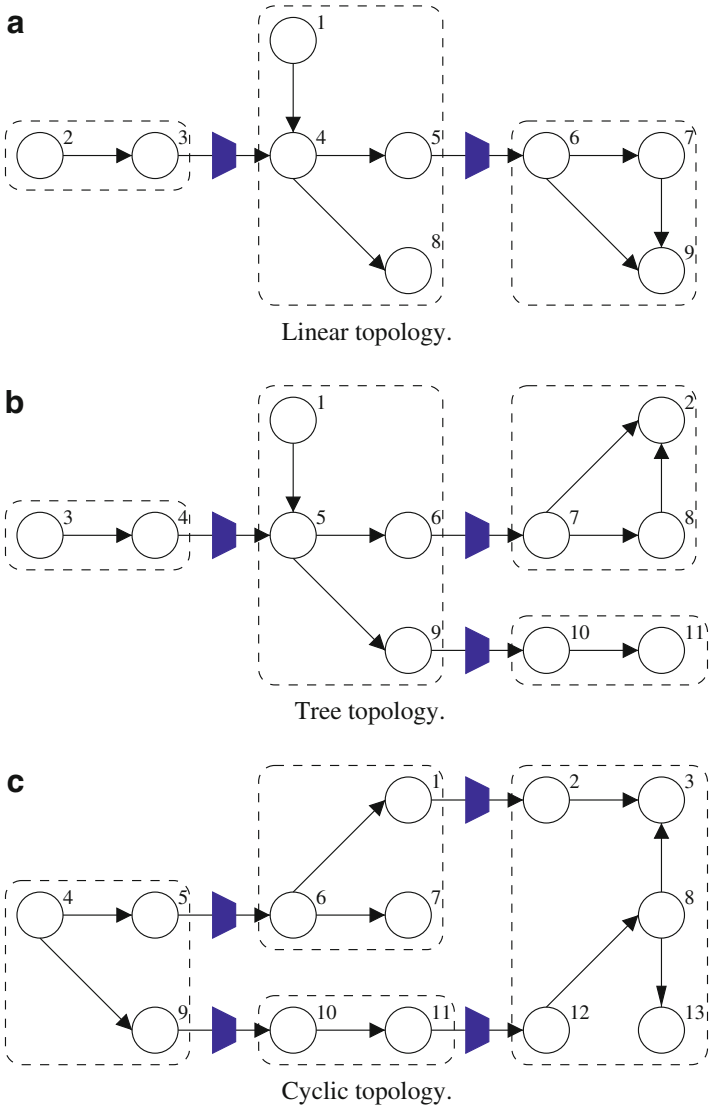


Fig. 2 Different kinds of pipeline network topologies. (**a**) Linear topology. (**b**) Tree topology. (**c**) Cyclic topology

For a complete literature review and detailed discussion of some of these techniques, the reader is referred to the recent surveys of Zheng et al. [40] and Ríos-Mercado and Borraz-Sánchez [29]. In the following sections, we review the most successful metaheuristic techniques applied to variations of the MFCP.

Tabu Search: An Approach for NLP Models

For the past few years, tabu search (see ▸ Chap. 25, "Tabu Search") has established its position as an effective metaheuristic guiding the design and implementation of algorithms for the solution of hard combinatorial optimization problems in a number of different areas. A key reason for this success is the fact that the algorithm is sufficiently flexible to allow designers to exploit prior domain knowledge in the selection of parameters and subalgorithms. Another important feature is the integration of memory-based components.

When addressing the MFCP, even though we are dealing with a continuous optimization problem, tabu search (TS), with an appropriate discrete solution space, is a very attractive choice due to the the nonconvexity of the objective function and the versatility of TS to overcome local optimality.

We now describe the TS-based approach of Borraz-Sánchez and Ríos-Mercado [4, 5], which is regarded as the most successful implementation of a metaheuristic for the MFCP. This TS takes advantage of the particular problem structure and properties and in fact can be regarded as a hybrid metaheuristic or matheuristic (see ▸ Chap. 5, "Matheuristics").

Let us consider the MFCP model given by (1), (2), (3), (4), (5) and (6), that is, the number of compressor units operating in each compressor station is known and fixed in advance. As established earlier, this is a nonconvex NLP. Current state of the art on solution techniques for this MFCP reveals these important facts:

- There are theoretical results indicating that in noncyclic systems, the values of the flow variables can be uniquely determined and fixed beforehand [30]. Therefore, the problem reduces to finding out the optimal set of pressure variables at each node in the network. Of course, the problem is still hard to solve, but it reduces its dimension in terms of the decision variables.
- As a direct consequence of this, there exists successful implementations mostly based on dynamic programming (DP) that efficiently solve the problem in noncyclic instances by appropriately discretizing the pressure variables.
- When in a cyclic system, we impose the limitation of fixing the flow variables in each arc, a nonsequential dynamic programming (NDP), developed by Carter [6], can been successfully applied for finding the optimal set of pressure variables. Although this algorithm has the limitation of narrowing the set of solutions to those subject to a fixed set of flows, it can be used within other flow-modification-based approaches.

It is clear that the TS approach is aiming at finding high-quality solutions for cyclic systems. It exploits the fact that for a given set of flows, an optimal set of pressure values can be efficiently found by NDP.

Nonsequential Dynamic Programming

We include in this section a brief description of the essence of the NDP algorithm. Further details can be found in [4]. Starting with a feasible set of flow variables, the NDP algorithm searches for the optimal set of node pressure values associated with that pre-specified flow. Rather than attempting to formulate DP as a recursive algorithm, at a given iteration, the NDP procedure grabs two connected compressors and replace them by a "virtual" composite element that represents the optimal operation of both compressors. These two elements can be chosen from anywhere in the system, so the idea of "sequential recursion" in classical DP does not quite apply here. After performing this step at a stage t, the system with t compressor stations has been replaced by an equivalent system with $t-1$ stations. The procedure continues until only one virtual element, which fully characterizes the optimal behavior of the entire pipeline system, is left. Afterwards, the optimal set of pressure variables can be obtained by a straightforward backtracking process. The computational complexity of this NDP technique is $O(|A_c|N_p^2)$, where N_p is the maximum number of elements in a pressure range discretization.

The Tabu Search Approach

Algorithm 1: Pseudocode of Procedure TS

Require: An instance of the MFCP.
Ensure: A feasible solution (X, P).
1: $(X, P)^{\text{best}} \leftarrow \emptyset$
2: TabuList $\leftarrow \emptyset$
3: $\bar{X} \leftarrow$ FindInitialFlow( )
4: **while** (stopping criteria not met) **do**
5: Generate neighborhood $V(\bar{X})$
6: **for** ($X \in V(\bar{X})$ such that $X \notin$ TabuList) **do**
7: $P \leftarrow$ NDP(X4)
8: **end for**
9: Choose best (non-tabu) solution (X, P)
10: **if** (|TabuList| == *TabuTenure*) **then**
11: Remove oldest element from TabuList
12: **end if**
13: TabuList $\leftarrow$ TabuList $\cup X$
14: $(X, P)^{\text{best}} \leftarrow$ Best((X, P), $(X, P)^{\text{best}}$)
15: **end while**
16: Return $(X, P)^{\text{best}}$

The main steps of the algorithm are shown in Procedure 1. Here, a solution $Y = (X, P)$ is partitioned into its set of flow variables X and set of pressure variables P. First note that the search space employed by TS is defined by the flow variables

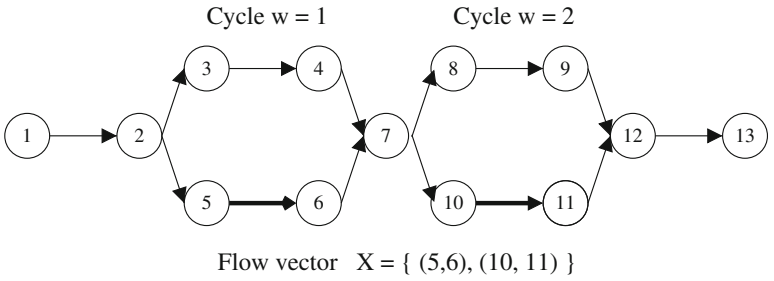


Fig. 3 Flow components of a feasible solution on a cyclic topology

X only because once the flow rates are fixed, the corresponding pressure variables are optimally found by NDP. Furthermore, we do not need to handle the entire set of flow variables, but only one per cycle. This is so because once you fix a flow rate in a cycle, the rest of the flows can be uniquely determined. Thus, a given state is represented by a vector $\hat{X} = (X_{\alpha_1}, \ldots, X_{\alpha_m})$, where α_w is an arc that belongs to a selected cycle w. Note that this set of arcs is arbitrarily chosen and that converting a flow from X to and from $\hat{X}$ is straightforward, so in the description X and $\hat{X}$ are used interchangeably. This situation is illustrated in Fig. 3. The network represents the associated reduced network. It is clear that given a specified amount of net flow entering at node 1, only one arc in each cycle is needed to uniquely determine the flows in each arc of the network. In this case, the bold arcs (5,6) and (10,11), one per cycle, suffice.

We now describe each component.

- *Initial set of flows*: first, in Step 3, in initial set of feasible flows is found. Here, different methods such as classical assignment techniques can be applied in a straightforward manner.
- *Neighborhood definition:* In Step 5, a neighborhood $V(\bar{X})$ of a given solution $\bar{X} = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m\}$ is defined as the set of solutions reachable from $\bar{X}$ via a modification of the current flow in each arc by Δ_x units in each of its components. This is given by

$$V(\bar{x}) = \{X' \in R^m \mid x'_w = \bar{x}_w \pm k\Delta_x, k = 1, 2, \ldots, N/2,\ w = 1, \ldots, m\}$$

where N is the predefined neighborhood size, Δ_x accounts for the mesh size, and the index w refers to the w-th cyclic component. Note that, for a given solution, the entire solution does not need to be stored but only the flow in the selected arc component to be modified. Note also that once this value is set, the rest of the flow variables in the cycle are easily determined, so in this sense, it is precisely this mass flow rate which becomes the attribute of the solution.

- *Optimal pressure values*: in Steps 6–8, the corresponding set of pressure values for the given flow is found by invoking the NDP algorithm only for those flow values that are non-tabu.
- *Tabu list:* then in Step 9, the best $X' \in V(\bar{X})$ which is non-tabu is chosen, and the corresponding subsets are updated accordingly. A tabu list (TabuList) stores recently used attributes, in our case, values of the X variables. The size of the TabuList (*TabuTenure*) controls the number of iterations; a particular attribute is kept in the list.
- *Stopping criterion*: the search usually terminates after a given number of iterations , or when no significant change has been found in certain number of iterations.

As we know from theoretical properties of pipeline networks [30], the flow modification step is unnecessary for noncyclic topologies because there exists a unique set of optimal flow values which can be determined in advance at preprocessing.

The algorithm was tested on several cyclic real-world size instances of up to 19 supernodes and seven compressor stations with excellent results. The method significantly outperformed the best earlier approaches finding solutions of very good quality relatively quickly.

Ant Colony Optimization: An Approach for MINLP Models

Let us consider now the problem where, in addition to the flow variables in each arc and the pressure variables in each node, the decision process involves determining the number of operating units in each compressor as well. This leads to an MINLP model. In this section, the ant colony optimization (ACO) algorithm by Chebouba et al. [7] for this version of the MFCP is described.

Ant Colony Optimization (see ▸ Chap. 13, "Ant Colony Optimization: A Component-Wise Overview") is a relatively new evolutionary optimization method that has been successfully applied to a number of combinatorial optimization problems. ACO is based on the communication of a colony of simple agents (called ants), mediated by (artificial) pheromone trails. The main source of ACO is a pheromone trail laying and following behavior of real ants which use pheromones as a communication medium. The pheromone trails in ACO serve as distributed, numerical information which the ants use to probabilistically construct solutions to the problem being solved and which the ants adapt during the algorithm's execution to reflect its search experience.

Regarding natural gas pipeline network optimization, Chebouba et al. [7] present an ACO metaheuristic for the MFCP with a variable number of compressor units within a compressor station. They focus on the linear topology case. As it was mentioned earlier, solving the MFCP on linear topologies has been successfully addressed by dynamic programming approaches when the number of compreessor units is fixed and known; however, when the number of individual compressor units

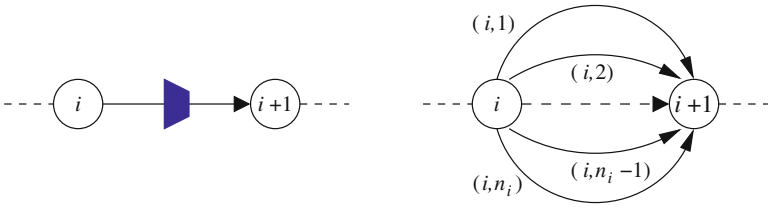


Fig. 4 Modeling compressor unit choices as a multigraph

is variable and part of the decision process, it leads to an MINLP that has a higher degree of difficulty.

Consider the MINLP given by objective function (7) subject to constraints (2), (3), (4), (5) and (6). When the number of individual compressor units within a compressor station are identical and hooked-up in parallel, the linear system, as depicted in Fig. 2a, can be represented by a multigraph with the compressor stations aligned sequentially where the i-th compressor station (compressor arc $(i, i+1)$ in the figure) is modeled by a set of n_i arcs between suction node i and discharge pressure $i+1$ (see Fig. 4). Here, n_i is the number of individual compressor units, and each of the multi arcs $(i, i+1)$ represents a decision on how many units are used in that particular station. Each multi-arc in the i-th station is denoted by $(i, i+1, r)$ (or simply (i, r)), where r identifies the number of individual compressor stations to be used in a particular solution. Let L be the set of edges in this multigraph given by $L = \{(i, r) : i \in \{1, \ldots, n\}, r \in \{1, \ldots, n_i\}\}$. In this case, the cost of arc (i, r) given by c_{ir} depends on the values of the pressure variables p_i and p_{i+1}. This will be determined during the construction of the solution. Following equation (7), the cost is then given by

$$c_{ir} = rg^{(1)}(x_{i,i+1}/r, p_i, p_{i+1}).$$

where it can be seen in a straightforward manner that, in the case of linear systems with known supply/demand values, the flow variables $x_{i,i+1}$ through the entire network can be determined and fixed beforehand. Furthermore, this cost is heuristically estimated once at the start of the procedure.

At the start of the algorithm, m ants are placed at the starting node. Ants build a solution while moving from node to an adjacent node by choosing one of the multi-arcs and by randomly generating values of the pressure variables for correct computation of the arc cost. During iteration t, each ant k carries out a partial path $T^k(t)$, and in this step, the choice of arc (i, r) depends on both the cost c_{ir} and the concentration of pheromone $\tau_{ir}(t)$ on arc (i, r) at iteration t. The pheromone trail takes into account the ant's current history performance. This pheromone amount is intended to represent the learned desirability of choosing the r-th edge at node i. The pheromone trail information is changed during problem solution to reflect the experience acquired by ants during problem-solving.

First, the algorithm introduces a transition rule depending on parameter $q_0 \in [0, 1]$, which determines the relative importance of intensification/diversification trade-off: evert time an ant at node i chooses arc (i, r) according to the following transition rule:

$$r = \begin{cases} \arg\max_u (\tau_{iu}(t))^\alpha / (c_{iu})^\beta & \text{if } q \leq q_0, \\ s & \text{otherwise.} \end{cases}$$

where q is random variable uniformly distributed in [0, 1], and s is a random variable chosen according to the following probability function:

$$p_{is}^k(t) = \frac{(\tau_{is}(t))^\alpha / (c_{is})^\beta}{\sum_u (\tau_{iu}(t))^\alpha / (c_{iu})^\beta}$$

As can be seen, low values of q_0 lead to diversification and high values of q_0 stimulates intensification. Parameters α and β control the relative importance of the pheromone trail and greedy construction value. The main steps of the algorithm are shown in Procedure 2.

Algorithm 2: Pseudocode of Procedure ACO

```
1:  t ← 0
2:  while (stopping criteria not met) do
3:     t ← t + 1
4:     X^best ← ∅
5:     for (k = 1, ..., m) do
6:        Build solution X
7:        Apply local updating rule along path of X
8:        X^best ← Best(X, X^best)
9:     end for
10:    Apply global updating rule along path of X^best
11: end while
12: Return X^best
```

The pheromone trail is changed both locally (Step 7) and globally (Step 10) as follows.

- *Local updating*: every time arc (i, r) is chosen by an ant, the amount of pheromone changes by applying this local trail update:

$$\tau_{ir}(t) \leftarrow (1 - \rho)\tau_{ir}(t) + \rho\tau_0$$

  where τ_0 is the initial pheromone value and ρ the evaporation rate.

- *Global updating*: upon the completion of a solution by every ant in the colony, the global trail updating is done as follows. The best ant (solution) from this finished iteration is chosen according to the best objective function value g^*. Then, in each arc $(i, i+1, r)$ used by this best ant, the trail is updated as:

$$\tau_{ir}(t+1) \leftarrow (1-\rho)\tau_{ir}(t) + \frac{\rho}{g^*}$$

This algorithm was tested on the Hassi R'mell-Arzew real-world pipeline network in Argelia consisting of five pipes, six nodes, five compressor stations, and three units in each compressor. They also built three additional cases with up to 23 compressor stations, and 12 compressor units in each compressor. This method performs reasonably well on these type of networks according to the authors' empirical work. A great advantage is its relatively ease of implementation.

The issue on how this algorithm can be modified so as to handle noncyclic systems remains an interesting topic for further investigation along this area.

Metaheuristic Approaches to Related Problems

In this section, we review some other related optimization problems in natural gas pipeline networks that have been addressed by metaheuristic methods.

Particle Swarm Optimization for Non-isothermal Systems

Wu et al. [39] address a variation of the problem where, rather than minimizing fuel consumption, the focus is on maximizing a weighted combination of the maximum operation benefit and the maximum transmission amount. The operation benefit is defined as the sales income minus the costs. These costs include gas purchasing cost, pipeline's operation cost, management cost, and compressor's running cost. The transmission amount is defined as the total gas volume that flows into the pipeline. In addition, a non-isothermal model is considered; that is, the authors consider the dynamics of the pipes being a function of temperature. Most of the literature focus on the isothermal case. They develop a particle swarm optimization (PSO) metaheuristic enhanced by an adaptive inertia weight strategy to adjust the weight value dynamically. In a PSO implementation (see ▶ Chap. 21, "Particle Swarm Methods"), the inertia weight parameter is used to balance the global and local search ability. If the weight has a large value, the particle will search in a broader solution space. If the weight has a small value, the evolution process will focus on the space near to the local best particle. Thus, the global and local optimization performances of the algorithm can be controlled by dynamically adjusting the inertia weight value. This method adjusts the inertia weight adaptively based on the distance from the particles to the global best particle [33].

They tested their metaheuristic (named IAPSO) in the Sebeie-Ningxiae-Lanzhou gas transmission pipeline in China. Nine stations along the pipeline distribute gases to 16 consumers. There are four compressor stations with eight compressors to boost the gas pressure. The results show that IAPSO has fast convergence, obtaining reasonably good balances between the gas pipeline's operations benefit and its transportation amount.

Simulated Annealing for Time-Dependent Systems

As mentioned earlier, the previous two chapters addressed steady-state systems. However, when the steady-state assumption does not hold, the constraints that describe the physical behavior through a pipeline cannot be represented in the simplifying form as in (4). On the contrary, this behavior is governed by partial differential equations with respect to both flow and time. Therefore, to handle this situation, a discretization over the time variable must be done resulting in a highly complex optimization problem.

The resulting model is a mixed-integer nonlinear problem where now both flow variables and pressure variables are also a function of time; that is, we now have x_{ij}^t and p_i^t variables for every arc $(i, j) \in A$ and time step $t \in T$, where T is the set of time steps.

Although some efforts have been made to address transient systems, one of the most successful techniques for handling this problem is the simulated annealing (SA) algorithm of Mahlke et al. [18]. In that work, the authors use the following main ideas. First, they relax the equations describing the gas dynamic in pipes by adding these constraints combined with appropriate penalty factors to the objective function. The penalty factor is dynamically updated resembling a strategic oscillation strategy. This gives the search plenty of flexibility. Then, they develop a suitable neighborhood structure for the relaxed problem where time steps as well as pressure and flow of the gas are decoupled. Their key idea of the neighborhood generation is a small perturbation of flow and pressure variables in the segments and nodes, respectively. An appropriate cooling schedule, an important feature of each SA implementation, is developed. They tested their metaheuristic on data instances provided by the German gas company E.ON Ruhrgas AG. The proposed SA algorithm yields feasible solutions in very fast running times.

Conclusion

In this chapter we described successful metaheuristic implementations for handling difficult optimization problems in fuel cost minimization of natural gas transportation networks. Compared to existing approaches, metaheuristics have the great advantage of not depending on gradient-based information such that they can handle nonlinearities and nonconvexities with relative ease.

Nonetheless, metaheuristics have been widely applied mostly to discrete linear optimization problems and not to fully extent to handle the nasty problems within the natural gas industry. Therefore, there is a tremendous area of opportunity from the metaheuristic perspective in this very important field. One must keep in mind that these are real-world problems where even a marginal improvement in the objective function value represents a significant amount of savings given the total flow operation of these networks throughout the year. Therefore, further research in this area is justified and needed from the practical and scientific perspective.

Important research issues such as how to derive new metaheuristics or how the developed metaheuristics can be applied, extended, modified, so as to handle MFCPs under different assumptions (e.g., non-isothermal models, nonidentical compressor units, non-transient models, uncertainty) remain to be investigated. In these lines we have seen some preliminary efforts citing, for instance, the work of Mahlke et al. [18] who present a simulated annealing algorithm for addressing a MFCP under transient conditions. However, further work is needed. We know that advanced concepts in metaheuristic optimization research, such as reactivity, adaptive memory, intensification/diversification strategies, or strategic oscillation, are worthwhile investigating. Furthermore, as we have seen in this chapter, these models have a rich mathematical structure that allow for hybridization where part of the problem can be solved with mathematical programming techniques while being guided within a metaheuristic framework.

We hope we can stimulate the interest of the scientific community, particularly from metaheuristic optimization field, to contribute to advance the state of the art in this very challenging research area.

Cross-References

- ▸ Ant Colony Optimization: A Component-Wise Overview
- ▸ Matheuristics
- ▸ Particle Swarm Methods
- ▸ Tabu Search

Acknowledgments The research of the author was supported by the Mexican Council for Science and Technology, grant CONACyT CB-2011-01-166397. The author would also like to thank the editors for their helpful remarks and suggestions.

References

1. Aalto H (2008) Optimal control of natural gas pipeline networks: a real-time, model-based, receding horizon optimisation approach. VDM Verlag, Saarbrücken
2. Anglard P, David P (1988) Hierarchical steady state optimization of very large gas pipelines. In: Proceedings of the 20th PSIG annual meeting, Toronto
3. Borraz-Sánchez C, Haugland D (2011) Minimizing fuel cost in gas transmission networks by dynamic programming and adaptive discretization. Comput Ind Eng 61(2):364–372

4. Borraz-Sánchez C, Ríos-Mercado RZ (2005) A hybrid meta-heuristic approach for natural gas pipeline network optimization. In: Blesa MJ, Blum C, Roli A, Sampels M (eds) Hybrid metaheuristics. Springer, Berlin, pp 54–65
5. Borraz-Sánchez C, Ríos-Mercado RZ (2009) Improving the operation of pipeline systems on cyclic structures by tabu search. Comput Chem Eng 33(1):58–64
6. Carter RG (1998) Pipeline optimization: dynamic programming after 30 years. In: Proceedings of the 30th PSIG annual meeting, Denver
7. Chebouba A, Yalaoui F, Smati A, Amodeo L, Younsi K, Tairi A (2009) Optimization of natural gas pipeline transportation using ant colony optimization. Comput Oper Res 36(6):1916–1923
8. Cobos-Zaleta D, Ríos-Mercado RZ (2002) A MINLP model for minimizing fuel consumption on natural gas pipeline networks. In: Proceedings of the XI Latin-Ibero-American conference on operations research, Concepción
9. De Wolf D, Smeers Y (2000) The gas transmission problem solved by an extension of the simplex algorithm. Manag Sci 46(11):1454–1465
10. Domschke P, Geißler B, Kolb O, Lang J, Martin A, Morsi A (2011) Combination of nonlinear and linear optimization of transient gas networks. INFORMS J Comput 23(4):605–617
11. Ehrhardt K, Steinbach MC (2005) Nonlinear optimization in gas networks. In: Bock HG, Kostina E, Phu HX, Rannacher R (eds) Modeling, simulation and optimization of complex processes. Springer, Berlin, pp 139–148
12. Flores-Villarreal HJ, Ríos-Mercado RZ (2003) Computational experience with a GRG method for minimizing fuel consumption on cyclic natural gas networks. In: Mastorakis NE, Stathopulos IA, Manikopoulos C, Antoniou GE, Mladenov VM, Gonos IF (eds) Computational methods in circuits and systems applications. WSEAS Press, Athens, pp 90–94
13. Jin L, Wojtanowocz AK (2010) Optimization of large gas pipeline network – a case study in China. J Can Pet Technol 49(4):36–43
14. Ke SL, Ti HC (2000) Transient analysis of isothermal gas flow in pipeline network. Chem Eng J 76(2):169–177
15. Lall HS, Percell PB (1990) A dynamic programming based gas pipeline optimizer. In: Bensoussan A, Lions JL (eds) Analysis and optimization of systems. Lecture notes in control and information sciences, vol 144. Springer, Berlin, pp 123–132
16. Larson RE, Wismer DA (1971) Hierarchical control of transient flow in natural gas pipeline networks. In: Proceedings of the IFAC symposium on distributed parameter systems, Banff,
17. Luongo CA, Gilmour BJ, Schroeder DW (1989) Optimization in natural gas transmission networks: a tool to improve operational efficiency. Technical report. Stoner Associates, Inc., Houston
18. Mahlke D, Martin A, Moritz S (2007) A simulated annealing algorithm for transient optimization in gas networks. Math Methods Oper Res 66(1):99–115
19. Mantri VB, Preston LB, Pringle CS (1985) Transient optimization of a natural gas pipeline system. In: Proceedings of the 17th PSIG annual meeting, Albuquerque
20. Martin A, Möller M, Moritz S (2006) Mixed integer models for the stationary case of gas network optimization. Math Program 105(2–3):563–582
21. Osiadacz AJ (1994) Dynamic optimization of high pressure gas networks using hierarchical systems theory. In: Proceedings of the 26th PSIG annual meeting, San Diego
22. Osiadacz AJ (1998) Hierarchical control of transient flow in natural gas pipeline systems. Int Trans Oper Res 5(4):285–302
23. Osiadacz AJ, Bell DJ (1986) A simplified algorithm for optimization of large-scale gas networks. Optim Control Appl Methods 7(1):95–104
24. Osiadacz AJ, Chaczykowski M (2001) Comparison of isothermal and non-isothermal pipeline gas flow models. Chem Eng J 81(1):41–51
25. Osiadacz AJ, Swierczewski S (1994) Optimal control of gas transportation systems. In: Proceedings of the 3rd IEEE conference on control applications, Glasgow, pp 795–796. ISBN:0-7803-1872-2
26. Percell PB, Ryan MJ (1987) Steady-state optimization of gas pipeline network operation. In: Proceedings of the 19th PSIG annual meeting, Tulsa

27. Pratt KF, Wilson JG (1984) Optimisation of the operation of gas transmission systems. Trans Inst Meas Control 6(5):261–269
28. Ríos-Mercado RZ (2002) Natural gas pipeline optimization. In: Pardalos PM, Resende MGC (eds) Handbook of applied optimization, chap 18.8.3. Oxford University Press, New York, pp 813–825
29. Ríos-Mercado RZ, Borraz-Sánchez C (2015) Optimization problems in natural gas transportation systems: a state-of-the-art review. Appl Energy 147:536–555
30. Ríos-Mercado RZ, Wu S, Scott LR, Boyd EA (2002) A reduction technique for natural gas transmission network optimization problems. Ann Oper Res 117(1–4):217–234
31. Ríos-Mercado RZ, Kim S, Boyd EA (2006) Efficient operation of natural gas transmission systems: a network-based heuristic for cyclic structures. Comput Oper Res 33(8):2323–2351
32. Schmidt M, Steinbach MC, Willert BM (2015) High detail stationary optimization models for gas networks. Optim Eng 16(1):131–164
33. Suresh K, Ghosh S, Kundu D, Sen A, Das S, Abraham A (2008) Inertia-adaptive particle swarm optimizer for improved global search. In: Proceedings of the eighth international conference on intelligent systems design and applications. IEEE Computer Society, Los Alamitos, pp 253–258
34. Tabkhi F, Pibouleau L, Hernandez-Rodriguez G, Azzaro-Pantel C, Domenech S (2010) Improving the performance of natural gas pipeline networks fuel consumption minimization problems. AIChE J 56(4):946–964
35. Tao WQ, Ti HC (1998) Transient analysis of gas pipeline network. Chem Eng J 69(1):47–52
36. Wong PJ, Larson RE (1968a) Optimization of natural-gas pipeline systems via dynamic programming. IEEE Trans Autom Control AC–13(5):475–481
37. Wong PJ, Larson RE (1968b) Optimization of tree-structured natural-gas transmission networks. J Math Anal Appl 24(3):613–626
38. Wu S, Ríos-Mercado RZ, Boyd EA, Scott LR (2000) Model relaxations for the fuel cost minimization of steady-state gas pipeline networks. Math Comput Model 31(2–3):197–220
39. Wu X, Li C, Jia W, He Y (2014) Optimal operation of trunk natural gas pipelines via an inertia-adaptive particle swarm optimization algorithm. J Nat Gas Sci Eng 21:10–18
40. Zheng QP, Rebennack S, Iliadis NA, Pardalos PM (2010) Optimization models in the natural gas industry. In: Rebennack S, Pardalos PM, Pereira MVF, Iliadis NA (eds) Handbook of power systems I, energy systems. Springer, Berlin, pp 121–148

Network Optimization

38

Luciana S. Buriol

Contents

Introduction 1124
Preliminaries 1125
Basic Network Problems 1125
Point-to-Point Shortest Path Problem (PPSPP) 1125
Maximum Flow Problem (MFP) 1126
Minimum Cost Flow Problem (MCFP) 1127
Multicommodity Flow Problems 1128
Weight Setting Problem (WSP) 1129
The WSP Applied to Telecommunication Networks 1130
Heuristic Approaches for the WSP 1131
Virtual Network Embedded Problem (VNEP) 1134
Single VNEP Variant 1135
Heuristic Approaches for the VNEP 1136
Conclusions 1138
Cross-References 1139
References 1139

Abstract

Several real-world problems can be modeled as graph problems. Graph algorithms and theories that have evolved for decades can be applied to solve the problem on hand. Interestingly, many of these graph problems can be solved polynomially, while small changes in a problem definition turn the problem difficult. In this chapter, we explore this path from polynomial network problems to NP-hard ones. Along the chapter, we visit several problems, dedicating more extended discussions to two real-world problems: the

L. S. Buriol (✉)
Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
e-mail: buriol@inf.ufrgs.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_46

weight setting problem, originated from telecommunication networks, and the virtual network embedded problem, a recent stated optimization problem from the computer network area. For these two problems, we discuss their heuristic solution, since only small instances can be solved exactly within a reasonable amount of time.

Keywords
Network optimization · Traffic engineering · Network virtualization

Introduction

Many problems can be modeled as graphs. Once a problem can be modeled as a graph, all graph algorithms and theories that have evolved for decades can be applied to analyze and solve the problem at hand. It is the case, for example, of social networks or webgraphs. In webgraphs, nodes represent webpages and edges are hyperlinks, while in social networks nodes usually represent people, and arcs represent interaction among them. In these two types of networks basically, the graph topology is analyzed, which usually configures complex and massive networks. Google and Facebook base some of their algorithms on these graphs. On the other hand, in biological networks, nodes can be represented by genes, and the correlation between each pair of nodes is represented through edge weights in a complete graph. By analyzing these graphs, scientists discover genes related to diseases and can then propose new drugs and observe how they affect the edge weights of the graph. Health schools and computer science groups have made several joint efforts in this direction.

Another class consists of graphs originating from services, such as telecommunication and road networks. These service graphs naturally are represented as a set of elements that includes a topology, but also additional information about nodes and arcs such as capacity, distance, demand, and delay are considered in computations over the network. In this chapter, we focus on problems over service networks. We will also discuss strategies applied to network optimization problems. Initially, we introduce related polynomial problems, and then we discuss constraints that turn the problems difficult to solve. Among the hard problems, we put some emphasis on two NP-hard problems: the weight setting problem (WSP) and virtual network embedded problem (VNEP). WSP has its main applications in traffic engineering problems from transit and telecommunications, while VNEP is a computer network problem.

We present most of the problems by their definition and also by their mathematical formulation. We aim to make the reader familiar with the mathematical problem modeling since optimization can often exploit this feature when solving a problem. Moreover, by analyzing their mathematical formulations, we can also highlight how characteristics of the problem change its modeling and affect its algorithms and solution space.

Preliminaries

Graphs naturally model network problems. Let $G = (V, A)$ denote a directed graph (DAG), where V is the set of nodes and A is the set of arcs. In a directed graph, the set of outgoing arcs from u is represented by $N^+(u)$, and similarly $N^-(u)$ represents the set of incoming arcs to node u. We denote the size of these sets by $|V| = n$ and $|A| = m$. An arc (u, v) has u as the tail node and v as the head node. Throughout this chapter we consider problems in simple graphs, that is, graphs without self-loops or parallel edges. However, directed graphs can admit symmetric arcs (u, v) and (v, u). A path in a graph is a sequence of arcs that connect pairs of neighbor nodes. The path is simple if there are no loops. Along the text, link and arc are used as synonymous.

Basic Network Problems

Initially, we introduce three representative polynomial time network problems that are found independently or as part of the core of more complex optimization problems. From several problem candidates, the point-to-point shortest path problem, maximum flow problem, and minimum cost flow problem were chosen to be discussed. These are fundamental network problems that have several variants that embrace a large number of practical applications.

Point-to-Point Shortest Path Problem (PPSPP)

Given a directed graph $G = (V, A)$ with weights w associated to arcs $w : A \to \mathcal{R}^+$, and source s and target t nodes, the PPSP problem consists in finding the path of minimum cost between nodes s and t. If a variable $x_a \in \{0, 1\}$ is associated to each $a \in A$, representing whether arc a is part of the solution, the point-to-point shortest path problem can be mathematically formulated as:

$$\textbf{min} \quad \sum_{a \in A} w_a x_a \tag{1}$$

$$\textbf{s.a} \quad \sum_{a \in N^+(v)} x_a - \sum_{a \in N^-(v)} x_a = \begin{cases} 1, \text{for } v = s, \\ 0, \text{for } v \in V \setminus \{s, t\}, \\ -1, \text{for } v = t. \end{cases} \tag{2}$$

$$x_a \in \{0, 1\} \text{ for } a \in A.$$

The objective is to minimize the cost path (Eq. 1). The path constraints (2) guarantee to find a valid path between nodes s and t. If $w : A \to \mathcal{R}^{+*}$, i.e., w has only positive values, the path found is always simple, otherwise, the shortest

path can have cycles of zero cost. The weight function w associated to arcs can represent time, cost, distance, or other measures, depending on the application. A typical application of this problem is to find a route in a road network of a navigation system, as a GPS, where w represents distance or time or the result of a cost function involving these two measures.

This problem can be solved exactly in time $O(m \log n)$ using binary heaps by the Dijkstra's algorithm. If additional information is available (or possible to compute), such as estimated distances (it cannot be an overestimation) between each pair of nodes, an admissible heuristic function can be used as part of the evaluation function of the A^* algorithm, and its time performance can be sped up. Also, in some situations, heuristics can be used such that a sub-optimal solution is returned. It is the case of *anytime algorithms* [31], which a feasible solution is requested in a short time, and as more time is available, the solution can be improved to a given ϵ distance error or up to optimality. Another case where the algorithm performance can be sped up is when a large number of queries of a path between a pair of nodes are requested, considering a static input [23, 29, 30]. In this case, information can be pre-computed and stored to be used in multiple queries. Finally, if one or more arc weight values of the input graph change, a previously calculated shortest path can be updated, instead of recomputed [36] from scratch. This approach is efficient when the application requires successive changes and updates of the graph.

Next, we discuss two flow problems. We focus on the integer version of the problems, i.e., the flow traversing each arc is always an integer value. For an extensive study in network flow problems, we refer to [1] which addresses theoretical and practical aspects of several problems.

Maximum Flow Problem (MFP)

The maximum flow problem (MFP) is a fundamental problem in network optimization. Given a directed graph $G = (V, A)$ with arc capacities $l : A \to \mathcal{R}$ associated with arcs, and source s and target t nodes, the problem consists in sending the maximum amount of flow between s and t without surpassing the arc capacities. If variables $x_a \in \mathcal{N}$ represent the amount of load on each $a \in A$, where $d_s = F$, $d_t = -F$, and $d_u = 0$ for $u \in V \setminus \{s, t\}$, and F stands for the maximal amount of flow traversing the network, this problem can be mathematically formulated as:

$$\textbf{max} \quad F = \sum_{a \in N^+(s)} x_a \tag{3}$$

$$\textbf{s.a} \quad \sum_{a \in N^+(v)} x_a - \sum_{a \in N^-(v)} x_a = d_v, \text{ for } v \in V, \tag{4}$$

$$x_a \leq l_a, \text{ for } a \in A, \tag{5}$$

$$x_a \in \mathcal{N}, \text{ for } a \in A. \tag{6}$$

The objective function (3) maximizes the flow outgoing node s and traversing the network directed to node t, while constraints (5) assure capacities are not surpassed. Constraints (4) guarantee flow conservation: the difference between the total flow emanating from v and entering node v is equal to its supply/demand.

The MFP problem can be solved polynomially. In 1956 the pseudo-polynomial algorithm of Ford-Fulkerson was proposed. It has a time complexity of $O(m^2 \log F)$, where F is the maximum flow traversing the network. In 1972 a specialization of the previous algorithm was proposed which led to the Edmonds-Karp algorithm, whose time complexity is $O(nm^2)$, the first polynomial algorithm for solving the problem. A few other polynomial algorithms were proposed over the years, with a remark to the $O(n^3)$ algorithm of Goldberg and Tarjan [22] (fastest in practice) and the recently proposed $O(mn)$ algorithm of Orlin [34].

Minimum Cost Flow Problem (MCFP)

If besides capacities we have also costs $w : A \to \mathcal{R}$ associated with arcs, each node u can supply or demand a flow $d_u \in \mathcal{N}$, we can then define the minimum cost flow problem. The problem consists in transporting the flow through the network, respecting capacity constraints, and minimizing the cost function. With $x_a \in \mathcal{N}$ being the flow amount on arc a, the objective function of the problem is

$$\textbf{min} \quad \sum_{a \in A} w_a x_a$$

while the constrains are the same (4), (5), and (6) from the MFP, with d_u positive for supply nodes and set to negative for demand nodes. Moreover, the sum of all supplies should be equal to the sum of all demands, otherwise, the problem has no solution. This problem and several variants can be solved by the algorithms for the MFC, with a simple modification of the input graph [28, Section 7.7].

The arc capacities naturally define upper bounds on the flow on arcs. A common variant of flow problems is to consider also a lower bound on arc capacities, forcing that a given minimum amount of flow circulates on certain arcs. If lower bounds are considered on the flow problems above, then the problem is still polynomial [28, Section 7.7].

Consider the set S of source nodes and a set T of target nodes. In case $S \cap T = \emptyset$, both MFP and MCFP remain polynomial due to the Hoffman-Kruskal theorem on totally unimodular matrices [25]. However, if $S \cap T \neq \emptyset$, these problems are not polynomial anymore, and we discuss some variants in the next subsection.

Multicommodity Flow Problems

The MCFP and MFP route a commodity per time. In multicommodity flow problems, not one, but several commodities are routed simultaneously on the network.

Let $K = \{(o(1), d(1)), (o(2), d(2)), \ldots, (o(|K|), t(|K|)\} \subseteq V \times V$ denote the set of commodities or origin-destination (OD) pairs, where $o(k)$ and $t(k)$ represent, respectively, the origination and destination nodes for $k = 1, \ldots, |K|$. Each commodity k has an associated demand of traffic flow $d_k = d_{o(k),t(k)}$, i.e., for each OD pair $(o(k), t(k))$, there is an associated flow d_k that emanates from node $o(k)$ and terminates in node $t(k)$. The total set of commodities defines a demand matrix whose each entry $[o(k)][t(k)]$ specifies the demand $d_k = d_{o(k),t(k)}$.

The MFP and MCFP problems can be easily extended to a multicommodity flow version. Given an arbitrary OD matrix, the multicommodity MCFP can be formulated as:

$$\textbf{min} \quad \sum_{a \in A} w_a \sum_{k \in K} x_a^k \tag{7}$$

$$\textbf{s.a} \quad \sum_{k \in K} \sum_{a \in N^+(v)} x_a^k - \sum_{k \in K} \sum_{a \in N^-(v)} x_a^k = d_v^k, \text{ for } v \in V \tag{8}$$

$$\sum_{k \in K} x_a^k \leq l_a, \text{ for } a \in A, \tag{9}$$

$$x_a \in \mathcal{N}, \text{ for } a \in A.$$

The time complexity of different variants of the problem is classified as polynomial or not, depending on the problem definition. For both, direct and undirect graphs, the decision version of the multicommodity integer flow problem is NP-complete even if the number of commodities is two [17].

Heuristics are largely applied as the solution approach for NP-hard multicommodity network optimization problems. The reasons are diverse. There are cases solved by mathematical programming approaches when the instances are small. However, for most cases, an exact approach does not solve medium instances of the problem even if a long time is available for processing. It is common to happen that the model cannot even be uploaded by the solver due to its size. There are also situations in which the problem cannot be mathematically modeled by linear or nonlinear constraints, as, for example, some electrical network characteristics such as electrical impedance. More often the problem can be modeled, but it results in a nontrivial nonlinear problem model that cannot be efficiently solved in a reasonable amount of time. Finally, some problems take as input data sets composed of multiple data that are in many cases estimates or average values. An exact solution, in this case, many times does not represent the best solution for the problem. For all these

applications, heuristics are largely applied, and in most cases, they are the only possible practical resolution approach for the problem.

Even when a problem is solved heuristically, modeling the problem and solving it exactly is always important to find out which size an exact approach can be used in practice and if there are subsets of instances that are easier to solve exactly.

The next two subsections detail two NP-hard problems that are found in service networks and which are solved through metaheuristics. Their solution approaches are also presented and discussed.

Weight Setting Problem (WSP)

The weight setting problem (WSP) is defined in a directed weighted capacitated network G = (V,A), with $w : A \rightarrow \mathcal{N}$ and $l : A \rightarrow \mathcal{R}^+$. The problem consists in attributing a weight on each arc of the network such that when each flow from a demand matrix is sent along the shortest paths, a cost function $\phi(l_a)$, which depends on the arc flow l_a, is minimized. Usually, ϕ is a congestion cost function, and it can be defined in different ways according to specific goals.

The WSP has its primary application in telecommunication networks. The Internet is divided into autonomous systems (AS). Each AS controls its interior routing by an interior gateway protocol. Common interior gateway protocols, for instance, open shortest path first (OSPF), allow the operator to define the routes by setting integer weights on the network links. For example, in the main application described in this chapter, weights are set in the range $[1 : 20]$, and ϕ minimizes the network congestion.

Another application of the WSP is in road networks. A road network can be represented as a directed graph $G = (V, A)$ where V represents the set of nodes (street or road intersections or points of interest) and A the set of arcs (street or road segments). Each arc $a \in A$ has an associated capacity c_a, and a time t_a, called the *free flow time*, that represents the time spent when traversing the unloaded arc a. To calculate the congestion on each link, a potential function Φ_a is computed as a function of the load or flow ℓ_a on arc a, along with other arc-tuning parameters. The flow in an arc is computed taking into account that a driver takes the shortest path. The cost of a path is calculated as the sum of tolls allocated in the path or as a function of the time and tolls. The tollbooth problem [38] consists in attributing toll values on r arcs of a road network, such that when demand is sent along the shortest path between origin and destination, the network congestion is minimized. The weight w_a that the WSP problem defines on an arc a represents in the tollbooth problem the toll charged for each unit of flow traversing arc a. Note that in this case, at most r weights are allocated.

We detail in the rest of this section the WSP applied to telecommunication networks, revising the current approaches being used to solve the problem. We initiate this discussion describing the problem formally.

The WSP Applied to Telecommunication Networks

WSP receives as input a capacitated directed graph $G = (V, A)$ with a capacity of $l_a \in \mathcal{R}^+$ for each arc $a \in A$. Besides that, an OD demand matrix, with $d_k \in \mathrm{R}\ \forall k \in K$, where K is the set of commodities or OD pairs, is also given as input. The *OSPF weight setting problem* consists in determining weights to be assigned to the links so as to optimize a cost function, typically associated with a network congestion measure. Thus, the problem defines an integer weight $w_a \in [1 : 2^{16}]$ (most of the studies in the literature use $w_a \in [1 : 20]$) $\forall a \in A$. The flow is routed along the shortest paths between $o(k)$ and $t(k)$ for each OD pair. The shortest paths are calculated according to the link weights w_a defined by the problem. If there is more than one shortest path to an OD pair, the flow is evenly split at each node among all outgoing arcs that belongs to the shortest path graph to the flow destination. The cost function takes into account the amount of load on each arc of the network. In [20] a piece-wise linear function $\Phi = \sum_{a \in A} \phi_{l_a}$ is used to calculate the cost function Φ. However, depending on the application, one can consider simpler cost functions such as the ratio between the load and the capacity on the arc $\frac{l_a}{c_a}$. A ratio on an arc larger than one means the arc is overloaded.

A mathematical model for the weight setting problem is given in equations (10)–(20). The decision variables for this model determine the weight value of an arc a. Denote by $w_a \in \{W_{\min}, W_{\min} + 1, \ldots, W_{\max}\}$ as the weight value attributed to arc $a \in A$, where $W_{\min} \in \mathcal{N}$ and $W_{\max} \in \mathcal{N}$ are the minimum and maximum values of weights, respectively. For convenience we specify $W_{\min} = 1$. Let Q be the set of all nodes that are a destination of at least one OD pair. The auxiliary binary variable y_a^q is set to one ($y_a^q = 1$) if arc $a \in A$ is part of a shortest path to destination node $q \in \mathcal{Q}$. Finally, auxiliary variable δ_v^q is the shortest path distance from node $v \in V$ to destination node $q \in \mathcal{Q}$, and the constants M_1, M_2 and M_3 are sufficiently larger numbers. Below the problem tackled in [8] is formulated inspired in constraints presented in [7].

$$\min \ \Phi = \sum_{a \in A} \phi(l_a) \tag{10}$$

Subject to:

$$\ell_a = \sum_{q \in \mathcal{Q}} x_a^q, \ \forall a \in A, \tag{11}$$

$$\sum_{a \in N_v^+} x_a^q - \sum_{a \in N_v^-} x_a^q = d_{v,q}, \ \forall v \in V \backslash \{q\}, \ \forall q \in \mathcal{Q}, \tag{12}$$

$$w_a + \delta_{a_h}^q - \delta_{a_t}^q \geq 0, \ \forall a \in A, \ \forall q \in \mathcal{Q}\,, \tag{13}$$

$$\delta_q^q = 0, \ \forall q \in \mathcal{Q}, \tag{14}$$

$$w_a + \delta_{a_h}^q - \delta_{a_t}^q \geq (1 - y_a^q)/M_1, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{15}$$

$$w_a + \delta_{a_h}^q - \delta_{a_t}^q \leq (1 - y_a^q)M_2, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{16}$$

$$y_a^q \geq x_a^q, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{17}$$

$$M_3 y_a^q + M_3 y_b^q \leq 2M_3 - x_a^q + x_b^q, \ \forall a, b \in \mathsf{A}^2_{N_v^+}, \forall v \in V, \ \forall q \in \mathcal{Q}, \tag{18}$$

$$W_{\min} \leq w_a \leq W_{\max}, \ \forall a \in A, \tag{19}$$

$$x_a^q \in \mathcal{N}, y_a^q \in \{0, 1\}, w_a \geq 0, \delta_v^q \geq 0 \, ; \forall a \in A, \forall v \in V, \forall q \in \mathcal{Q}, \tag{20}$$

Objective function (10) minimizes average user travel time. Constraints (11) define the total flow on each arc $a \in A$, while constraints (12) impose flow conservation. The input data d_{uv} is the flow originated in u with destination in v. The following constraints impose the flow of each commodity to follow the shortest path between the corresponding OD pair. An arc a belongs to the shortest path to destination q if the distance $\delta_{a_h}^q - \delta_{a_t}^q$ is equal to the arc cost, which in this case is w_a. Thus, the constraints (13) define the shortest path distance for each node $v \in V$ and each destination $q \in \mathcal{Q}$. For consistency, constraints (14) require, for all $q \in \mathcal{Q}$, that the shortest distance from q to itself be zero. Constraints (15) and (16) together with (13) and (14) determine whether arc $a \in A$ belongs to the shortest path and thus determine the values of y_a^q, for $q \in \mathcal{Q}$. Constraints (15) require that an arc that does not belong to the shortest path has reduced cost $w_a + \delta_{a_h}^q - \delta_{a_t}^q > 0$. It also prevents flow from traversing zero-cost cycles. Constraints (16) assure that if the reduced cost of arc $a \in A$ and destination $q \in \mathcal{Q}$ is equal to zero, then arc a belongs to the shortest path to destination q, i.e., $y_a^q = 1$. Constraints (17) assure that flow is sent only on arcs belonging to a shortest path. Constraints (18) guarantee that flow is split evenly among all shortest paths (even-split constraints). In these constraints, $\mathsf{A}^2_{N_v^+}$ is the set of all ordered groups of two distinct elements of N_v^+. In [38] this constraint is also used and presented in detail. Constraints (19) limit the minimum and the maximum weight value. The last constraints define the domains of the variables.

Heuristic Approaches for the WSP

Fortz and Thorup showed that the weight setting problem for OSPF is NP-hard [20]. Several authors have proposed further heuristic solutions for solving variants of the problem applied in different contexts. Table 1 summarizes a literature review on solution approaches for solving the WSP. The first column of the table refers to the problem application. The problem is found in telecommunication and road networks. We mean by network operation the optimization on the use of available resources, while network design is related to the allocation of resources to attend minimal conditions for the network operation. Column $|V|/|A|/|OD|$ presents the measures (number of nodes, arcs, and OD pairs) of the largest network considered in the computational results for each reference indicated in the third column of the table. Finally, the last column presents the solution approach of each work

Table 1 Solution approaches for solving the WSP applied in different contexts

Application	$\|V\|$ / $\|A\|$ / $\|OD\|$	Reference	Solution approach
Telecommunications operation	100/403 /–	[20, 21]	TS
Telecommunications operation	100/503/9,900	[16]	GA
Telecommunications operation	56/224/3,080	[35]	SA
Telecommunications operation	100/503/9,900	[9]	MA
Telecommunications design	30/148/–	[7]	MultiObj
Telecommunications design	71/350/4,960	[10]	MA
Telecommunication operation	50/88/662	[5]	MIP
Road network operation	974/2,153/9,505	[38]	BRKGA

indicated in the previous column. The methods used are tabu search (TS), genetic algorithms (GAs), simulated annealing (SA), memetic algorithm (MA), multi-objective optimization (MultiObj), heuristic based on mathematical programming (MIP), and biased random-key genetic algorithms (BRKGAs).

In some of the works indicated in the third column of the table, the authors also presented results from optimal approaches, for comparison purposes. For example, in [38], which applies WSP to road networks, the largest network that CPLEX was able to solve optimally in less than 30 minutes has dimensions 9/26/68. If the instance sizes double, in about 50% of the runs, no feasible solution is found in 30 minutes. However, the BRKGA proposed in the paper presents results for real-world instances two orders of magnitude larger, within the same 30 minutes.

Construction Heuristics for the WSP

For all telecommunication operation problems from Table 1, a solution can be simply represented as a vector w of m weights, one for each arc. Once the weight vector is generated, the solution cost can be calculated according to Fig. 1.

In order to update the new arc loads, the shortest paths are computed to all destination nodes $t \in T$ and arrive at a graph $g^t = (V, A^t)$, $\forall\ t \in T$ (Line 2). This is achieved using Dijkstra's well-known shortest path algorithm. Since shortest paths are computed to all destination nodes, the directions of all arcs are reversed in G, and the distances π_u^t, $\forall\ u \in V$ to a destination in T are computed. Given the shortest paths to each destination, auxiliar vectors g^t (Line 3) and δ^t (Line 4) are computed considering each destination t. Each position $g_a^t \in \{0, 1\}$ indicates if arc a belongs to at least one shortest path with destination in t. On the nodes side, each position δ_v^t stores the number of outgoing arcs of node v that belongs to at least one shortest path with destination in t. The flows L^t (Line 5) from all OD demand pairs with destination t are then routed. Finally, the total flows l_a $\forall a \in A$ (Line 7). The cost of a solution is computed according to an evaluation function.

Any solution composed of integer weights generated within the range $[1{:}2^{16}]$ is a feasible solution in all telecommunication operation studies from Table 1. In fact, the interval [1:20] is adopted by most of the methods, since values are still within the range accepted by OSPF protocol and it was observed that with a larger

```
procedure EvaluateSolution(w)
1   forall t ∈ T do
2       π^t ← ReverseDijkstra(w);
3       g^t ← ComputeSPG(w, π^t);
4       δ^t ← ComputeDelta(g^t);
5       L^t ← ComputePartialLoads(μ, δ, π, g^t);
6   end forall
7   l ← ComputeTotalLoads(L);
8   Φ ← Σ_{a∈A} φ_a;
9   return(Φ);
end procedure
```

Fig. 1 Pseudo-code for a WSP solution evaluation procedure

range the results do not improve [16, 20]. If one or more arcs have their capacities surpassed, the solution evaluation will reflect this in a high objective function cost. In practice, in telecommunication networks a link overload means retransmission. In road networks, a link overload means high traffic. Thus, the problem allows the overload imposing a high cost in the objective function.

Even if any random solution is a feasible solution, it is hard to design a high-quality initial solution for this problem. Thus, most of the approaches use initial random solutions. However, two solution approaches were adopted in [10] that usually provides good initial solutions. One defines weights for each arc a proportional to the inverse of the arc capacity $1/c_a$. A second approach solves a relaxation of the problem considering the model with constraints (13), (14), (15), (16), (17), and (18), i.e., constraints that force the flow to take the shortest paths are not considered. The relaxation returns a set of fractional weights, which are rounded to the closest integer.

Local Search Heuristics for the WSP

The neighborhood used by most of the methods changes the weight of a single arc per time. In [20] an arc is chosen at random, and its weight is changed by a value selected at random in the interval [1:20]. Every time a weight value of an arc changes,

the solution is reevaluated. To avoid recomputing solutions that were already evaluated, a hash function is used to detect when a solution was previously evaluated.

In [9] the local search also changes one arc weight per time but in a different fashion. Only the arcs with higher traffic have their weight changed. The initial step of the local search is to evaluate the current solution, given its weight vector. Then, the arcs with the higher traffic calculated according to ϕ_a have their weights changed, one arc at a time. However, the weight can only increase, one unit per time, while the solution is improving, or up to a specific weight defined by the algorithm. Instead of recomputing the whole shortest path graph from scratch, the graph and flow are only updated, to reflect the change. Moreover, unit changes are faster to compute than random changes [11]. Updating the graph is about 15 times faster than recomputing from scratch, as reported by the authors.

Virtual Network Embedded Problem (VNEP)

In the last few years, the amount of data routed through telecommunication networks has increased considerably. To attend the increasing demand, while maintaining a high quality of service for low prices, new enabling technologies are being deployed. One of them employs virtualizing environments which allow for multiple networks to simultaneously and transparently share the same physical structure. Likewise, service providers can offer customized services or co-location for expanded network presence [19]. Virtualization is a growing trend in the implementation of cloud computing architectures.

The virtual network embedding problem (VNEP) is central to achieve network virtualization. It consists of mapping virtual networks with heterogeneous architectures on physical infrastructures. Virtual nodes are mapped into physical substrate nodes, and virtual links are mapped into paths in the physical substrate network. Additionally, physical resources are finite and must be used judiciously: nodes have a processing capacity, and links have a bandwidth limit. The main objective of the VNEP is to minimize the use of the underlying resources such as bandwidth while respecting the mapping constraints. VNEP is a multicommodity flow problem since each virtual link corresponds to an OD pair since it consumes capacity of the substrate networks.

Several variants of VNEP have been proposed in the literature. For example, one virtual network can be mapped at a time [13], or several virtual networks are mapped simultaneously [24, 26]. Physical nodes and links can be restricted to host a limited number of virtual nodes and links, or specific virtual links can have a location restriction. Some applications have security restrictions which limit the subset of physical nodes and links that can be used for mapping [3]. Additional constraints can also be present such as delay [27], efficient use of energy [6], and redundancy [37]. Some of the VNEP variants allow path splitting. Others do not. When path splitting is enabled, virtual links are mapped to multiple paths in the substrate network.

The VNEP was shown to be NP-hard by a reduction from the bin packing problem [33]. Due to the difficulty of solving the VNEP, few exact algorithms have been proposed in the literature. For most practical purposes, heuristic algorithms can provide good sub-optimal solutions for the problem.

Single VNEP Variant

This section describes the VNEP variant that maps a single virtual network into the substrate. Thus, a VNEP instance has as input a virtual network and a physical substrate network. The physical substrate is represented by an undirected graph $G^S = (V^S, A^S)$ with a CPU capacity of C_s for each physical node $s \in V^S$ and a bandwidth capacity B_a for each arc $a \in A^S$. The virtual network is represented by an undirected graph $G^V = (V^V, A^V)$ along with a demand C_v for each virtual node $v \in V^V$ and a bandwidth demand of B_k for each virtual link $k \in A^V$.

The objective of the proposed problem is to find a feasible mapping of the virtual nodes and links onto the physical network with minimal cost. A feasible mapping is a pair of functions (f_v, f_e): a mapping of nodes $f_v : V^V \rightarrow V^S$ and a mapping of links $f_{e=(w,u)} : A^V \rightarrow P$, where P is the set of paths in the substrate graph with endpoints $f_v(w)$ and $f_v(u)$. Each virtual node has to be mapped into a single substrate node with enough CPU capacity to host it. A substrate node can host at most one virtual node. Each virtual link $(w, u) \in A^V$ has to be mapped to a path in the physical graph between the nodes $f_v(w)$ and $f_v(u)$. An arc can host several virtual links, but the sum of their demands should not surpass the capacity of the arc. The cost of a mapping is the amount of bandwidth used in the physical network by the mapping.

Figure 2 presents an instance of the problem composed of a physical network with four nodes (left) and a virtual network with three nodes (center). Edges and nodes are labeled with their capacities or demands. The optimal mapping is shown on the right side of the figure. The optimal solution is to map node a to C, b to A, and c to B; the virtual link (a, b) is mapped to $C - B - A$, and the virtual link (b, c) is mapped to $A - B$. The cost of this solution is 50.

An ILP model for VNEP with path splitting was presented in [14], while [2] considers a single path variant of the problem. Based on these models, [33] presented the ILP model below. Let the decision variables $x_{vs} = 1$ iff the substrate node s hosts the virtual node v. And let $y_{vwst} = 1$ iff the physical link (s, t) hosts the virtual link (v, w).

$$\min \sum_{(s,t)\in A^S} \sum_{(v,w)\in A^V} B_{vw}\, y_{vwst}$$

$$s.t. \sum_{v\in V^V} C_v x_{vs} \leq C_s \qquad \forall s \in V^S \qquad (21)$$

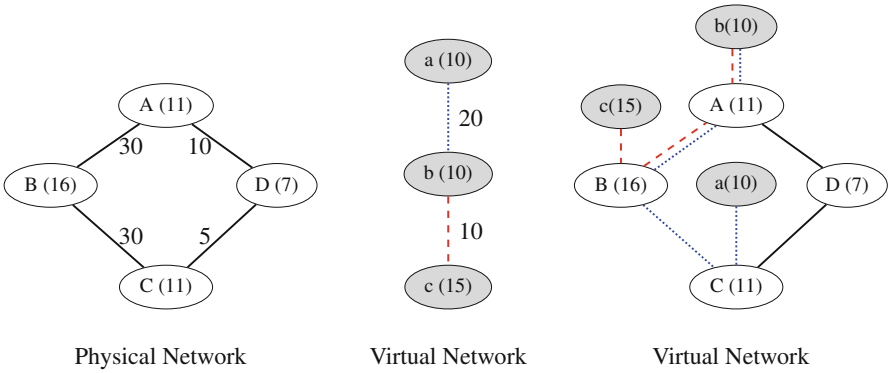


Fig. 2 An input instance for the VNEP (left and center) and the corresponding optimal solution (right)

$$\sum_{s \in V^S} x_{vs} = 1 \qquad \forall v \in V^V \tag{22}$$

$$\sum_{v \in V^V} x_{vs} \leq 1 \qquad \forall s \in V^S \tag{23}$$

$$\sum_{t \in V^S} (y_{vwst} - y_{vwts}) = x_{vs} - x_{ws} \qquad \forall (v,w) \in A^V, s \in V^S \tag{24}$$

$$\sum_{(v,w) \in A^V} B_{vw} y_{vwst} \leq B_{st} \qquad \forall (s,t) \in A^S \tag{25}$$

$$x_{vs} \in \{0,1\} \qquad \forall v \in V^V, s \in V^S \tag{26}$$

$$y_{klmn} \in \{0,1\} \qquad \forall (k,l) \in A^V, (m,n) \in A^S \tag{27}$$

The objective function minimizes the amount of bandwidth used. Constraints (21) ensure that the substrate capacities are not surpassed. Constraints (22) enforce that every virtual node is mapped to a substrate node, while (23) enforce that every substrate node hosts at most one virtual node. Constraints (24) ensure that every virtual link is mapped to a path into the substrate graph. Finally, constraints (25) ensure that the bandwidth capacities of physical edges are not surpassed.

Heuristic Approaches for the VNEP

Since the problem is NP-hard, and only small instances can be solved optimally [33], heuristics are a natural choice as solution approach to tackling this problem. In

this section, we discuss the main heuristic solution approaches adopted by different studies applied to variants of the problem.

Some authors use rounding schemes in an attempt to provide a solution for the problem. In [15] and [2] the authors present ILP models for different variants of VNEP, which they deemed impractical to solve optimally on large instances. To scale the instance sizes, they present rounding schemes applied to the relaxed solutions in order to obtain integer solutions. In both cases, a feasible solution is not guaranteed, but in practice the rounded solution is usually feasible.

In [15] a relaxation of the problem is solved by a solver, and a rounding algorithm is then applied to define to which substrate node each virtual node is mapped. Once nodes are mapped, a multicommodity flow algorithm is applied to determine the substrate path taken for each virtual link. The node mapping works as follows. One virtual node is mapped at a time. For each virtual node, a cluster comprised by a set of substrate nodes is defined according to a distance given as input. Among these nodes, one is selected for hosting the virtual node. The node is chosen according to the mapping variable and other input values.

In [2] the rounding scheme is different. Among the node mapping variables, the highest real value is rounded to 1, defining a mapping to a virtual node. Thus, this variable is fixed, and the relaxed problem is solved again. This procedure is repeated until all virtual nodes are mapped. In fact, four variations of this procedure were proposed in the paper. A similar procedure is applied to link mapping variables.

Considering metaheuristics, ant colony optimization is among the most used. Table 2 presents a summary of results found in the literature. The first column indicates whether the problem is the one defined in the previous section (basic) or if it has additional constraints. In this case, the type of additional constraints that impacts the solution of the problem the most is indicated. Column $|V^V|/|V^S|$ presents the number of virtual ($|V^V|$) and substrate nodes ($|V^S|$) of the largest network considered in the computational results for each reference indicated in the third column of the table. Finally, the last column presents the solution approach of each work indicated in the previous column. The solution approaches are simulated annealing (SA), rounding on the relaxed problem (LP-R), ant colony optimization (ACO), and branch and price (B and P).

Table 2 Solution approaches for solving the VNEP applied in different contexts

Additional constraints	$\|V^V\| / \|V^S\|$	Reference	Solution approach
Node mapping	10/50	[14]	LP-R
QoS requirements	10/100	[18]	ACO
Repository of images	10/50	[2]	LP-R
Multiple mappings	–/35	[12]	ACO
Security	10/500	[4]	SA
Basic	8/80	[33]	B&P

Construction Heuristics for the VNEP

In [32, 33] the authors show that providing in polynomial time a solution that is guaranteed to be feasible is possible only if $P = NP$. Rounding schemes can also be used as construction heuristics. Thus, any proposed method generates solutions not guaranteed to be feasible. In [33] a construction heuristic was used at each node of the branch, bound tree aiming at improving the lower bound for that branch of the tree and then bounding branches earlier. The proposed heuristic works as follows. Each virtual node v is mapped to the free physical node s for which the value of $x_{v,s}$ is the largest. After all nodes are mapped, a breadth-first search is used to map virtual links to paths in the physical substrate graph. The mapping of edges can fail if no path with sufficient bandwidth is found.

In [4] a constructive heuristic is proposed and used by a simulated annealing metaheuristic algorithm. Virtual nodes are placed semi-randomly on substrate nodes, and then physical paths are allocated between these routers for each virtual link. The paths are calculated through Dijkstra's algorithm. The algorithm considers the weight of each physical link (i,j) to be the number of virtual links previously mapped to it and added by one. This weight calculation aims at favoring the selection of physical paths with higher amounts of free resources.

Local Search Heuristics for the VNEP

The most common neighborhoods used in this problem are the `swap` and `change`. In the `swap`, two virtual nodes swap the substrate nodes that host them. In the `change` neighborhood, a virtual node is moved to a substrate node which was not hosting any virtual node [4].

Conclusions

In this chapter, we discussed several network problems. A few computationally easy problems are presented since often they are the core of more complex problems. Moreover, often a small modification of a computationally easy problem can lead to an NP-hard problem. A fundamental multicommodity NP-hard problem was presented, and then two other real-world and more complex problems were discussed in detail: the weight setting problem (WSP) and the virtual network embedded problem (VNEP).

The WSP is used mainly for determining OSPF routing in telecommunication networks. However, it also finds applications in road networks, as the tollbooth problem described in this chapter. Some components of the main algorithms applied to solve the WSP in telecommunication networks were presented, and the problem was discussed. A similar approach was taken with regard to the VNEP. The problem was presented, motivated by components of proposed algorithms. Both are NP-hard network problems whose main solution methods rely on heuristics.

With the increase of the input data of almost all problems, the use of heuristics as a solution approach tends to increase. When designing a heuristic algorithm, it is essential to justify the decisions taken by the algorithms proposed and the

parameters used, and besides reporting the results, statistical analysis is always important to clarify the efficiency of the methods.

Cross-References

- Biased Random-Key Genetic Progamming
- Evolutionary Algorithms
- Random-Key Genetic Algorithms

References

1. Ahuja RK, Orlin JB, Magnanti TL (1993) Network flows: theory, algorithms, and applications. Prentice-Hall, Upper Saddle River
2. Alkmim G, Batista D, da Fonseca N (2013) Mapping virtual networks onto substrate networks. J Internet Serv Appl 4:1–15
3. Bays L, Oliveira R, Buriol L, Barcellos M, Gaspary L (2012) Security-aware optimal resource allocation for virtual network embedding. In: Network and service management (CNSM), pp 378–384
4. Bays L, Oliveira R, Buriol L, Barcellos M, Gaspary L (2014) A heuristic-based algorithm for privacy-oriented virtual network embedding. In: 14th IEEE/IFIP network operations and management symposium (NOMS 2014), pp 1–8
5. Bley A (2011) An integer programming algorithm for routing optimization in ip networks. Algorithmica 60(1):21–45
6. Botero J, Hesselbach X, Duelli M, Schlosser D, Fischer A, de Meer H (2012) Energy efficient virtual network embedding. IEEE Commun Lett 16(5):756–759
7. Broström P, Holmberg K (2006) Multiobjective design of survivable IP networks. Ann Oper Res 147:235–253
8. Buriol L, Resende M, Ribeiro C, Thorup M (2005) A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 46:36–56
9. Buriol L, Resende M, Ribeiro C, Thorup M (2005) A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 46(1):36–56
10. Buriol L, Resende M, Thorup M (2007) Survivable IP network desing with OSPF routing. Networks 49(1):51–64
11. Buriol L, Resende M, Thorup M (2008) Speeding up dynamic shortest-path algorithms. INFORMS J Comput 20:191–204
12. Cao W, Wang H, Liu L (2014) An ant colony optimization algorithm for virtual network embedding. In: Algorithms and architectures for parallel processing. Lecture notes in computer science, vol 8630, pp 299–309
13. Chowdhury N, Boutaba R (2010) A survey of network virtualization. Comput Netw 54(5):862–876
14. Chowdhury NMMK, Rahman MR, Boutaba R (2009) Virtual network embedding with coordinated node and link mapping. In: INFOCOM. IEEE, pp 783–791
15. Chowdhury M, Rahman M, Boutaba R (2012) ViNEYard: virtual network embedding algorithms with coordinated node and link mapping. IEEE/ACM Trans Netw 20(1):206–219
16. Ericsson M, Resende M, Pardalos P (2002) A genetic algorithm for the weight setting problem in OSPF routing. J Comb Optim 6:299–333
17. Even S, Itai A, Shamir A (1976) On the complexity of timetable and multicommodity flow problems. SIAM J Comput 5:691–703

18. Fajjari I, Aitsaadi N, Pujolle G, Zimmermann H (2011) Vne-ac: virtual network embedding algorithm based on ant colony metaheuristic. In: IEEE international conference on communications (ICC), pp 1–6
19. Feamster N, Gao L, Rexford J (2007) How to lease the internet in your spare time. SIGCOMM Comput Commun Rev 37(1):61–64
20. Fortz B, Thorup M (2000) Internet traffic engineering by optimizing OSPF weights. In: INFOCOM, pp 519–528
21. Fortz B, Thorup M (2004) Increasing internet capacity using local search. Comput Optim Appl 29(1):13–48
22. Goldberg A, Tarjan R (1988) A new approach to the maximum-flow problem. J ACM 35:921–940
23. Goldberg A, Kaplan H, Werneck R (2007) Better landmarks within reach. In: Workshop on experimental algorithms, pp 38–51
24. Guerzoni R, Trivisonno R, Vaishnavi I, Despotovic Z, Hecker A, Beker S, Soldani D (2014) A novel approach to virtual networks embedding for SDN management and orchestration. In: Network operations and management symposium (NOMS 2014). IEEE, pp 1–7
25. Hoffman A, Kruskal J (1956) Integral boundary points of convex polyhedra. Ann Math Stud 38:223–246
26. Houidi I, Louati W, Ameur W, Zeghlache D (2011) Virtual network provisioning across multiple substrate networks. Comput Netw 55(4):1011–1023
27. Inführ J, Raidl G (2011) Introducing the virtual network mapping problem with delay, routing and location constraints. In: Pahl J, Reiners T, VoßS (eds) Network optimization. Lecture notes in computer science, vol 6701. Springer, Berlin/Heidelberg, pp 105–117
28. Kleinberg J, Tardos E (2005) Algorithm design. Addison Wesley, Boston
29. Köhler E, Mühring R, Schilling H (2006) Fast point-to-point shortest path computations with arc-flags. In: 9th DIMACS implementation challenge
30. Lauther U (2006) An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. In: 9th DIMACS implementation challenge
31. Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S (2008) Anytime search in dynamic graphs. Artif Intell 172:1613–1643
32. Moura L (2015) Branch & price for the virtual network embedding problem. Master's thesis, Federal University of Rio Grande do Sul, Porto Alegre
33. Moura L, Gaspary L, Buriol LS (2017) A branch-and-price algorithm for the single-path virtual network embedding problem. Networks Online 1–15. https://doi.org/10.1002/net.21798
34. Orlin J (2013) Max flows in o(nm) time, or better. In: Proceedings of the forty-fifth annual ACM symposium on theory of computing, pp 765–774
35. Pioro M, Szentsi A, Harmatos J, Juttner A, Gajownicczek P, Kozdrowski S (2002) On open shortest path first related network optimization problems. Perform Eval 48(4):201–223
36. Ramalingam G, Reps T (1996) An incremental algorithm for a generalization of the shortest-path problem. J Algorithms 21:267–305
37. Shamsi J, Brockmeyer M (2008) Efficient and dependable overlay networks. In: IEEE international symposium on parallel and distributed processing (IPDPS), pp 1–8
38. Stefanello F, Buriol L, Hirsch M, Pardalos P, Querido T, Resende M, Ritt M (2017) On the minimization of traffic congestion in road networks with tolls. Ann Oper Res 249:119–139

Optimization Problems, Models, and Heuristics in Wireless Sensor Networks

39

Vinicius Morais, Fernanda S. H. Souza, and Geraldo R. Mateus

Contents

Introduction 1142
Problem Definition 1145
Optimization Models 1147
Coverage and Density Control 1147
Routing or Connectivity 1148
Clustering and Mobile Sink 1149
Heuristic Methods 1151
Sensor Location, Placement, and Coverage 1151
Density Control, Clustering, Routing, and Sink Mobility 1153
Integrated Problems 1154
Conclusion 1156
Cross-References 1156
References 1157

Abstract

This chapter provides an overview and a comprehensive discussion of problems, models, algorithms, and applications in a vast and growing literature of wireless sensor networks. Being a particular kind of ad hoc network, many power management and communication protocols may be designed specifically for those networks. The critical issues considered in these protocols are the objectives, the

V. Morais (✉) · G. R. Mateus
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
e-mail: vwcmorais@dcc.ufmg.br; mateus@dcc.ufmg.br

F. S. H. Souza
Departamento de Ciência da Computação, Universidade Federal de São João del-Rei, São João del-Reil, Brazil
e-mail: fsumika@ufsj.edu.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_53

quality of communication, the energy consumption, and the network lifetime. Moreover, due to the large-scale aspect inherent in some applications, traditional exact solution approaches are not practical, so heuristics may be adopted instead. The chapter starts by introducing the main concepts in the design of WSN and a wide range of problems and applications. Basic formulations and algorithms are also discussed, together with their benefits and drawbacks.

Keywords
Optimization problems · Heuristics wireless sensor networks

Introduction

Wireless sensor networks (WSN) are a special kind of ad-hoc networks used to monitor events and phenomena in a given place [1, 2]. They have become popular due to their wide applicability in many different areas such as environmental (wildlife and environment), industrial (mines, production, manufacture), and health, in order to monitor temperature, humidity, pressure, and movement [3–5]. The study of such networks has been a fruitful area of research over the years and continues to enable the permanent evolution of technologies.

WSN consist of autonomous, small, and compact devices capable of performing activities such as sensing, processing, and communication. These devices are sensor nodes, and among them, may have one or more special nodes called sink, which is responsible for collecting the sensed data and managing the network. The sensors are made up of sensing boards, processor, communication radio, and battery. Because of their purposes, size, and low cost, they have serious restrictions of energy, low performance, and small radius of communication. For networks installed in areas with difficult access, it becomes unviable exchange or recharge the battery of a sensor. These characteristics imply a limited lifetime of the sensors and of the network itself.

The sensors are used to monitor a working area within a certain radius of coverage, besides being able to exchange information with other sensors and sinks. Two major communication patterns are often adopted: data collection and data dissemination. Data collection consists of sensors sending the collected data to a sink, while data dissemination concerns information from a sink being sent to the sensors. Those communication processes can be performed through one (single) or more (multi) hops or links. In this chapter whenever referring to the transmission of information, both senses are being considered. A sink aggregates the information received from sensors and usually has unlimited or renewable energy. It can be static or mobile, visiting the monitored region and collecting information from the sensors.

The sensor nodes are generally static but may also have mobility [6–8]. *Mobile WSN* consist of a number of sensors that can move on their own and interact with the physical environment. A sensor in a mobile network can also sense, process

information, and communicate like a static node. Mobility may be applied to the whole network or only to a subset of sensors. The degree of mobility can vary, including constant movement of the sensors or alternating periods of mobility/immobility. Those factors impact directly the network dynamics. Alongside with mobile networks, there are four more types of networks, which are distinguished mainly with respect to the place where the sensors are deployed. The sensors can be deployed on land, underground, and underwater [2]. A *terrestrial network*, for instance, consists of several sensors deployed on land, either in an ad-hoc or in a preplanned manner. In an *underground network*, the sensors are placed in caves, mines, or underground, while on an *underwater network*, the devices are deployed into rivers or oceans. Among these networks one can define a *multimedia network*, where the sensors are used to handle multimedia data. While a dense deployment of sensors is often employed in a terrestrial WSN, in an underwater WSN, a sparse deployment is used. The maintenance cost and difficulty of implementation of each network are directly dependent on the application constraints.

The classification of WSN depends primarily on their application. They are commonly classified into monitoring and tracking categories. Monitoring applications include patient monitoring in medical centers, security detection, habitat monitoring, power, inventory location, factory and automation processes, seismic and structural monitoring in the industrial or public context, and home-office environmental monitoring to provide basic services to smart environments, among others. Tracking applications include military missions, wildlife tracking, and traffic control. Consequently, the technologies used for each WSN are directly dependent on the network type. Refer to Yick et al. [2] for a survey of sensor technologies in practical situations.

WSN can also be classified into homogeneous or heterogeneous. WSN are said to be homogeneous, when composed by devices with the same hardware capabilities (processor, memory, battery, and communication device features). Conversely, WSN are heterogeneous when composed by devices with different capabilities. Another possibility to classify WSN is based on their organization. The networks can be hierarchical or flat. A sensor network is hierarchical when nodes are grouped for the purpose of communication, otherwise it is flat.

The lifetime of a WSN can be defined as the network operation time until the first sensor fails due to lack of energy. This chapter does not treat cases of failure for other reasons, such as mechanical failure or if a sensor is destroyed. Another definition of lifetime is given by the monitoring time until the coverage of the entire working area and network connectivity are not guaranteed.

Many factors influence the design of WSN. Fault tolerance, scalability, energy consumption, production cost, environment, network topology, hardware constraints, and transmission rate are some points to be considered. Due to the low battery capacity of the sensors, the energy consumption is one of the most important constraints on WSN [9]. In this context several problems are addressed to ensure coverage, monitoring, and connectivity, aiming to maximize the network lifetime within a minimum quality standard or quality of service (QoS) (QoS usually

refers to quality as perceived by the user and/or application). In spite of energy consumption being the most critical factor in WSN design, other parameters such as number of sensors, coverage, and network connectivity must also be considered.

The problems that appear in the context of WSN are studied independently or in an integrated manner. In general, to define them it is supposed that the working area to be covered is divided into small squares which represent points or demand nodes that require coverage by at least one sensor. This is a well-known approach applied in wireless network design. Each point concentrates the demand of its square. A point is considered covered by a sensor when the referred sensor is able to monitor it and if the point is within the radius of coverage of that sensor. The smaller are the square dimensions, the closer they are to describing continuous area. In the literature there are several works that do not make use of this discretization process, opting to explore the continuous space. Another assumption usually found in the literature is the use of unit disk graph to model the communication among sensors. A homogeneous transmission range for the sensors is considered and taken as the disk radius. Many works employ this model, although transmission ranges can be heterogeneous and also not defined as perfect disks. However, due to its theoretical simplicity, it is commonly applied. An evolution of this model, named quasi unit disk graph is proposed in [10].

Many algorithms can be devised to deal with WSN problems. Regarding optimality guarantees, they can be divided into exact approaches and heuristics. The former comprise algorithms developed to solve mathematical models based on integer linear programming (ILP), while the latter include proposals which relieve optimality conditions in order to provide scalability. Also, algorithms can perform in a centralized or distributed fashion. Centralized algorithms concern decision-making is done by a single entity in the network owning global information of the entire network. This entity is usually a sink node which is able to disseminate the solution for all sensors in the network. On the other hand, distributed algorithms indicate that many entities in the network take a joint decision based only on local information. Global information can improve the optimization process but is not always available and may not be applied in practice due to the overhead not allowed in real-time applications. Local information seems more reasonable in practice, but suffers from lower-quality solutions compared to the ones with global information. An alternative to take advantage of both methods is to use clustering, which can optimize locally inside each cluster. Thus, both exact methods and heuristics can be accomplished in such hybrid solutions. Distributed algorithms concern a wide topic of research [11–14] and are out of scope of this chapter.

The remainder of the chapter is organized as the following. A brief definition of classical problems appearing in WSN is presented in section "Problem Definition". Mixed integer linear programming formulations for these problems are outlined in section "Optimization Models". The aim is to formally describe some problems and give some intuition about exact solution approaches. Heuristic methods applied on the design of WSN are addressed in section "Heuristic Methods". Finally, the chapter is closed in section "Conclusion".

Problem Definition

To monitor inhospitable and inaccessible environments, sensors are deployed in an ad-hoc manner, launched from an aircraft or an unmanned vehicle. Thus, it is not possible to know beforehand their exact locations. However, knowing the location of these sensors is generally important to route data and to ensure connectivity. The problem of defining the positions of the sensors is known as *localization problem*. Some localization methods include Global Positioning System (GPS), beacon (anchor) nodes, and proximity-based localization that make use of the coordinates of neighbor nodes to determine their own localization. A review of location problems in WSN is given in [15].

It is straightforward to note that a fundamental requirement in the design of WSN is to ensure coverage of the working area. This problem is named as the *coverage problem*. The coverage level can be total or partial. In general, full coverage is often preferred. Therefore, the coverage problem chooses a set of active sensors such that each point is within a range of at least one sensor. The loss of coverage in at least one point can be characterized as end of the network lifetime.

To monitor accessible environments, the sensors may be optimally deployed in a preplanned manner. The *placement (deployment) problem* seeks to minimize the number of sensors deployed in order to guarantee connectivity and coverage of the entire monitoring area. The placement problem is a variant of the well-known art gallery problem [16], the problem of placing the fewest number of guards in a given area such that the surveillance is guaranteed. For the following problems, a usual assumption is to consider that the location of the sensors is known.

Another fundamental requirement in the design of WSN is the connectivity. The data collected by the sensors must be delivered to the sinks. Thus, routes must be established connecting each active sensor to a sink. A feasible route consists of links between pairs of sensors/sinks that respect the communication radius of each node. The fewer is the number of links, the smaller is the energy consumption. The *routing problem* aims to find feasible routes from each active sensor to the sinks, or vice versa, while optimizing a given resource. A solution for the routing problem consists of rooted trees, centered at the sinks, spanning a subset of active sensors. Automatically, the sensors nearest to the sinks may spend too much energy given the number of routes traversing those nodes. Such a drawback is known as the sink neighborhood problem [17]. There are two kinds of routing methods: reactive routing methods and proactive routing methods. In reactive methods the routes are calculated just in time. In proactive methods the routing tables are created and stored regardless of the moment the routes are used. However, in networks with large number of sensors, storing the routing tables can lead to excessive usage of memory. To overcome this problem, the sensors can be clustered and communication hierarchies be defined.

The sensors consume energy to perform communication, process, and aggregate data, besides maintenance, activation, or deactivation processes. There are several ways of dealing with the energy consumption. One possibility to extend the network

lifetime is to use a large number of sensors to ensure the desired coverage. However, to avoid using too much energy at a redundant coverage, the sensors do not perform at the same time. In this case, each sensor can be active, working on the network, or stay on standby, sleep mode, since energy consumption is extremely low in this mode. When active sensors fail due to depletion of their batteries, new sensors must be activated. The optimization problem arising in this context is the *density control problem* (DCP). The purpose of DCP is to schedule a sequence of activation and deactivation of sensor nodes in order to minimize the energy consumption and maximize the network lifetime. The network lifetime comes to an end when the network requirements are no longer ensured.

All the problems above have been considered for flat WSN. Whereas the communication among sensors and sinks is one of the most expensive operations in terms of energy consumption, an alternative to flat networks is to set hierarchies. In hierarchical networks, the sensors are clustered, and a single sensor is elected as the *cluster-head* in each cluster. It is up to the cluster-head aggregating and transmitting the information directly to a sink or through other cluster-heads. In such a *clustering problem*, from time to time, the cluster-head role can be swapped among the sensors within the same cluster to avoid premature failure due to lack of energy. Some classical combinatorial optimization problems such as p-median and p-center may apply in a clusterization scheme.

In the routing problem for hierarchical WSN, two kinds of routes are considered: *intra-cluster* route and *intercluster* route. In intra-cluster routes the communication occurs among sensors and cluster-head within the same cluster through single or multi-hop strategies. In intercluster routes the connections are among cluster-heads and sinks. In both levels a tree topology must be defined to transmit the information from the sensors to their cluster-heads and from the cluster-heads to the sinks.

In some applications the mobility of the sinks is explored to gather sensed information through the network. This approach prevents the sensors to spend their limited energy in relaying of messages through a long path until the sinks. However, the message delivery latency can increase significantly, thanks to the long route traversed by a sink to visit the whole working area. In order to go around of such problem, clustering schemes must be used so that each sink only visits a reduced set of sensors (for instance, the cluster-heads). In this context, the communication between sensors and sink is only possible through a cluster-head using multi-hop or single-hop approaches to define the *intra-cluster* routes. Classical optimization problems such as shortest path and traveling salesman problem (TSP) are used to model the trails traversed by the mobile sinks.

Figure 1 illustrates optimization problems in WSN. Different tasks and objectives may be desired requiring specific approaches to deal with the problems described. In Fig. 2, for instance, two different approaches are presented exploring the mobility of the sink. In the first approach, a mobile sink visits each sensor in the working area to collect sensed data. This problem can be modeled with the TSP. Another alternative is to define a reduced TSP route, defining a subset of sensors to be visited by the mobile sink such that all the other sensors are close enough to establish the communication. This approach can be seen as a TSP by neighborhood.

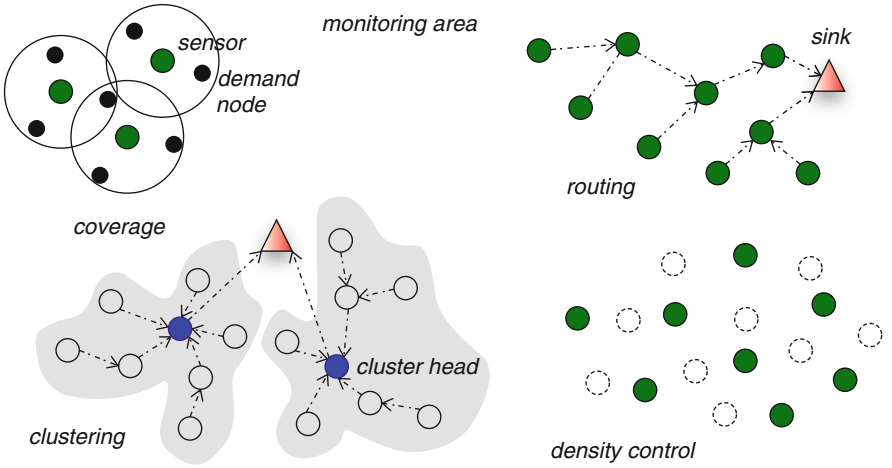


Fig. 1 Illustration of classical optimization problems in WSN

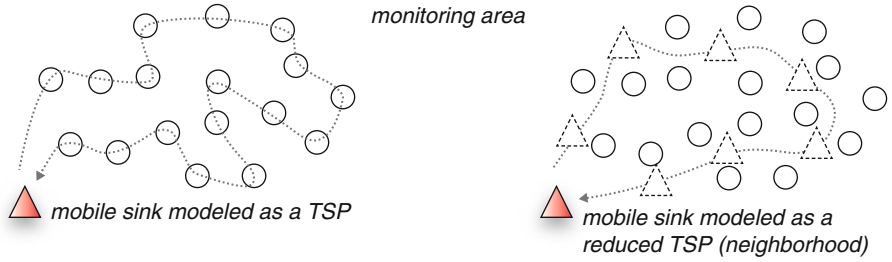


Fig. 2 Illustration of mobile sink approaches modeled with the TSP

Optimization Models

As presented earlier, many optimization problems arise in the design of WSN. Many of these problems are variants of classical combinatorial optimization problems that can be formulated as linear mixed-integer programming (MIP) models [18]. In the following, mathematical models for some of these problems are presented.

Coverage and Density Control

Mathematical formulations for coverage and density control problems can be found in [19–21]. The models consider the following notation: $S = \{1, \ldots, n\}$ is the set of sensors; $P = \{1, \ldots, p\}$ is the set of demand points to be covered by the sensors; matrix $A(n \times p)$ indicates whether a sensor $i \in S$ is able to cover ($a_{ij} = 1$) or not ($a_{ij} = 0$) the point $j \in P$ regarding the sensing range R_s^i. To each sensor $i \in S$, an activation energy E_a^i is also assigned.

A formulation to the coverage and density control problems uses binary variables $\{x_i \in \mathbb{B} : i \in S\}$ and $\{y_{ij} \in \mathbb{B} : i \in S, j \in P\}$ to define a subset of sensors selected to be active and the coverage relationship between S and P. If $x_i = 1$ holds, sensor $i \in S$ is active. If $y_{i,j} = 1$, demand point $j \in P$ is covered by sensor $i \in S$. Then, the coverage and density control formulation is given by:

$$\min \left\{ \sum_{i \in S} E_a^i x_i : (x, y) \in (2) - (5) \right\} \tag{1}$$

$$\sum_{i \in S} a_{ij}\ y_{ij} \quad \geq 1 \qquad \forall j \in P; \tag{2}$$

$$a_{ij}\ y_{ij} \quad \leq x_i \qquad \forall i \in S, \qquad \forall j \in P; \tag{3}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i \in S, \qquad \forall j \in P; \tag{4}$$

$$x_i \in \{0, 1\} \qquad \forall i \in S. \tag{5}$$

Constraints (2), (3), (4), and (5) assure that active sensors cover the set of demand points. Constraints (2) guarantee that every demand point $j \in P$ is covered by at least one sensor $i \in S$. Constraints (3) ensure that only active sensors can provide the coverage of demand points. Constraints (4)–(5) impose variables to be binary. Finally, objective function (1) minimizes the energy consumed by the activation of sensor nodes.

Routing or Connectivity

Mathematical models exploring connectivity in the context of WSN are described in [19, 21, 22]. The models consider a set of static sinks $M = \{1, \ldots, m\}$ and a communication range (R_c^i) associated with each sensor $i \in S$. The matrix $B(n \times (n + m))$ indicates whether a sensor or sink $i \in S$ and $j \in S \cup M$ relies $(b_{ij} = 1)$ or not $(b_{ij} = 0)$ within the communication range R_c^i of each other. To each sensor $i \in S$ is also assigned the routing energy (E_r^i) required to communicate with other sensors or sinks.

Let $\{z_{jk}^i \in \mathbb{B} : i \in S, j \in S, k \in S \cup M\}$ denote decision variables used to indicate whether the sensors $j \in S$ and $k \in S \cup M$ belong $(z_{jk}^i = 1)$ or not $(z_{jk}^i = 0)$ to the communication path from sensor $i \in S$ to a sink node k. Alongside with the decision variables x_i defined in the first model, the routing problem, modeled as routing trees, can be formulated as:

$$\min \left\{ \sum_{i \in S} E_a^i\ x_i + \sum_{i \in S} \sum_{j \in S} \sum_{k \in S \cup M} E_r^i\ z_{jk}^i : (x, z) \in (7) - (12) \right\} \tag{6}$$

$$\sum_{k \in S \cup M} b_{jk}\, z^i_{jk} \leq x_j \qquad \forall i \in S \backslash \{k\}, \quad \forall j \in S \backslash \{k\}; \tag{7}$$

$$\sum_{j \in S} b_{jk}\, z^i_{jk} \leq x_k \qquad \forall i \in S \backslash \{k\}, \quad \forall k \in S \backslash \{j\} \cup M; \tag{8}$$

$$\sum_{j \in S} b_{jk}\, z^i_{jk} - \sum_{l \in S \cup M} b_{kl}\, z^i_{kl} = 0 \qquad \forall i \in S, \quad \forall k \in S \backslash \{i\}; \tag{9}$$

$$-\sum_{l \in S \cup M} b_{kl}\, z^i_{kl} = -x_i \qquad \forall i \in S, i = k; \tag{10}$$

$$z^i_{jk} \in \{0, 1\} \qquad \forall i \in S, j \in S, k \in S \cup M \tag{11}$$

$$x_i \in \{0, 1\} \qquad \forall i \in S. \tag{12}$$

Here, constraints (7), (8), (9), (10), (11), and (12) ensure connectivity among sensors and sinks. Constraints (7)–(8) assure that communication is only allowed among active sensors. Constraints (9) guarantee the flow balance for transshipment nodes. Constraints (10) indicate that the flow originates at active sensors. Constraints (11)–(12) impose the domain of the variables. Finally, the objective function (1) minimizes the routing and activation energies consumed.

Clustering and Mobile Sink

Valle et al. [23] proposed an integrated model for clustering and routing problems. The model considers a set of mobile sinks $M = \{1, \ldots, m\}$ and a set of arcs $A = \{(i, j), (j, i) : \forall i, j \in S, i \neq j\}$. Each arc $(i, j) \in A$ denotes a possible point-to-point movement for a sink. Additionally, d_{ij} is the Euclidean distance between i and $j \in S$. It is assumed that all sinks start from an initial sensor $i = 1$ and return to an artificial node 0, such that $\bar{A} = A \cup \{(i, 0) : \forall i \in S \backslash \{1\}\}$ and $\{(i, 0) : d_{i0} = d_{1i}, \forall i \in S \backslash \{1\}\}$. It is important to point out that each cluster-head is visited only once and by exactly one sink; a sensor $i \in S$ must be a cluster-head or be covered by a cluster-head; at least one cluster-head should belong to the set of neighbors of sensor i: $N(i) = \{j \in S : d_{ij} \leq R^i_c\}$.

In order to model the integrated clustering and routing problem, assume the following decision variables: $\{r^m_i \in \mathbb{B} : m \in M, i \in S\}$, taking value $r^m_i = 1$ if sensor $i \in S$ is a cluster-head in the route of the sink $m \in M$, otherwise $r^m_i = 0$; $\{w^m_{ij} \in \mathbb{B} : m \in M, (i, j) \in \bar{A}\}$ that takes value $w^m_{ij} = 1$ if arc $(i, j) \in \bar{A}$ belongs to route of the sink $m \in M$, otherwise $w^m_{ij} = 0$; $\{v^m_{ij} \in \mathbb{R}_+ : m \in M, (i, j) \in \bar{A}\}$, indicating the flow, originated at m, traversing arc $(i, j) \in \bar{A}$; and, finally, $\{h \in \mathbb{R}_+\}$ that denotes the longest length among all routes. The integrated clustering and routing problems can be formulated by the following MIP model:

$$\min \{h : (r, w, v, h) \in (14) - (27)\} \tag{13}$$

$$\sum_{i \in S \setminus \{1\}} v_{1,i}^m = \sum_{i \in \{0\} \cup S \setminus \{1\}} r_i^m \qquad \forall m \in M; \tag{14}$$

$$\sum_{j \in \{0\} \cup S \setminus \{1\}} v_{ij}^m - \sum_{j \in S} v_{ji}^m = -r_i^m \qquad \forall i \in S \setminus \{1\}, \quad \forall m \in M; \tag{15}$$

$$\sum_{i \in S} v_{i,0}^m = 1 \qquad \forall m \in M; \tag{16}$$

$$v_{ij}^m \leq n w_{ij}^m \qquad \forall (i, j) \in \bar{A}, \quad \forall m \in M; \tag{17}$$

$$w_{ij}^m \leq r_i^m \qquad \forall (i, j) \in \bar{A}, \quad \forall m \in M; \tag{18}$$

$$w_{ij}^m \leq r_j^m \qquad \forall (i, j) \in \bar{A}, \quad \forall m \in M; \tag{19}$$

$$\sum_{m \in M} r_i^m \leq 1 \qquad \forall i \in S \setminus \{1\}; \tag{20}$$

$$\sum_{m \in M} \sum_{j \in N(i)} r_j^m + \sum_{m \in M} r_i^m \geq 1 \qquad \forall i \in S \setminus \{1\}; \tag{21}$$

$$\sum_{j \in S \cup \{0\}} w_{ij}^m \leq 1 \qquad \forall i \in S, \quad \forall m \in M; \tag{22}$$

$$h \geq \sum_{(i,j) \in \bar{A}} d_{ij} w_{ij}^m \qquad \forall m \in M; \tag{23}$$

$$r_1^m = r_0^m = 1 \qquad \forall m \in M; \tag{24}$$

$$r_i^m \in \{0, 1\} \qquad \forall i \in S \cup \{0\}, \quad \forall m \in M; \tag{25}$$

$$v_{ij}^m \geq 0 \qquad \forall (i, j) \in \bar{A}, \quad \forall m \in M; \tag{26}$$

$$w_{ij}^m \in \{0, 1\} \qquad \forall (i, j) \in \bar{A}, \quad \forall m \in M. \tag{27}$$

Constraints (14), (15), (16), (17), (18), (19), (20), (21), (22), (23), (24), (25), (26), and (27) ensure the integration of clustering and routing problems using mobile sinks. Constraints (14), (15), and (16) guarantee the flow conservation for nodes $\{1\}$, $S \setminus \{1\}$ and $\{0\}$, respectively. Note that constraints (14) assure that the flow of commodity m starting at node 1 is equal to the number of cluster-heads to be visited in the route plus the artificial node $\{0\}$. On the other hand, constraints (15) impose that each node i visited by $m \in M$ must consume a unit of flow. Inequalities (17), (18), and (19) are coupling constraints, ensuring that commodities only traverse arcs selected for the route and arcs of the route have endpoints consuming a unit of flow. Constraints (20) guarantee that a cluster-head is visited once by a single route. Constraints (21) assure that each sensor is a cluster-head or has a cluster-

head as neighbor. Taken together, constraints (14), (15), (16) and (22) guarantee the topology of selected arcs induce M elementary routes connecting vertices 1 and 0. Constraints (23) define the longest length among all routes to be minimized by the objective function (13). Constraints (24) impose that the starting node 1 and the final artificial node 0 are part of all routes. Finally, constraints (25), (26), and (27) define the domain of variables.

Models (1), (6), and (13) can be implemented in optimization packages and solved with a branch-and-bound (BB) algorithm. However, the main drawback of this approach is that only instances of limited size are solvable in feasible computational time. Since WSN are designed as large-scale networks, it is not expected that a traditional BB is able to solve the problem to optimality. Therefore, one must resort to heuristic-based methods to tackle the problems.

Heuristic Methods

This section draws attention to heuristic-based methods used in the design of WSN. In general, heuristics are designed to face challenging and complex optimization problems in which exact optimization methods fail, especially when dealing with time-constrained online applications that are combinatorial in nature. Due to the practical and technological relevance of WSN, heuristics are preferentially the methods to be used. However, each heuristic is designed to tackle a specific problem. So, a given approach may not be generalized to solve different problems. Motivated by the need of general solution strategies, optimization researches have been focusing on the study of metaheuristics. A metaheuristic combines intensification and diversification procedures to perform guided searches through one or more neighborhood structures. Some metaheuristics used in the design of WSN are greedy randomized adaptive search procedure (GRASP) [24], path relinking (PR) [25], simulated annealing (SA) [26], iterated local search (ILS) [27], tabu search (TS) [28], variable neighborhood search (VNS) [29, 30], genetic algorithms (GA) [31], memetic algorithms (MA) [32], ant colony optimization (ACO) [33], and particle swarm optimization (PSO) [34]. For supplementary reading, Glover and Kochenberger [35] are referred to provide a good overview of the most popular metaheuristics. In addition to [36–40], here, the reader can find some hints of how to use heuristics to solve practical WSN optimization problems when studied independently or in an integrated manner.

Sensor Location, Placement, and Coverage

As indicated above, the design of WSN has to concern with several classical problems. These include localization, placement, coverage, density control, clustering, routing, and sink mobility. In general, the purpose of solving these problems is to maximize the network lifetime, keeping connectivity, ensuring coverage, and

reducing cost. Many existing applications require information about the geographic position of each node in the network. So, the localization problem must be solved first. The heuristics proposed in the literature for localization are generally based on GPS system, beacon (anchor) nodes, and proximity-based localization. These methods make use of the position of neighbor sensors, obtained during the installation of the network, to determine the position of a sensor. Molina and Alba [41], for instance, proposed heuristics based on SA, GA, and PSO to solve the localization problem. Their heuristics determine the sensor nodes locations using the trilateration technique. The global objective of their heuristic is to minimize the measured distance error comparing the node-to-node distances with the distance measured when using GPS. Shahrokhzadeh et al. [42] proposed a centralized (solved by a sink) SA-based heuristic for solving sensor nodes localization problem. Their heuristic is based on Euclidean distance and mathematical techniques, such as trilateration and triangulation.

The placement of the sensors is a key factor in the design of WSN. The deployment may be an activity performed during the installation of a network or may also be a continuous process to replace failed sensors or to improve the coverage over a certain area. As stated above, the placement problem seeks to minimize the number of sensors deployed in order to guarantee connectivity and coverage of the entire monitoring area. A number of deterministic greedy heuristics can be described to determine the minimum number of sensor needed. In general, those methods try to anticipate the sensor deployment in all candidates' points and keep the best position selected. Brazil et al. [43] modeled the placement problem as a version of the Steiner problem and solved it by means of a greedy-based heuristic. Sasikumar et al. [44] proposed two-phase heuristics for placing sensors in a heterogeneous network, called nearest to base station and max residual capacity. In this work, they separate the nodes on: sensors, relay nodes (RN), and base station (sink node). The RN node performs functions equivalent to a cluster-head, but may have unlimited energy. In the first phase, their heuristics place the RN nodes, wherein the placement problem is modeled as the minimum set covering problem. Then the sensors are placed on the second phase with a greedy criterion based either on the distance to the RN nodes or on residual energy remaining on the node. To solve the sink placement problem, Laszka et al. [45] provided a GA-based heuristic. SA-based heuristics for placement and coverage problems were also developed in [46].

Deschinkel [47] concentrates on a centralized scheme to solve the coverage problem, where a large number of sensors are randomly deployed in the monitoring area. The problem was modeled as the non-disjoint cover sets problem and solved by means of a column generation (CG) heuristic. In this procedure the subproblem is solved with a heuristic for the classical set covering problem that uses the dual multipliers from the master problem. They compare the results obtained by the heuristic with that provided by a MIP solver. Cardei et al. [48] also deal with coverage problem. The interesting aspect of their work is given when the authors consider the property that each sensor has adjustable sensing ranges. To solve the problem, a greedy-based heuristic was used.

Density Control, Clustering, Routing, and Sink Mobility

The solution algorithms for the placement problem may define a dense network. This characteristic increases the robustness against failures, however, leads to a large redundancy that can result in congestion and waste of energy. To overcome this problem, the density control problem is used to select a subset of sensors to be active while keeping the others inactive (sleep mode), which allows the reduction of the initial topology. Delicato et al. [49] formulated the DCP problem as a knapsack problem. In this approach, the expected time horizon is divided in small time periods. For each period, a subset of sensors must be activated. By formulating the DCP as a knapsack problem, each period is equivalent to a knapsack, so the problem is to find a set of nodes (items) to keep active (to be inserted in a knapsack) per period. In their model, coverage and connectivity are also considered. To solve the problem, a greedy-based heuristic was proposed. In this heuristic the sensors are sorted by their residual energy and then, following the given preference, the sensors are set to be active (inserted in a knapsack).

An interesting and effective multi-start (MS)-based heuristic to solve the density control problem is given by Karasabun et al. [50]. Their heuristic is composed by two phases, a constructive routine to define a set of initial active sensors and an iterative improving local search procedure. To ensure the network connectivity, the Steiner problem is solved over the active sensor nodes.

Clustering is another optimization problem considered when defining the network topology. The clustering problem directly contributes for the scalability of networks. Many existing protocols for defining clusters and selecting cluster-heads associated with them are based on random algorithms or heuristics to classical combinatorial optimization problems, such as p-median, dominating set, and set covering problems. In such approaches, every sensor node is either in a set of cluster-heads or is assigned to a cluster-head. In general, among the cluster-heads, a multi-hop communication scheme is used, while inside a cluster multi- and/or single-hop communication could be used. Santos et al. [51] modeled the clustering problem as the independent dominating set problem. Two greedy heuristics are described in that reference.

Matos et al. [52] call attention to the importance of rotating the sensor that acts as cluster-head within a single cluster. Cluster-heads consume much more energy than a sensor that does not perform this feature, since they need to aggregate, send, and receive sensing data. Thus, cluster-head rotation helps to avoid a premature death of a sensor, disconnecting the network. The rotation can be done from time to time or after some amount of data has been transferred by the network. In addition, Matos et al. [52] also proposed a centralized GRASP-based heuristic with path relinking as an intensification phase to solve the clustering problem. The problem was modeled as the p-median problem, where the cluster-heads are chosen among the alive sensors. The intra-clusters communication follows the single-hop strategy. Their heuristic is composed of three phases. In the constructive phase, a feasible solution is generated by the randomized greedy algorithm described in [53]. Then, a swap local search is applied. Finally, a path relinking routine is applied as

intensification phase. To attest the effectiveness of their approach, they compared their heuristic with methods described in [54, 55]. Albath et al. [56] consider that a sensor to be selected as cluster-head must have enough residual energy. In their work, the problem of properly choosing cluster-heads was modeled as the minimum dominating set problem.

Looking for metaheuristics methods, Ferentinos and Tsiligiridis [57] described a GA-based heuristic used in an agriculture application. Their algorithm is designed to solve clustering and density control problem while respecting connectivity constraints. Another GA-based heuristic for clustering problem is addressed by Kuila et al. [58]. Ting and Liao [59] described the clustering problem as the k-cover problem and solved it with a MA-based heuristic.

The transmission of sensing data and the dissemination of information in WSN are referred as the routing problem. This problem is defined mainly when the nodes position and the cluster-heads selected are already known. The main objective when solving the routing problem is to guarantee the network connectivity. Routing in WSN design is tackled by many authors [19,21,22,60] through a multi-hop strategy and fixed sink nodes as well as through mobile sinks [23,61,62].

Minimum spanning tree, shortest path, and TSP are preferentially used to model the routing problem in WSN design. In general the edge weight (length or cost) represents the energy needed to transmit data. Thus, routing algorithm based on Kruskal, Prim, Dijkstra, Bellman-Ford, or TSP methods can be applied. As stated in section"Problem Definition", the sink neighborhood problem [17] arises when dealing with fixed sink nodes. To overcome that problem, some alternatives take place. Some works consider multiple sinks or, in some contexts, explore the network by mobile sinks. More sinks result in shorter routes from sensors to their closest sink, while mobility may lead to an efficient scheme for the energy control.

Centralized and distributed heuristics to control and coordinate the current movement of multiple sinks seeking for the lifetime maximization in WSN are described by Basagni et al. [63]. In this reference, a sink is considered active when it is stopped, waiting for the sensor data, and inactive when it is moving through the network. In their approach at least one sink must move at a time, so it is necessary to define a schedule for the movements of sinks. The routing scheme was modeled with TSP problem and solved by a Christofides' heuristic [64].

To control the sink movements, Basagni et al. [17] introduced a distributed heuristic, named greedy maximum residual energy. In their heuristic, a sink defines its own route by moving from its current location to a new position by giving preference to areas with the highest residual energy. More precisely, the greedy criterion of the proposed heuristic is the amount of energy left in the sensors around a mobile sink that moves in direction of the sensors with higher energy to collect their data.

Integrated Problems

Most the works found in the WSN literature describe hierarchical approaches to the design of networks. Those works are drawn for solving WSN problems in multiple steps. Although less, there are works in the literature that address a number of

classical WSN problems in an integrated manner [21, 65–72]. A natural clustering and routing problem is the minimum connected dominating set problem [73–75], the problem of finding a connected dominating set of cluster-heads with smallest cardinality. Aioffi et al. [66], for instance, have provided a constructive heuristic to an integrated problem, named integrated clustering and routing problem (ICRP). Their proposed heuristic is based on the cheapest insertion algorithm for the Euclidean TSP. In such method, a set of routes is iteratively defined, one for each mobile sink, at the same time as the sensors are grouped into clusters. Each sink node visits some cluster-heads, in such a way that every sensor node in the network is either a cluster-head visited by a route or lies within a cluster associated with a cluster-head in one of the possible routes. The greedy criterion of the proposed heuristic is based on the distance among the last cluster-head assigned to a route and the other cluster-heads not routed yet. Hybrid heuristics based on GRASP and ILS

Table 1 Description of the cited references on heuristic-based methods for WSN problems. Consider the following acronyms list: coverage problem (CP), density control (DCP), localization problem (LP), routing problem (RP), clustering problem (CL), and placement problem (PP). Multi-hop communication (MhC), fixed sink (FS), mobile sink (MoS), and multiple sink (MuS)

Reference	Problems										Methods
Aioffi et al. [82]	CP	DCP	RP	CL	–	–	MhC	–	MoS	–	Greedy
Aioffi et al. [66]	–	–	RP	CL	–	–	–	–	MoS	MuS	GRASP, ILS
Alba and Molina [46]	CP	–	–	–	PP	–	–	–	–		SA
Albath et al. [56]	–	–	–	CL	–	–	–	FS	–	–	Greedy
Basagni et al. [17]	–	–	–	–	–	–	–	–	MoS	–	Greedy
Basagni et al. [63]	–	–	RP	–	PP	–	–	FS	MoS	MuS	Greedy
Brazil et al. [43]	–	–	–	–	PP	–	–	–	–	–	Greedy
Cardei et al. [48]	CP	DCP	–	–	–	–	–	–	–	–	Greedy
Delicato et al. [49]	–	DCP	–	–	–	–	–	–	–	–	Greedy
Deschinkel [47]	CP	–	–	–	–	–	–	–	–	–	CG heuristic
Ferentinos and Tsiligiridis [57]	–	DCP	RP	CL	–	–	–	FS	–	–	GA
Guney et al. [68]	CP	–	RP	–	–	LP	–	–	–	–	TS
Karasabun et al. [50]	–	DCP	–	–	–	–	–	–	–	–	MS
Kuila et al. [58]	–	–	–	CL	–	–	–	FS	–	–	GA
Laszka et al. [45]	–	–	–	–	PP	LP	–	–	–	–	GA
Lin and Uster [83]	–	–	–	–	PP	–	–	–	–	–	MS
Matos et al. [52]	CP	–	RP	–	–	–	MhC	–	–	–	GRASP, PR
Molina and Alba [41]	–	–	–	–	–	LP	–	–	–	–	SA, GA, PSO
Santos et al. [51]	–	–	–	CL	–	–	–	FS	–	–	Greedy
Sasikumar et al. [44]	–	–	–	–	PP	–	–	–	–	–	Greedy
Shahrokhzadeh et al. [42]	–	–	RP	–	PP	–	–	–	–	–	SA
Ting and Liao [59]	–	–	–	CL	–	–	–	–	–	–	MA
Turkogullari et al. [76]	–	–	RP	–	PP	–	–	–	–	–	MIP-heu.
Uster and Lin [67]	–	DCP	–	CL	–	–	–	–	–	MuS	Constructive heu.
Xing et al. [65]	–	–	RP	CL	–	–	–	–	–	–	Greedy

metaheuristics were also applied to solve the problem. Another integrated approach is due to Xing et al. [65] that also studied clustering and routing problems. To solve the problem, a heuristic was used in the following schemes. First, it solves the Steiner problem to select the cluster-heads and then uses a TSP local search heuristic to find a route for each mobile sink.

A TS-based heuristic for an integrated coverage, sink location, and routing problems can be found in Guney et al. [68]. The heuristic starts by finding near-optimal sensor locations satisfying the coverage requirements, and then it solves the sink location and the data routing problem modeled as the p-median problem. Türkogullari et al. [76] proposed a hybrid heuristic to solve large instance of an integrated problem involving sensor placement, density control, sink location, and routing problem with low conservation, energy consumption, and budget constraints. In such algorithm, the optimal sensor-to-sink routes are solved as a linear program (LP), while the sink location problem is modeled as a set covering problem and solved by a disjoint sets heuristic. Another heuristic based on exact approach, used to solve large and real-life sink location and routing problems, could be found in [77].

Table 1 summarizes the previous cited works, highlighting the problems solved and the heuristic methods used. For an overview of the mathematical aspects of network optimization problems, we also refer to [78–81].

Conclusion

In this chapter, an overview of classical optimization problems and solution methods in the design of wireless sensor networks are presented. WSN have been widely studied due to their importance in many practical applications. Unlike a general network, WSN are designed for specific applications and take into account the application goals, the associated costs, the hardware capabilities, and other system constraints. Therefore, optimization techniques are shown to be an essential part in the design of new protocols. The integration of two or more problems, such as coverage, routing, density control, clustering, and so on, requires even more sophisticated approaches to achieve high-quality solutions. It is straightforward to conclude that optimization tools play a key role in the design of WSN. The referred methods in this chapter have the capacity to significantly improve metrics such as network lifetime, coverage rate, and communication delay. Looking to the future, WSN represent an attractive research area with several new possibilities where the major challenge is to approximate the theoretical tools for practical applications.

Cross-References

▸ Genetic Algorithms
▸ GRASP

- Iterated Local Search
- Memetic Algorithms
- Network Optimization
- Particle Swarm Methods
- Tabu Search
- Variable Neighborhood Search

Acknowledgments This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG), and Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES). Vinicius Morais is funded by CAPES BEX 7461/14-3.

References

1. Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E (2002) Wireless sensor networks: a survey. Comput Netw 38(4):393–422
2. Yick J, Mukherjee B, Ghosal D (2008) Wireless sensor network survey. Comput Netw 52(12):2292–2330
3. Xu N (2002) A survey of sensor network applications. In: IEEE communications magazine electronics, robotics and automotive mechanics conference, Washington, DC, pp 102–114
4. Arampatzis T, Lygeros J, Member S, Manesis S (2005) A survey of applications of wireless sensors and wireless sensor networks. In: Proceedings of the 13th Mediterranean conference on control and automation, Limassol, pp 719–724
5. Kuorilehto M, Hännikäinen M, Hämäläinen TD (2005) A survey of application distribution in wireless sensor networks. EURASIP J Wirel Commun Netw 2005:774–788
6. Yoneki E, Bacon J (2005) A survey of wireless sensor network technologies: research trends and middleware's role. Technical report UCAM-CL-TR-646, Computer Laboratory, University of Cambridge, Cambridge, UK
7. Munir S, Ren B, Jiao W, Wang B, Xie D, Ma J (2007) Mobile wireless sensor network: architecture and enabling technologies for ubiquitous computing. In: 21st international conference on advanced information networking and applications workshops, 2007, AINAW'07, Niagara Falls, vol 2, pp 113–120
8. Khan MI, Gansterer WN, Haring G (2013) Static vs. mobile sink: the influence of basic parameters on energy efficiency in wireless sensor networks. Comput Commun 36:965–978
9. Anastasi G, Conti M, Di Francesco M, Passarella A (2009) Energy conservation in wireless sensor networks: a survey. Ad Hoc Netw 7:537–568
10. Kuhn F, Wattenhofer R, Zollinger A (2008) Ad hoc networks beyond unit disk graphs. Wirel Netw 14:715–729
11. Al-Karaki JN, Kamal AE (2004) Routing techniques in wireless sensor networks: a survey. Wirel Commun 11:6–28
12. Langendoen K, Reijers N (2003) Distributed localization in wireless sensor networks: a quantitative comparison. Comput Netw 43:499–518
13. Madan R, Lall S (2006) Distributed algorithms for maximum lifetime routing in wireless sensor networks. IEEE Trans Wirel Commun 5:2185–2193
14. Kubisch M, Karl H, Wolisz A, Zhong LC, Rabaey J (2003) Distributed algorithms for transmission power control in wireless sensor networks. In: 2003 IEEE Wireless Communications and Networking, WCNC 2003, New Orleans, vol 1, pp 558–563
15. Boukerche A, Oliveira HA, Nakamura EF, Loureiro AAF (2007) Localization systems for wireless sensor networks. IEEE Wirel Commun 14:6–12

16. Efrat A, Har-Peled S, Mitchell JSB (2005) Approximation algorithms for two optimal location problems in sensor networks. In: 2nd international conference on broadband networks, BroadNets 2005, Boston, vol 1, pp 714–723
17. Basagni S, Carosi A, Melachrinoudis E, Petrioli C, Wang ZM (2008) Controlled sink mobility for prolonging wireless sensor networks lifetime. Wirel Netw 14(6):831–858
18. Wolsey LA (1998) Integer programming. Wiley-Interscience, New York
19. Nakamura F, Quintão F, Menezes G, Mateus G (2005) An optimal node scheduling for flat wireless sensor networks. Lect Notes Comput Sci 3420:475–482
20. Jarray F (2013) A lagrangean-based heuristics for the target covering problem in wireless sensor network. Appl Math Model 37:6780–6785
21. Castaño F, Bourreau E, Velasco N, Rossi A, Sevaux M (2015) Exact approaches for lifetime maximization in connectivity constrained wireless multi-role sensor networks. Eur J Oper Res 241(1):28–38
22. Lee J-H, Moon I (2014) Modeling and optimization of energy efficient routing in wireless sensor networks. Appl Math Model 38(7–8):2280–2289
23. Valle CA, Martinez LC, da Cunha AS, Mateus GR (2011) Heuristic and exact algorithms for a min–max selective vehicle routing problem. Comput Oper Res 38(7):1054–1065
24. Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
25. Resende MG, Ribeiro CC (2005) Grasp with path-relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers. Operations research/computer science interfaces series, vol 32. Springer, New York, pp 29–63
26. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
27. Lourenço HR, Martin OC, Stutzle T (2010) Iterated local search: framework and applications. In: Glover F, Kochenberger G, Hillier F (eds) Handbook of metaheuristics. International series in operations research and management science, vol 57, ch 12, 2nd edn. Springer, New York, pp 363–398
28. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549. Applications of integer programming
29. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130(3):449–467
30. Hansen P, Mladenović N (2001) Variable neighborhood search. In: Pardalos P, Resende M (eds) Handbook of applied optimization. Oxford University Press, New York
31. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, 2nd edn. 1992
32. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F, Dasgupta D, Moscato P, Poli R, Price KV (eds) New ideas in optimization. McGraw-Hill Ltd., Maidenhead, pp 219–234
33. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2–3):243–278
34. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings IEEE international conference on neural networks, Perth, vol 4, pp 1942–1948
35. Glover F, Kochenberger GA (2003) Handbook of metaheuristics. Kluwer Academic Publishers, Boston
36. Nieberg T (2006) Independent and dominating sets in wireless communication graphs. Ph.D. thesis, University of Twente
37. Ren H, Meng M-H, Chen X (2006) Investigating network optimization approaches in wireless sensor networks. In: International conference on intelligent robots and systems, Deajeon, pp 2015–2021
38. Li J (2008) Optimization problems in wireless sensor and passive optical networks. Ph.D. thesis, The University of Melbourne

39. Suomela J (2009) Optimisation problems in wireless sensor networks: local algorithms and local graphs. Ph.D. thesis, University of Helsinki
40. Gogu A, Nace D, Dilo A, Meratnia N (2012) Review of optimization problems in wireless sensor networks. In: Hamilton Ortiz J (ed) Telecommunications networks – current status and future trends. InTech, Rijeka, pp 153–180
41. Molina G, Alba E (2011) Location discovery in wireless sensor networks using metaheuristics. Appl Soft Comput 11(1):1223–1240
42. Shahrokhzadeh M, Haghighat AT, Mahmoudi F, Shahrokhzadeh B (2011) A heuristic method for wireless sensor network localization. In: Procedia computer science, vol 5. The 2nd international conference on ambient systems, networks and technologies (ANT-2011)/the 8th international conference on mobile web information systems (MobiWIS 2011). Elsevier, Niagara Falls, pp 812–819
43. Brazil M, Ras CJ, Thomas DA (2009) Deterministic deployment of wireless sensor networks. In: Proceedings of the world congress on engineering 2009, London, vol 1, p 863
44. Sasikumar P, Vasudevan SK, Ramesh M (2010) Heuristic approaches with energy management for node placement in wireless sensor networks. Int J Comput Appl 1(37):34–736
45. Laszka A, Buttyán L, Szeszlér D (2013) Designing robust network topologies for wireless sensor networks in adversarial environments. Pervasive Mob Comput 9(4):546–563
46. Alba E, Molina G (2008) Optimal wireless sensor network layout with metaheuristics: solving a large scale instance. In: Lirkov I, Margenov S, Waśniewski J (eds) Large-scale scientific computing. Lecture notes in computer science, vol 4818. Springer, Berlin/Heidelberg, pp 527–535
47. Deschinkel K (2011) A column generation based heuristic for maximum lifetime coverage in wireless sensor networks. In: SENSORCOMM: the fifth international conference on sensor technologies and applications, Nice, pp 1–6
48. Cardei M, Wu J, Lu M, Pervaiz M (2005) Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. In: IEEE international conference on wireless and mobile computing, networking and communications, Montreal, vol 3, pp 438–445
49. Delicato F, Protti F, Pirmez L, de Rezende JF (2006) An efficient heuristic for selecting active nodes in wireless sensor networks. Comput Netw 50(18):3701–3720
50. Karasabun E, Korpeoglu I, Aykanat C (2013) Active node determination for correlated data gathering in wireless sensor networks. Comput Netw 57(5):1124–1138
51. Santos AC, Bendali F, Mailfert J, Duhamel C, Hou KM (2009) Heuristics for designing energy-efficient wireless sensor network topologies. J Netw 4(6):436–444
52. Matos VO, Arroyo JEC, dos Santos AG, Goncalves LB (2012) An energy-efficient clustering algorithm for wireless sensor networks. Int J Comput Sci Netw Secu 12(10):6–15
53. Resende MGC, Werneck RF (2004) A hybrid heuristic for the p-median problem. J Heuristics 10:59–88
54. Heinzelman WB, Chandrakasan AP, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd annual Hawaii international conference on system sciences, Hawaii, vol 2, p 10
55. Heinzelman WB, Chandrakasan AP, Balakrishnan H (2002) An application-specific protocol architecture for wireless microsensor networks. IEEE Trans Wirel Commun 1: 660–670
56. Albath J, Thakur M, Madria S (2010) Energy constrained dominating set for clustering in wireless sensor networks. In: 24th IEEE international conference on advanced information networking and applications (AINA), Perth, pp 812–819
57. Ferentinos KP, Tsiligiridis TA (2007) Adaptive design optimization of wireless sensor networks using genetic algorithms. Comput Netw 51(4):1031–1051
58. Kuila P, Gupta SK, Jana PK (2013) A novel evolutionary approach for load balanced clustering problem for wireless sensor networks. Swarm Evol Comput 12:48–56
59. Ting C-K, Liao C-C (2010) A memetic algorithm for extending wireless sensor network lifetime. Inform Sci 180(24):4818–4833

60. Ding L, Gao X, Wu W, Lee W, Zhu X, Du D-Z (2011) An exact algorithm for minimum cds with shortest path constraint in wireless networks. Optim Lett 5(2):297–306
61. Gandham S, Dawande M, Prakash R, Venkatesan S (2003) Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In: Proceedings of IEEE globecom 2003, San Francisco, vol 1, pp 377–381
62. Behdani B, Yun YS, Smith JC, Xia Y (2012) Decomposition algorithms for maximizing the lifetime of wireless sensor networks with mobile sinks. Comput Oper Res 39(5):1054–1061
63. Basagni S, Carosi A, Petrioli C, Phillips CA (2009) Heuristics for lifetime maximization in wireless sensor networks with multiple mobile sinks. In: IEEE international conference on communications, ICC'09, Dresden, pp 1–6
64. Christofides N (1976) Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report 388, Carnegie-Mellon University, Graduate School of Industrial Administration
65. Xing G, Wang T, Jia W, Li M (2008) Rendezvous design algorithms for wireless sensor networks with a mobile base station. In: Proceedings of the 9th ACM international symposium on mobile Ad Hoc networking and computing, MobiHoc'08. ACM, New York, pp 231–240
66. Aioffi WM, Valle CA, Mateus GR, da Cunha AS (2011) Balancing message delivery latency and network lifetime through an integrated model for clustering and routing in wireless sensor networks. Comput Netw 55(13):2803–2820
67. Üster H, Lin H (2011) Integrated topology control and routing in wireless sensor networks for prolonged network lifetime. Ad Hoc Netw 9(5):835–851
68. Güney E, Aras N, Altinel IK, Ersoy C (2012) Efficient solution techniques for the integrated coverage, sink location and routing problem in wireless sensor networks. Comput Oper Res 39:1530–1539
69. Türkogulları YB, Aras N, Altınel IK, Ersoy C (2010) A column generation based heuristic for sensor placement, activity scheduling and data routing in wireless sensor networks. Eur J Oper Res 207(2):1014–1026
70. Raiconi A, Gentili M (2011) Exact and metaheuristic approaches to extend lifetime and maintain connectivity in wireless sensors networks. In: Proceedings of the Network optimization: 5th international conference, INOC 2011, Hamburg, 13–16 June 2011. Springer, Berlin/Heidelberg, pp 607–619
71. Gentili M, Raiconi A (2013) α-coverage to extend network lifetime on wireless sensor networks. Optim Lett 7(1):157–172
72. Carrabs F, Cerulli R, D'Ambrosio C, Raiconi A (2015) Exact and heuristic approaches for the maximum lifetime problem in sensor networks with coverage and connectivity constraints. Technical report, University of Salerno
73. Blum J, Ding M, Thaeler A, Cheng X (2005) Connected dominating set in sensor networks and MANETs. Springer, Boston, pp 329–369
74. Gendron B, Lucena A, da Cunha AS, Simonetti L (2014) Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. INFORMS J Comput 26(4):645–657
75. Buchanan A, Sung JS, Butenko S, Pasiliao (2015) An integer programming approach for fault-tolerant connected dominating sets. INFORMS J Comput 27(1):178–188
76. Türkogulları YB, Aras N, Altınel IK, Ersoy C (2010) An efficient heuristic for placement, scheduling and routing in wireless sensor networks. Ad Hoc Netw 8(6):654–667
77. Güney E, Aras N, Altınel IK, Ersoy C (2010) Efficient integer programming formulations for optimum sink location and routing in heterogeneous wireless sensor networks. Comput Netw 54(11):1805–1822
78. Rajasekaran S, Pardalos P, Hsu DF (2000) Mobile networks and computing. DIMACS – series in discrete mathematics and theoretical computer science, vol 52. American Mathematical Society, Providence
79. Oliveira CA, Pardalos PM (2011) Mathematical aspects of network routing optimization. Volume 53 of 1931–6828. Springer, New York

80. Resende MG, Pardalos PM (2006) Handbook of optimization in telecommunications. Springer, New York. No. 1 in 78-0-387-30662-9
81. Pardalos PM, Ye Y, Boginski VL, Commander CW (2012) Sensors: theory, algorithms, and applications. Volume 61 of 1931–6828. Springer, New York
82. Aioffi WM, Mateus GR, Quintao FP (2007) Optimization issues and algorithms for wireless sensor networks with mobile sink. In: International network optimization conference, Spa
83. Lin H, Uster H (2014) Exact and heuristic algorithms for data-gathering cluster-based wireless sensor network design problem. IEEE/ACM Trans Netw 22:903–916

Particle Swarm Optimization for the Vehicle Routing Problem: A Survey and a Comparative Analysis

40

Yannis Marinakis, Magdalene Marinaki, and Athanasios Migdalas

Contents

Introduction 1164
Particle Swarm Optimization 1166
Application of Particle Swarm Optimization Algorithm in Vehicle Routing Problems 1170
Main Variants of VRP Solved by PSO 1171
Different PSO Variants 1175
Analysis of the Algorithms 1182
Conclusions 1186
References 1186

Abstract

In the last few years, a number of books and survey papers devoted to the vehicle routing problem (VRP) or to its variants or to the methods used for the solution of one or more variants of the VRP have been published. Also, in these years, the field of swarm intelligence algorithms has had a significant growth. One of the most important swarm intelligence algorithms is the particle swarm optimization (PSO). Although the particle swarm optimization was first published in 1995, it took around 10 years in order researchers to publish papers using a PSO algorithm for the solution of variants of the VRP. However, in the last 10 years, many journal papers, conference papers, and book chapters have been published

Y. Marinakis (✉) · M. Marinaki
School of Production Engineering and Management, Technical University of Crete, Chania, Greece
e-mail: marinakis@ergasya.tuc.gr; magda@dssl.tuc.gr

A. Migdalas
Industrial Logistics, Luleå Technical University, Luleå, Sweden

Department of Civil Engineering, Aristotle University of Thessalonike, Thessalonike, Greece
e-mail: athmig@ltu.se; samig@civil.auth.gr

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_42

where a variant of VRP is solved using a PSO algorithm. Thus, it is significant to present a survey paper where a review and brief analysis of the most important of these papers will be given. This is the main focus of this chapter.

Keywords
Vehicle routing problem · Particle swarm optimization

Introduction

The vehicle routing problem is one of the most important problems in the field of supply chain management, of logistics, of combinatorial optimization, of transportation, and, in general, of operational research. The interest in this problem has been recently increased both from theoretical and practical aspect. There are a number of reasons for this growth. From the practical point of view, the problem is one of the most important problems in the supply chain management, and, thus, the finding of the optimal set of routes will help the decision makers to reduce the cost of the supply chain and to increase the profit. Also, in the formulation of the problem, the managers could simulate the complete network and add any of the constraints concerning the customers, the vehicles, the routes, and, also, the traffic conditions of the network and the energy consumption of the vehicles. Thus, someone could solve a realistic problem and find a near optimal set of solutions.

The reason that usually a near optimal set of solutions is found is that the problem from its origin (it was first introduced by Dantzig and Ramser in 1959 [46]) was proved to be NP-hard problem even in its simpler version, the capacitated vehicle routing problem, where the only constraint that was taken into account was the capacity of the vehicles (and later the maximum tour length constraint was added). Thus, it is impossible in real-life applications to find an optimal solution. For this reason, a number of heuristic, metaheuristic (mainly), evolutionary, and nature-inspired approaches have been proposed for the solution of the VRP and its variants. Also, exact algorithms have been proposed in order to solve the problem. They are, mainly, used for the solution of the simplest vehicle routing problems (the problems with as few as possible constraints) and for a small number of nodes.

From the theoretical point of view, there are a huge number of researchers that deal with the solution of a variant (or more variants) of the problem. These variants of the problem focus on a specific constraint, and the researchers are trying to find an algorithm that gives new best solutions in a specific set of benchmark instances in short computational time with as less as possible parameters in order to give a more general algorithm. Thus, a new researcher that he/she would like to focus to a specific variant of a vehicle routing problem, he/she could find a large number of very good papers focusing to VRP or to its variants and papers focusing to the methods that he/she would like to implement for the solution of the problem. The most important and well-studied variants of the vehicle routing problem are the capacitated vehicle routing problem [39,40,55,98,100], the vehicle routing problem

with time windows [43, 49, 50, 159, 160], the open vehicle routing problem [152], the vehicle routing problem with simultaneous pickup and deliveries [123, 170], the vehicle routing problem with backhauls [29, 30, 171], the multidepot vehicle routing problem [32, 99, 122], the stochastic vehicle routing problem [62, 77, 142, 162], the dynamic vehicle routing problem [61, 142, 145, 146], the periodic vehicle routing problem [5], the split delivery vehicle routing problem [6], the heterogeneous fleet vehicle routing problem [64], the asymmetric vehicle routing problem [178], the vehicle routing problem with two- or three-dimensional loading constraints [21, 66], etc. Nowadays, more complicated problems have been published where more than one from the previously mentioned problems are combined in order to create a new more challenging problem. These problems are denoted from some researchers as rich vehicle routing problems [96]. However, a number of other problems have been introduced in the last years in order to cope with new needs that arise from new realistic situations of the life, like the cumulative capacitated vehicle routing problem [127], the evacuation vehicle routing problem [189], the green vehicle routing problem [103], the pollution routing problem [15], etc. Finally, there are a number of combined problems that need a solution of a vehicle routing problem, like the location routing problem [97, 126, 144], the inventory routing problem [26, 27, 54], the location inventory routing problem [78], the production routing problem [1], the production inventory distribution problem [13], and the ship routing problem [37, 38, 150, 151]. Also, a number of publications with a real case application have been realized where the authors simulate the realistic scenario, formulate the problem (or use an existing formulation) using a combination of the existing variants or introducing a new one if it is possible, and use an existing algorithm (or propose a new one or a suitable modified variant of an existing one) to solve it.

Thus, in the last years, the EURO Working Group on Vehicle Routing and Logistics Optimization, the VeRoLog, was created (www.verolog.eu); two different series of conferences devoted to the vehicle routing problem variants and applications have been introduced, the one is an annual conference which is organized from the VeRoLog working group since 2012 and the second is a triennial workshop on freight transportation and logistics denoted as Odysseus started in Crete 2000. A number of books devoted to the vehicle routing problem have been published. The first one was published in 1988, and it was a very inspiring book for any new researcher that would like to study the vehicle routing problem or just to know about the vehicle routing problem [70]. Twenty years later, one of the two authors (Bruce Golden) of the first published book [70] published a new book [72] that summarizes the work performed in these 20 years and gives some new directions for future research. Before the publication of this book, another very successful and inspiring book was published by Paolo Toth and Daniele Vigo [170] which is the most widely read and most cited book in the field. The success of this book led the same team of authors to publish in the end of 2014 the second edition of the book [172] which covers all the field of the vehicle routing problem with a more complete rewriting of some of the chapters of the previous edition of the book and with the new directions that have been created based on the new

challenges in the field. A volume of the *Handbooks in Operations Research and Management Science* was devoted to vehicle routing problems and to more general applications, like the arc routing problem [9], and it was published in [10]. Finally, two other books devoted to the vehicle routing problem have been published in [138, 192]. Also, a large number of surveys have been published, initially, devoted to general vehicle routing problems and, nowadays, to variants of the vehicle routing problem or to methods that are applied for the solution of the vehicle routing problem or of its variants. Some of these survey papers can be found in [7, 8, 17, 19, 20, 23, 24, 28, 52, 55, 60, 63, 65, 71, 81, 96, 98, 100, 103, 113, 140, 165, 177].

Thus, in this chapter, as a general review for the vehicle routing problem cannot be restricted in one chapter, we decided to focus in a specific algorithm, the particle swarm optimization (PSO) algorithm, and in the application of this algorithm for the solution of the vehicle routing problem and its main variants. This is the first survey chapter, at least to our knowledge, that is devoted to this method for the VRP. In the following sections, initially, a brief presentation of the particle swarm optimization algorithm will be given, and, then, we will present the variants of the vehicle routing problem in which a particle swarm optimization algorithm has been applied for their solution.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a very popular global optimization method that was originally proposed by Kennedy and Eberhart as a simulation of the social behavior of social organisms such as bird flocking and fish schooling [86]. PSO uses the physical movements of the individuals in the swarm. Complete surveys, in the time that they are published, for the particle swarm optimization can be found in [11, 12, 41, 141]. Nowadays, a complete review for the particle swarm optimization is very difficult to be performed as the range of variants and of applications of PSO covers the whole field of optimization, and, thus, only surveys in a specific subject like the one presented in this chapter (application of PSO algorithm in vehicle routing problems) can be presented without the length of the paper to increase dramatically.

In general, in a PSO algorithm, a set of solutions are used where each solution is denoted as a particle. These solutions create the swarm. In many algorithms, more than one swarms exist. Initially, the solutions are randomly initialized in the solution space. Two are the main vectors that describe a particle, the position vector (x_{ij}, where i denotes the particle ($i = 1, \cdots, N$, N is the swarm size) and j denotes the corresponding dimension of the particle ($j = 1, \cdots, d$, d is the dimension of the problem)) and the velocities vector (v_{ij}). The performance of each particle is evaluated on the predefined fitness function ($f(x)$).

Thus, each particle is randomly placed in the d-dimensional space as a candidate solution. One very simple and effective way to initialize the particles was given in

[53] where the minimum value of the solution space is denoted as $x_{\min,j}$ and the maximum value is denoted as $x_{\max,j}$. Then, the values of the solution vector for each particle are calculated from the following equation:

$$x_{ij}(0) = x_{\min,j} + r_j(x_{\max,j} - x_{\min,j}) \tag{1}$$

where r_j is a random number between $(0, 1)$.

The velocity of the i-th particle v_{ij} is defined as the change of its position. The algorithm completes the optimization through following the personal best solution of each particle and the global best value of the whole swarm. Thus, in each iteration, except of the current position, another vector is used which is the personal best position of each particle through the iterations. The best member of the personal best position vector is the global best position of the whole swarm. Each particle adjusts its trajectory toward its own previous best position and the global best position, namely, $pbest_{ij}$ and $gbest_j$, respectively. The velocities and positions of particles are updated using the following equations [86]:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) + c_2 rand_2(gbest_j - x_{ij}(t)) \tag{2}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3}$$

where t is the iteration counter, c_1 and c_2 are the acceleration coefficients, and $rand_1$ and $rand_2$ are two random variables in the interval (0, 1). The values of c_1 and c_2 could be constant, or they could be adapted during the iterations. Usually in the most published papers concerning an application of PSO in a continuous or discrete problem, these values were set equal to 2. However, there is a possibility to take different values or to adjust their values during the iterations, for example, using the following equations [53]:

$$c_1 = c_{1,\min} + \frac{c_{1,\max} - c_{1,\min}}{iter_{\max}} \times t \tag{4}$$

$$c_2 = c_{2,\min} + \frac{c_{2,\max} - c_{2,\min}}{iter_{\max}} \times t \tag{5}$$

where $\text{iter}_{\max}$ is the maximum number of iterations and $c_{1,\min}$, $c_{1,\max}$, $c_{2,\min}$, and $c_{2,\max}$ are the minimum and maximum values that c_1 and c_2 can take, respectively. In the beginning of the procedure, the values of c_1 and c_2 are small and, then, are increasing until they reach their maximum values. By doing this, on the first iterations, there is a great freedom of movement in the particle solution space in order to find the optimum quickly.

A number of different velocity equations have been proposed during the last years. Nowadays, other researchers use one of the velocity equations proposed in the past that have been proved to perform well (mainly, the inertia equation or the constriction equation), or, less frequently, they propose a new one which either is a

variant of a previously published equation or simulates something from the nature in which the specific researcher focuses in his/her study. The most important and known velocities equations are the following:

- Inertia particle swarm optimization (IPSO) [157]:

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1\, rand_1(pbest_{ij} - x_{ij}(t)) \\ &\quad + c_2 rand_2(gbest_j - x_{ij}(t)) \end{aligned} \tag{6}$$

  The difference between this variant and the one presented on Eq. (2) is the use of the inertia weight w. The inertia weight w is adapted during the iterations and is given by the following equation:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{\text{iter}_{\max}} \times t \tag{7}$$

  where $w_{\max}$ and $w_{\min}$ are the maximum and minimum values of the inertia weight.
- Constriction particle swarm optimization [42]:

$$\begin{aligned} v_{ij}(t+1) &= \chi(v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) \\ &\quad + c_2 rand_2(gbest_j - x_{ij}(t))) \end{aligned} \tag{8}$$

  where the constriction factor, χ, is used:

$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|} \text{ and } c = c_1 + c_2, c > 4 \tag{9}$$

- Another version of the constriction particle swarm optimization [53]:

$$\begin{aligned} v_{ij}(t+1) &= \chi(v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) \\ &\quad + c_2 rand_2(gbest_j - x_{ij}(t))) \end{aligned} \tag{10}$$

  The only difference from the previous version is the use of the parameter k in the constriction factor [53]:

$$\chi = \frac{2k}{|2 - c - \sqrt{c^2 - 4c}|} \text{ and } c = c_1 + c_2, c > 4 \tag{11}$$

- Clerc and Kennedy [42] proposed a simpler form of the constriction factor, the **condensed form**, for the calculation of the velocities of the particles:

$$v_{ij}(t+1) = \chi(v_{ij}(t) + c(p_{mj} - x_{ij}(t))) \tag{12}$$

where

$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|} \text{ and } c = c_1 + c_2, c > 4 \tag{13}$$

and

$$p_{mj} = \frac{c_1 pbest_{ij} + c_2 gbest_j}{c}. \tag{14}$$

- Cognition-only particle swarm optimization [85]
  In this variant, the velocity equation is given by:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) \tag{15}$$

  The social factor of velocities equation is not used.
- Social-only particle swarm optimization [85].
  In this variant, the velocity equation is given by:

$$v_{ij}(t+1) = v_{ij}(t) + c_2 rand_2(gbest_j - x_{ij}(t)) \tag{16}$$

  The cognition factor of velocities equation is not used.
- Local neighborhood topology particle swarm optimization [53]:

$$\begin{aligned} v_{ij}(t+1) = v_{ij}(t) &+ c_1 rand_1(pbest_{ij} - x_{ij}(t)) \\ &+ c_2 rand_2(lbest_{ij} - x_{ij}(t)) \end{aligned} \tag{17}$$

  where the term of $gbest$ in the previous algorithms has been replaced with the term $lbest$, which means that instead of a global best population, a local best population is used.

$$lbest_{ij} \in N_i | f(lbest_{ij}) = \min f(x_{ij}), \forall x \in N_i \tag{18}$$

  The neighbor N_i is defined by [53]:

$$\begin{aligned} N_i = &\ pbest_{i-n_{N_i}}(t), \ldots, pbest_{i-1}(t), pbest_i(t), \\ &pbest_{i+1}(t), \ldots, pbest_{i+n_{N_i}}(t) \end{aligned} \tag{19}$$

  A particle's best position ($pbest_{ij}$) in a swarm is calculated from the equation:

$$pbest_{ij} = \begin{cases} x_{ij}(t+1), & \text{if } f(x_{ij}(t+1)) < f(x_{ij}(t)) \\ pbest_{ij}, & \text{otherwise} \end{cases} \tag{20}$$

The optimal position of the whole swarm is calculated by the equation:

$$gbest_j \in \{pbest_{1j}, pbest_{2j}, \cdots, pbest_{Nj}\} | f(gbest_j) = \min\{f(pbest_{1j}), f(pbest_{2j}), \cdots, f(pbest_{Nj})\} \quad (21)$$

Most applications of PSO have concentrated on the optimization in continuous space, while a lot of work has been done to the discrete optimization problems beginning from the next two papers [87, 157]. In the following, a pseudocode of the particle swarm optimization algorithm is presented.

algorithm `Particle Swarm Optimization`
Initialization
Select the number of swarms
Select the number of particles for each swarm
Initialization of the position and velocity of each particle
Calculation of the initial cost function (fitness function) value of each particle
Keep global best particle (solution) of the whole swarm
Keep personal best of each particle
Main Phase
Do until the maximum number of iterations has not been reached:
    Calculate the velocity of each particle
    Calculate the new position of each particle
    Evaluate the new fitness function of each particle
    Update the best solution of each particle
    Update the best particle of the whole swarm
Enddo
Return the best particle (the global best solution)

Application of Particle Swarm Optimization Algorithm in Vehicle Routing Problems

In this chapter, the papers in which a variant of the particle swarm optimization algorithm is applied for the solution of a variant of the vehicle routing problem are presented. In this section, an analysis is given on how these papers have been selected. As there are a large number of papers in the literature, we will in the most important of them. A very difficult task is to define which of the papers are the "most important" ones. We use a number of criteria to show the impact of each paper in the research community. The first criterion is the impact factor of the journal or the importance of the conference in which the paper has been published. The second criterion is the number of citations (based on Google Scholar) that each paper has received through the years. Papers that have been published in the current year are not expected to have a large number of citations. However, some of them based on their computational results are included in the analysis. Finally, the third

criterion is the computational results given in each paper. There are a number of papers that have demonstrated their PSO variant giving results in one or in a small set of instances. These papers are not analyzed as the effectiveness of their variant is not sufficiently proved. Also, there are a number of papers that, although they have been published in very good journals, they present a PSO variant for the solution of a real-life application and they do not give a comparative analysis with other algorithms from the literature in benchmark instances; thus, it is very difficult to show the effectiveness of the PSO variant. These kinds of papers are not included in the analysis. Finally, there are a number of papers that they present a novel PSO procedure, they apply the method in classic sets of benchmark instances, and their computational results are competitive with the results of the most effective algorithms from the literature. This last category of papers is those that are analyzed in this chapter.

Main Variants of VRP Solved by PSO

The main variants of the vehicle routing problem that a particle swarm optimization algorithm has been applied are the following.

Capacitated Vehicle Routing Problem
Abbreviation: ***CVRP*** *Definition:* Each vehicle must start and finish its tour at the depot. Capacity constraints of the vehicles. Maximum tour duration of each route. Not split deliveries allowed. Customers have only demands and service time [19, 20, 72, 170].

Open Vehicle Routing Problem
Abbreviation: ***OVRP*** *Definition:* Same constraints as in the CVRP except that the vehicles do not return in the depot after the service of the customers [152].

Vehicle Routing Problem with Time Windows
Abbreviation: ***VRPTW*** *Definition:* Same constraints as in the CVRP and in addition each customer must be serviced within a specific time window. Vehicles and depots may, also, have a time window. Minimization, initially, of the number of routes and, then, minimization of the total traveled distance [70, 72, 138, 158, 159, 170].

Vehicle Routing Problem with Simultaneously Pickup and Delivery
Abbreviation: ***VRPSPD*** *Definition:* Same constraints as in the CVRP. The customers require not only delivery of products but, also, a simultaneous pick up of products from them. It is assumed that the delivery is performed before the pickup and that the vehicle load should never be negative or larger than the vehicle capacity [70, 72, 138, 170].

Vehicle Routing Problem with Stochastic Demands
Abbreviation: ***VRPSD*** *Definition:* A vehicle with finite capacity leaves from the depot with full load and has to serve a set of customers whose demands are known only when the vehicle arrives to them. A route begins from the depot and visits each customer exactly once and returns to the depot. This is called an a priori

tour, and it can be seen as a template for the visiting sequence of all customers [18, 62, 162].

Dynamic Vehicle Routing Problem

Abbreviation: ***DVRP*** *Definition:* In dynamic problems, part or all of the input is unknown and is revealed dynamically during the design or execution of the routes [140].

Multidepot Vehicle Routing Problem

Abbreviation: ***MDVRP*** *Definition:* More than one depots are used for the customers' service. There is a possibility for each customer to be clustered and served from only one depot, or the customers may be served from any of the depots using the available fleet of vehicles [122, 149].

Vehicle Routing Problem with Stochastic Travel Times

Abbreviation: ***VRPSTT*** *Definition:* In this variant, the travel time between every pair of nodes is a random variable related to traffic jam, road maintenance, or weather conditions. The stochasticity appears in the arcs (road) of the network due to unexpected conditions [196].

Vehicle Routing Problem with Fuzzy Demands

Abbreviation: ***VRPFD*** *Definition:* In this variant, the demands could be represented as fuzzy variables.

Periodic Vehicle Routing Problem

Abbreviation: ***PVRP*** *Definition:* In this problem, vehicle routes must be constructed over multiple days where during each day within the planning period, a fleet of capacitated vehicles travels along routes that begin and end at a single depot [56]. The objective of the PVRP is to find a set of tours for each vehicle that minimizes total travel cost while satisfying the constraints of the problem [56].

Location Routing Problem

Abbreviation: ***LRP*** *Definition:* In this variant, the optimal location to be used for the storage facilities have to be decided. From these locations, the vehicles will begin their routes, in a way that the total cost of the routing (distance, fuel, time, etc.) and facility location (running costs, rent or property cost, etc.) will be the minimum. At the same time, the optimal routes for the vehicles have to be found in order to satisfy the demand of the customers [47, 119, 126, 144].

Location Routing Problem with Stochastic Demands

Abbreviation: ***LRPSD*** *Definition:* In this variant, the same constraints as in the case of the location routing problem hold except that the demands of the customers have stochastic and not deterministic values.

Team Orienteering Problem

Abbreviation: ***TOP*** *Definition:* This is a variant of the vehicle routing problem with profits or pricing. In this problem, a set of locations is given, each one with a score. The goal is to determine a fixed number of routes, limited in length, that visit some locations and maximize the sum of the collected scores [174]. The objective of the TOP is to construct a certain number of paths starting at an origin and ending at a destination that maximize the total profit without violating predefined limits [154].

Production Routing Problem
Abbreviation: ***PRP*** *Definition:* This variant is a hybridization of VRP with production problems. These problems are very complicated problems as they include, among others, decisions concerning the number of workers to be hired, the quantities of products to be produced using various strategies, the amount of inventories to be maintained, the assignment of the customers in different manufacturing plans or depots, and, finally, the design of the routes in order to satisfy the customers' demands.

Ship Routing Problem
Abbreviation: SRP Definition: In this variant, the optimum routing and scheduling of different types of ships are calculated [37, 38, 48, 150, 151].

Vehicle Routing Problem with heterogeneous fleet
Abbreviation: ***HVRP*** *Definition:* In this variant, instead of an homogeneous fleet of vehicles, the company uses a set of vehicles with different capacities, and, thus, the fleet of vehicles is heterogeneous.

Inventory Routing Problem
Abbreviation: ***IRP*** *Definition:* In this variant of the vehicle routing problem, a simultaneous allocation of inventories and decision of routing schedules is realized. The objective of the IRP is to minimize the total cost (the distribution and inventory costs of retailers) [26, 27].

Vehicle Routing Problem with Uncertain Demands
Abbreviation: ***VRPUD*** *Definition:* In this variant, the demand is uncertain with unknown distribution.

Vehicle Routing and Scheduling Problem
Abbreviation: ***VRSP*** *Definition:* In general, the goal of VRSP is to determine the set of trips that a vehicle will make during the day in order to reduce the transportation costs.

Other variants of VRP, combinations of VRP with other problems, or real-life applications of VRP that have been solved using a variant of a particle swarm optimization algorithm are the distance constraint vehicle routing problem [169], the multiple destination routing problem (MVRP) [193], the multivehicle assignment problem (MVAP) [51], the production routing problem [1], the berth allocation problem [168], the disaster relief logistics [22], the garbage collection system [95], the integrated production and distribution [175], the urban transit routing problem [84], the production and pollution routing problem [94], the reducing vehicle emissions and fuel consumption [133], the pedestrian-vehicle mixed evacuation (P-VMEP) [201], the time-dependent vehicle routing problem (TDVRP) [130], the emergency vehicle scheduling problem [57], the emergency logistics (EL) [189, 197], the evacuation vehicle routing problem (EvVRP) [191], the environmental vehicle routing problem (EnVRP) [59], the integrated production scheduling and vehicle routing [35], and the reverse logistics (RL) [134].

Table 1 presents the number of papers (journal papers, international conference papers, and book chapters, denoted as Journal, Conf, and BC in the table, respectively) in which a variant of the particle swarm optimization algorithm has been used

Table 1 Number of papers using particle swarm optimization in each variant of the vehicle routing problem

Variant	Journal	Paper	Conf and BC	Paper
CVRP	8	[4, 33, 92, 109, 116, 147, 167, 176]	7	[79, 93, 101, 111, 156, 163, 183]
VRPSPD	3	[3, 36, 69]	4	[68, 164, 184, 195]
LRP	2	[104, 108]	2	[14, 135]
HVRP	2	[16, 188]		
VRPTW	2	[2, 73]	9	[25, 31, 58, 80, 105, 118, 124, 182, 200]
VRPUD			2	[34, 186]
TOP	2	[125, 154]	3	[44, 45, 153]
MVAP			1	[51]
DVRP	2	[91, 132]	3	[89, 90, 131]
OVRP	2	[120, 128]	4	[110, 139, 180, 199]
VRPSD	1	[117]	3	[112, 137, 198]
PVRP	3	[129, 148, 166]		
RL			1	[134]
VRPFD	1	[185]	1	[136]
VRPSTT			1	[155]
MDVRP	1	[82]	4	[161, 179, 181, 194]
DCVRP	1	[169]		
VRSP	1	[173]		
EvVRP	1	[191]	2	[189, 190]
MVRP			1	[193]
EL			1	[197]
LRPSD	1	[107]		
EnVRP			1	[59]
IRP	1	[106]		
TDVRP	1	[130]		
PRP	1	[1]		
SRP	1	[48]		

for the solution of a variant of the vehicle routing problem. As it was expected, most of the papers concern an application of a PSO variant to the capacitated vehicle routing problem with eight journal papers published. There is no other variant of VRP with more than three journal papers. However, in the vehicle routing problem with time windows, there are a large number of conference papers and book chapters (nine in total). As we can see, a variant of PSO algorithm has been applied in 27 different variants of vehicle routing problem. In total, there are 37 journal papers and 52 international conference papers and book chapters with a PSO variant in a vehicle routing problem.

Although when the particle swarm optimization algorithm was initially proposed, there were no any applications of the algorithm for the solution of the vehicle routing

Table 2 Journals that have published at least one PSO algorithm for the solution of a vehicle routing problem variant

Journal	Abbreviation	I.F.	Nr	TNrCit
Transportation Science	TS	3.295	1	60
Networks and Spatial Economics	NETS	3.250	1	23
Expert Systems with Application	ESWA	2.981	3	295
Applied Soft Computing	ASOC	2.857	6	201
Neurocomputing	NC	2.392	1	2
Engineering Applications of Artificial Intelligence	EAAI	2.368	1	130
Transportation Research Part E	TRE	2.279	2	57
IEEE Transactions on Systems, Man, and Cybernetics-Part C	IEEE SMC	2.171	1	74
Computers and Industrial Engineering	CIE	2.086	6	363
Journal of Intelligent Manufacturing	JIM	1.995	3	94
Computers and Operations Research	COR	1.988	2	297
Entropy	Entropy	1.743	1	20
Measurement	Measurement	1.742	1	12
International Journal of Advanced Manufacturing Technology	IJAMT	1.568	2	57
Annals of Operations Research	ANOR	1.406	1	33
Applied Mathematics and Computation	AMC	1.345	1	38
Applied Intelligence	APIN	1.215	1	8
Optimization Letters	OptL	1.019	1	5
Journal of Zhejiang University SCIENCE A	SCIENCE A	0.941	1	198
Memetic Computing	MEME	0.900	1	13
International Journal of Operational Research	IJOR	–	1	49
Procedia Engineering	PrE	–	1	8
Journal of Mathematical Modelling and Algorithms	JMMA	–	1	67
American Journal of Applied Sciences	AJAS	–	1	13
Transportation Research Journal	TRJ	–	1	4

problem; in the last years, a number of researchers have solved a variant of VRP using a PSO algorithm. In Table 2 journals that have published at least one paper with this topic are presented. Journals are sorted based on their impact factor for 2015 (denoted as I.F. in the table) that it is presented in column three. Finally, in columns 4 and 5, the number of papers (denoted as Nr in the table) that has been published in each journal and the total number of citations of all papers based on Google Scholar on 30 May 2017 (denoted as TNrCit in the table) are presented.

Different PSO Variants

As it is mentioned earlier, the main problem of the application of a PSO algorithm for the solution of a VRP was that the PSO algorithm is suitable for continuous

optimization problems and the VRP is a combinatorial optimization problem with a specific structure in the solution which needs in most cases a path representation in order to have an effective algorithm. Thus, there are a number of algorithms that use a 0/1 representation of the solution. However, these algorithms are not the most effective algorithms for the solution of this kind of problems. We mentioned that from the first years that PSO was proposed, there were implementations of PSO [157] suitable for discrete optimization problems but not for routing type problems. Thus, the first effective application of a PSO algorithm in VRP was in 2006 [33]. Since then a number of different algorithms have been proposed that are suitable for the solution of a VRP variant. Most of them are hybridized with a local search algorithm, as almost every evolutionary algorithm if it is applied for the solution of a VRP problem. In the following, we present and analyze the most important variants of PSO that have been applied for the solution of a VRP variant. The choice was performed using the same criteria that were mentioned earlier, especially the one concerning the effectiveness of the algorithms.

SR-PSO. One of the most used representations of PSO for the solution of a VRP variant denoted as SR-PSO (SR-1 PSO or SR-2 PSO) was proposed in [4] where two representations of the particles were used for the solution of the CVRP. In the first one, the particle consists of $n + 2m$ dimensions. The first n dimensions are related to customers (each customer is represented by one dimension). The last $2m$ dimensions are related to vehicles (each vehicle is represented by two dimensions as the reference point in the Cartesian map). The priority matrix of vehicles is constructed based on the relative distance between these points and customers' location [4]. A customer is prioritized to be served by the vehicle which has the closer distance. In the second representation, the particle consists of a $3m$-dimensional particle where it is decoded as a real number. All dimensions are related to vehicles; each vehicle is represented by three dimensions: two dimensions for the reference point and one dimension for the vehicle coverage radius [4]. The algorithms use a number of local search algorithms. A number of papers either from the same authors or from other research groups have used the same representations for solving a VRP variant.

The research group that proposed this representations have applied them for the solution of the vehicle routing problem with time windows [2], of the vehicle routing problem with simultaneously pickups and deliveries [3] (the most important publication of an application of PSO-based algorithm for solving a VRP based on the number of citations), of the multidepot VRP with pickup and deliveries [161] and [82], and of the location routing problem [104].

Other researchers that used the same encoding and decoding scheme for solving a VRP variant are the one from Hu et al. [76] that proposed a hybrid chaos-particle swarm optimization algorithm (HPSO) for solving VRPTW. In [121], a variant of the algorithm was proposed for the solution of a vehicle routing problem with uncertain demands. In [16], a variant of the algorithm was presented for a rich vehicle routing problem, the vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows (VRPHFMBTW).

Quantum PSO – QPSO. One of the first PSO implementations for the solution of the CVRP was the one published in [33]. In this PSO implementation, a quantum discrete PSO algorithm is used [187]. The algorithm is hybridized with a simulated annealing algorithm [88] for the improvement of the particles. They use a zero one encoding scheme where value equal to one means that the vehicle serves the customer and value equal to zero means that another vehicle will serve the customers. The dimension of the vector is equal to $N \times K$ where N is the number of nodes and K is the number of vehicles.

HybGenPSO. In [109], a hybridization of a genetic algorithm with a particle swarm optimization, algorithm for the solution of the CVRP was presented. In this memetic algorithm, the role of the genetic algorithm was to create the new generations, and the role of the PSO algorithm was to improve the population between two different generations. Thus, it was a memetic algorithm with the difference that usually in a memetic algorithm, each member of the population is evolved between two generations using a local search algorithm, while in this memetic algorithm, the population was evolved using a global search algorithm. This idea was inspired by the fact that in real life, each member of a population is evolved by reacting with other members of the population and not by itself. The algorithm uses the inertia equation for the velocities (Eq. 6) and was tested in the two classic sets of benchmark instances, and a number of best-known solutions were found.

HybPSO. In [116], a hybrid particle swarm optimization algorithm was used for the solution of the capacitated vehicle routing problem. The algorithm is an improved version of the algorithm used in [108] for the solution of the location routing problem, and it uses the same three algorithms in the hybridization phase, the multiple phase neighborhood search – greedy randomized adaptive search procedure (MPNS-GRASP), the expanding neighborhood search (ENS) [114], and the path relinking (PR) [67]. The first method is used for the creation of the initial solutions, the second one is used as a local search phase, and the third one is used for the movement of the particles. The difference of the algorithm proposed in [116] from the algorithm denoted as combinatorial neighborhood topology PSO (proposed in [111]) is that the first one uses a transformation between continuous and discrete values and vice versa based on the relative position indexing [102] and it, also, uses a procedure that does not affect the solution of each particle. In the combinatorial neighborhood topology PSO, no transformation is needed at all. The algorithm proposed in [116] uses the inertia equation for the velocities (Eq. 6). The algorithm was tested in a classic set of benchmark instances, and a number of best-known solutions were found.

In [110], a new version of the hybrid particle swarm optimization algorithm that was presented in [116] for the solution of the CVRP was presented where instead of using the expanding neighborhood search algorithm for the improvement of each particle separately, the variable neighborhood search algorithm was used [74]. Another difference is that the inertia velocities equation has been replaced by the constriction velocities equation (Eq. 8).

In [117], an improved version of the hybrid particle swarm optimization algorithm that was presented in [110] for the solution of the OVRP was presented where instead of using the variable neighborhood search algorithm for the improvement of each particle separately, a simpler local search algorithm based on 2-opt and 3-opt algorithms was used. Also, in the initialization phase of the algorithm, the particles start from random values, thus avoiding the use of a more sophisticated procedure. The reason that this simpler local search algorithm and the random initial values were used was that the calculation of the objective function of the VRPSDs is much more difficult and time-consuming than the calculation of the objective function of a simpler variant of VRP and, thus, the use of a simpler local search algorithm avoids the increase of the computational time of the whole procedure. Eight different versions of velocities equation were used in order to test which one of them is the most appropriate for the selected problem.

CNTPSO. In [108], a hybrid particle swarm optimization algorithm was used for the solution of the location routing problem. The PSO algorithm was hybridized with three other effective algorithms, the multiple phase neighborhood search – greedy randomized adaptive search procedure (MPNS-GRASP) [115], the expanding neighborhood search (ENS) [114], and the path relinking [67].

The novelty of the paper was that a procedure was used where there is no transformation from continuous to discrete values and the movement of the particles was performed using a path relinking procedure [67], where the current solution of the particle was used as starting solution and the target solution was either the local best particle or the global best particle. This was the first step of the very successful and effective topology that was denoted as combinatorial neighborhood topology and was used for the solution of the capacitated vehicle routing problem [111]. The velocities equation used was the inertia one (Eq. 6). In [111], one of the most successful versions of the PSO for the CVRP was published, denoted as combinatorial neighborhood topology PSO. This algorithm does not need a transformation from continuous to discrete values, and, thus, there is no loss of any information of good solutions that have been created in an iteration. In this algorithm, the position equation (Eq. 3) of the particles has been replaced by a path relinking strategy. The role of the velocities equation is limited to show which particle (or combination of particles) will be followed by the selected particle based on some conditions. This version of PSO is a very efficient version as it was proved by its application in the classic set of benchmark instances where better solutions were found compared to the ones found by other versions of PSO in less computational time.

In [112], an improvement of the CNTPSO was presented where a combination of this topology with an expanding neighborhood topology (ENT) is used. In this topology, there are not a standard number of local neighbors, but the number of local neighbors begins from a small number and increases using some conditions. When the number of neighbors becomes equal with the number of particles, then a global best PSO algorithm is used. The algorithm uses the same local search algorithm and the same initialization procedure as the algorithm presented in [117].

An improved version of CNTPSO with expanding neighborhood topology was presented in [107] for the solution of the location routing problem and of the location routing problem with stochastic demands. The difference of this algorithm from the initial CNTPSO is that it uses simultaneously a global neighborhood topology and the expanding neighborhood topology as it was described previously.

An adaptive version of CNTPSO is presented in [118] for the solution of the vehicle routing problem with time windows. In this version, all parameters (acceleration coefficients, iterations, local search iterations, upper and lower bounds of the velocities and of the positions, and number of particles in each swarm) are adapted during the procedure, and, thus, the algorithm works independently and without any interference from the user.

h_PSO. In [68, 69], two different algorithms for the solution of VRPSPD problem using a PSO algorithm were presented. Initially, it was presented as a conference paper [68], and its improved version was published as a journal paper in [69]. The authors presented a hybridization of PSO algorithm with a variable neighborhood search algorithm and used an annealing-based mechanism in order to maintain the diversity of particles. The inertia velocities equation was used (Eq. 6). They used a representation that began with a giant tour without taking into account the route restrictions, and, then, they used a decoding scheme to partition the giant tour into feasible routes. The procedure that they used is denoted as split procedure, and it was first presented by Prins in the frame of an evolutionary algorithm for the solution of the CVRP [143]. The algorithms were tested in the usually used benchmark set of instances for the problem, and a number of new best solutions were found.

DAPSO. The most important application of a particle swarm optimization algorithm for the solution of a dynamic vehicle routing problem was presented in [91]. In this paper, the authors solved the DVRP with dynamic requests, and a procedure was used where a partial static VRP was solved each time a new request was received. The authors hybridized the particle swarm optimization algorithm with a variable neighborhood search algorithm for the solution of the problem (DAPSO). The representation used in this work is a simple discrete representation which expresses the route of m vehicles over the n customers to serve. The representation allows the insertion of dynamic customers in the already planned route, and it is a permutation of the customers where as a DVRP is solved, the authors keep the coordinates of the customers, the time in which each customer is served, and if a customer has been served or not. Another very interesting application of PSO algorithm in this problem was presented in [89] where a multiswarm PSO algorithm was applied, a number of benchmark instances were solved, and comparisons with algorithms from the literature were given. Finally, a third publication of the same group of authors using a PSO algorithm was presented in [90].

SPSO. One of the most important applications of a PSO algorithm in the solution of the vehicle routing problem with time windows is the one presented in [73]. In this paper, the proposed SPSO-VRPTW treated the discrete search space as an arc set of the complete graph that was defined by the nodes in the VRPTW and regarded the candidate solution as a subset of arcs. A set-based representation method was proposed to characterize the discrete search space. The search space is represented by a universal set S. The elements in S can be divided into D dimensions.

2MPSO. A two-phase particle swarm optimization algorithm for the solution of the DVRP was presented in [131], and later it was improved in [132]. They presented an algorithm where a new equation of velocities for the particles was presented where local and global search topologies were combined.

PVPSO. In [120], a PSO algorithm for the OVRP is presented. The authors used a standard PSO for the encoding and the decoding procedure where all the elements of the positions vector are sorting in descending order and, then, the first is added in the first route, the second in the route with the least residual capacity, and so on until all elements are inserted in a route. The routes, then, were improved using one-move local search. They used the inertia equation of velocities (Eq. 6), and they tested their algorithm in a small set of instances and found very good solutions.

GLNPSO. A very interesting application of a PSO in the VRPFDs was the one presented in [185]. In this problem, the variant of VRP used included, also, soft time windows constraints, and the authors considered two objective functions, the minimization of the total travel cost and the maximization of the average satisfaction level of all customers in a fuzzy environment. The authors used a global-local-neighbor particle swarm optimization with exchangeable particles. An illustrative example that explains how the PSO algorithm had been applied in this problem was presented and analyzed in details.

PMPSO. In [92], the authors presented an approach that uses a probability matrix as the main device for particle encoding and decoding. In the decoding phase, not only the assignment of the customers in vehicles was realized, but, also, the routing of the customers was calculated. They used a number of local search algorithms (1-1 exchange, 2-opt, Or-opt) in order to improve the solution of the particles.

MOPSO. A very interesting formulation of a multiobjective competitive open vehicle routing problem with time windows was presented in [128]. In this problem, the reaching time to customers affects the sales amount. Therefore, distributors intend to service customers earlier than rivals to obtain the maximum sales. Moreover, a part of a driver's benefit is related to the amount of sales. Thus, the balance of goods carried in each vehicle is important in view of the limited vehicle capacities. They gave a complete analysis of the new problem and of the formulation, and they used a multiobjective PSO algorithm for the solution of the problem. In this multiobjective algorithm, the nondominated solutions are stored in

a repository, and, then, if a new solution dominates a solution from the repository or a personal best solution of a particle, then the new solution replaces the dominated solution.

StPSO. A very interesting paper was published in [173]. In this paper, the authors solved a very complicated vehicle routing and scheduling problem with cross-docking. They use an encoding scheme to represent the scheduling in a string. Thus, all are encoded as genes in a 1-by-n string where the length is equal to the number of vehicles. Thus, the digit in the i_{th} chromosome indicates the route number to which the i_{th} vehicle is assigned [173]. A similar variant of the algorithm has been proposed in [166] for the solution of the periodic vehicle routing problem.

NPSO. In [83], a nested PSO was given for the solution of the CVRP. Initially, feasible solutions are created in clusters using sweep algorithm, and, then, a route optimization was performed inside the clusters. The PSO was used in both phases, in the first phase for reorganizing the routes and in the second phase for leading to the optimization of the routes.

DPSO. A discrete PSO algorithm is presented in [125] for the solution of the team orienteering problem. The authors hybridized their algorithm with a variable neighborhood search algorithm. Another application of DPSO for the solution of the CVRP was proposed in [147]. The authors hybridized their algorithm with an iterated local search algorithm.

DHPD. In [75], a hybridization of a PSO algorithm with a differential evolution algorithm was presented. In the paper, an indirect representation was proposed. For N customers, each individual was encoded as a real number vector with N dimensions. The integer part of each dimension or element in the vector represents the vehicle. Thus, the same integer part represents the customer in the same vehicle. The fractional part represents the sequence of the customer in the vehicle [75]. Another similar formulation of the algorithm was proposed in [180] for the solution of the OVRP. The authors presented a very interesting representation of the particles' solution where in the encoding procedure, the customers took real values and in the decoding procedure, the customers with the same integer part are assigned in the same team (vehicle or group of vehicles if the capacity of the vehicle was violated). The authors tested their algorithm using the classic set of benchmark instances, and they obtained very good results.

MODPSO. In [195], a PSO algorithm was used for the solution of the VRPSPD. The very interesting part of the algorithm was the solution representation where the m customers were divided in $m + 1$-dimensional particles. In the decoding process of the algorithm, the particles were transformed to vehicle allocation s with the sweep algorithm, and, then, the priority matrices of customers served by the same vehicle were evaluated. Based on the two matrices, the vehicle routes were constructed.

Analysis of the Algorithms

In Table 3, the most important papers that use a variant of PSO for a solution of a variant of VRP are presented. The selection of the papers has been performed based on the criteria mentioned previously. The ranking in Table 3 is based on the number of citations that each of these papers has on 30 May 2017 on Google Scholar. Also, in Table 3, papers with a small number of citations are presented, but these papers either present very effective algorithms based on their computational results or have great potential for fast increase on the number of citations due to the year of publication. In Table 3, in addition to the number of citations, it is presented which variant of the VRP is solved (column 2), with which variant of PSO (column 3), in which journal, book, and conference it is published (column 4), and the year published (column 5). The number of citations is given in the last column.

As we can see, there is only one paper with more than 200 citations [3] with 289 citations and one that it is near to 200 (paper [33] with 198 citations). Two other papers have more than 150 citations (paper [109] with 167 and paper [4] with 162 citations). Finally, there are two papers with more than 100 citations, paper [116] with 130 citations, and paper [73] with 117 citations. In total, there are six papers with more than 100 citations. Four of them present algorithms for solving the capacitated vehicle routing problem, and the other two present algorithms for solving the vehicle routing problem with simultaneous pickup and delivery. Also, two of them ([3] and [4]) are works from a specific research group that have made a significant contribution in the field. In Table 3, there are, also, other four papers from the same research group (paper [2] for the solution of the VRPTW with 49 citations, paper [161] for the solution of MDVRP with 23 citations, paper [82] for the solution of MDVRP with 8 citations, and paper [104] for the solution of LRP with 3 citations). The other research group with significant contribution in the field studied in this chapter is the one that has two publications in the six more significant publications in the field ([116] and [109]) and other six publications that are presented in Table 3 (paper [117] for the solution of the VRPSD with 76 citations, paper [108] for the solution of LRP with 67 citations, paper [111] for the solution of CVRP with 9 citations, paper [110] for the solution of OVRP with 8 citations, paper [112] for the solution of VRPSD with 7 citations, paper [107] for the solution of LRP and LRPSD with 5 citations). Of course, there are papers with significant impact that are presented in this table as papers with 78 citations [91], papers with 74 citations [73], or papers with 70 citations [120] which have influenced a large number of researchers.

One of the most important tables in this survey is the last one (Table 4). In this table, the computational results of the most important papers are summarized and presented analytically. In this table, they are not presented papers that the proposed algorithm was tested in one or in a small set of instances, especially when these instances are not available in the Internet. Also, they are not presented papers that the proposed PSO algorithm solves a real-life problem as we cannot compare the results and we cannot see their effectiveness. The only papers that are presented are the ones that the proposed algorithm was tested in one or more well-known

Table 3 Total number of citations in the most important papers that use a PSO variant for solving a VRP variant

Paper	VRP variant	PSO variant	Published in	Year	Citations
Ai and Kachitvichyanukul [3]	VRPSPD	SR-PSO	COR	2009	289
Chen et al. [33]	CVRP	QPSO	SCIENCE A	2006	198
Marinakis and Marinaki [109]	CVRP	HybGenPSO	ESWA	2010	167
Ai and Kachitvichyanukul [4]	CVRP	SR-PSO	CIE	2009	162
Marinakis et al. [116]	CVRP	HybPSO	EAAI	2010	130
Goksal et al. [69]	VRPSPD	h_PSO	CIE	2013	117
Khouadjia et al. [91]	DVRP	DAPSO	ASOC	2012	78
Marinakis et al. [117]	VRPSD	HybPSO	ASOC	2013	76
Gong et al. [73]	VRPTW	SPSO	IEEE SMC	2012	74
MirHassani and Abolghasemi [120]	OVRP	PVPSO	ESWA	2011	70
Marinakis and Marinaki [108]	LRP	CNTPSO	JMMA	2008	67
Moghaddam et al. [121]	VRPUD	SR-PSO	CIE	2012	57
Xu et al. [185]	VRPTW	GLNPSO	TRE	2011	55
Ai and Kachitvichyanukul [2]	VRPTW	SR-PSO	IJOR	2009	49
Belmecheri et al. [16]	HVRP	SR-PSO	JIM	2013	42
Yao et al. [188]	HVRP	IPSO	ANOR	2016	33
Kim and Son [92]	CVRP	PMPSO	JIM	2012	31
Norouzi et al. [128]	OVRP	MOPSO	NETS	2012	23
Sombuntham and Kachitvichayanukul [161]	MDVRP	SR-PSO	IMECS 2010	2010	23
Vahdani et al. [173]	VRSP	StPSO	JIM	2012	21
Hu et al. [76]	VRPTW	SR-PSO	Entropy	2013	20
Okulewicz and Mańdziuk [131]	DVRP	2MPSO	ICAISC 2013	2013	16
Kanthavel and Prasad [83]	CVRP	NPSO	AJAS	2011	13
Muthuswamy and Lam [125]	TOP	DPSO	MEME	2011	13
Norouzi et al. [129]	PVRP	IPSO	Measurement	2015	12
Marinakis and Marinaki [111]	CVRP	CNTPSO	EvoCOP	2013	9
Qi [147]	CVRP	DPSO	PrE	2011	8
Marinakis and Marinaki [110]	OVRP	HybPSO	ANTS	2012	8
Kachitvichyanukul et al. [82]	MDVRP	SR-PSO	CIE	2015	8
Hu and Wu [75]	OVRP	DHPD	WCICA	2010	7
Marinakis and Marinaki [112]	VRPSD	CNTPSO	GECCO	2013	7
Marinakis [107]	LRPSD	CNTPSO	ASOC	2015	5
Tavakkoli Moghaddam et al. [166]	PVRP	StPSO	TRJ	2012	4
Liu and Kachitvichyanukul [104]	LRP	SR-PSO	IEAS	2013	3
Okulewicz and Mańdziuk [132]	DVRP	2MPSO	ASOC	2017	0

set(s) of benchmark instances, and the authors presented their results in such a way that the results are comparable with the computational results of other algorithms from the literature. In Table 4 in the first column, a reference of the paper is given, in the second column the problem(s) that is (are) solved is mentioned, and in the third, fourth, and fifth columns, the number of sets of benchmark instances, the

Table 4 Computational results of papers that used a variant of a PSO algorithm for the solution of a vehicle routing problem variant

Paper	VRP	Sets	NBI	Nodes	NBS	EBS	Quality	Quality range
Ai and Kachitvichyanukul [3]	VRPSPD	3	84	20–40	Average results are given			
			40	50	Average results are given			
			14	50–199	Average results are given			
Chen et al. [33]	CVRP	1	16	29–134	0	7	0.969	0.00–4.56
Marinakis and Marinaki [109]	CVRP	2	14	51–200	0	10	0.046	0.00–0.23
			20	200–483	0	1	0.60	0.00–0.91
Ai and [4] Kachitvichyanukul	CVRP	2	16	29–134	0	10	0.065	0.00–0.29
			14	51–200	0	4	0.874	0.00–2.51
Marinakis et al. [116]	CVRP	1	14	51–200	0	7	0.084	0.00–0.29
Goksal et al. [69]	VRPSPD	2	40	50	0	40	0.00	0.00
			14	50–199	0	6	0.00	0.00–3.16
Khouadjia et al. [91]	DVRP	1	21	50–199	5	0	NMl	NM
Marinakis et al. [117]	VRPSD	2	21	51–200	21	0	−1.105	−3.13 to −0.001
			40	16–60	New set of benchmark instances			
Gong et al. [73]	VRPTW	3	56	25	0	0	8.72	0.18–60.56
		29 best solutions based on number of vehicles						
			56	50	0	0	6.99	0.00–41.33
		32 best solutions based on number of vehicles						
			56	100	6	9	2.96	−12.69 to 15.70
MirHassani and Abolghasemi [120]	OVRP	1	15	32–50	0	12	0.196	0.00–2.605
Marinakis and Marinaki [108]	LRP	1	19	12–318	6	13	−0.08	−0.68 to 0.00
Moghaddam et al. [121]	VRPUD	1 (CVRP)	60	32–101	0	53	0.04	0.00–1.027
Ai and [2] Kachitvichyanukul	VRPTW	2	56	25	0	0	0.323	0.2–1.2
			56	50	0	0	1.148	0.2–7.2
Kim and Son [92]	CVRP	2	16	29–134	0	10	0.142	0.00–0.733
			14	51–200	0	6	0.712	0.00–2.95
Sombuntham and Kachitvichayanukul [161]	MDVRP	1	29	100	0	15	0.96	0–13.94
Kanthavel and Prasad [83]	CVRP	1	16	29–134	0	16	0.00	0.00–0.00
Qi [147]	CVRP	1	12	32–80	0	4	0.577	0.00–2.40
Marinakis and Marinaki [110]	OVRP	3	14	51–200	0	7	0.11	0.00–0.32
			8	51–200	0	4	0.13	0.00–0.38
			8	200–480	0	0	0.07	0.01–0.21
Marinakis and Marinaki [111]	CVRP	2	14	51–200	0	11	0.019	0.00–0.14
			20	200–483	0	1	0.37	0.00–0.81

(*continued*)

Table 4 continued

Paper	VRP	Sets	NBI	Nodes	NBS	EBS	Quality	Quality range
Marinakis and Marinaki [112]	VRPSD	1	40	16–60	27	13	−0.64	−3.18 to 0.00
Liu and Kachitvichyanukul [104]	LRP	1	30	50–200	1	0	3.29	−0.02 to 16.12
Marinakis [107]	LRP	6	16	12–150	0	8	0.24	0.00–2.12
			30	20–200	0	13	0.42	0.00–2.54
			36	100–200	0	6	0.86	0.00–3.10
	LRPSD		16	12–150	15	1	−0.11	0.00–0.41
			30	20–200	10	20	0.00	−0.02 to 0.00
			36	100–200	32	4	−0.04	−0.08 to 0.00

number of instances in each set (NBI), and the range in which the number of nodes of each set fluctuates are given for each paper, respectively. Finally, in the last four columns, the number of new best solutions found in the paper in the year of its publication (NBS), the number of instances in which the proposed algorithm found a solution equal to the best-known published solution (EBS), the average quality of the solutions, and the range of the qualities of the solutions are given, respectively. The quality of a solution is calculated using the following equation $\text{Quality} = \frac{(c_{\text{PSO}} - c_{\text{BKS}})}{c_{\text{BKS}}}\%$, where c_{PSO} denotes the cost of the solution found by the mentioned PSO algorithm in the corresponding paper and c_{BKS} is the cost of the best-known solution.

It was a difficult task to summarize all the results and to create this table as every researcher presents the produced results in a different way. For example, in the most important and cited paper [3], the authors present the average results in the well-known benchmark instances, and, then, they give an extensive analysis in variants of the studied problem, and it was difficult to present analytically the results of the paper. In other papers, a new set of benchmark instances is created [117], and, thus, it was difficult to give comparisons for this set of instances. However, in most of the papers that are presented in this table, we could extract the information that we would like to have in order to see the effectiveness of each one of the algorithms. In most of the algorithms presented in the table, at least a number of EBS were found, and in few of them, new best solutions were given. It is very difficult to analyze each paper separately due to space limitation, and this analysis can be found easily in each one of the papers. However, we could say that the application of PSO in VRP variants gives very interesting computational results. In most cases, the computational results are competitive with the results of the most effective algorithms from the literature. Thus, alternative propositions exist for someone that he/she would like to apply a PSO algorithm for the solution of a VRP variant.

Conclusions

In this chapter, an analytical review of the papers proposing and applying a PSO algorithm for the solution of VRP variants was presented. In total, around 100 papers were found. In some of them, new best solutions in VRP variants were presented, in other papers the authors presented the solutions that have been found that are very good and competitive solutions and in some instances equal to the best-known solutions but not new best solutions in the VRP variant solved, in other papers the authors presented and solved a new variant of the VRP with a PSO algorithm, and, finally, there are some papers that gave only illustrative examples of how the authors worked in specific instances of a VRP variant. In general, this chapter gives to the research community a basis that someone could use if he/she would like to apply a PSO algorithm for solving a specific variant of the VRP as he/she could find what has already been published and with what results and he/she could see how the researchers cope with the problems that may arise from the application of a PSO algorithm to the VRP or to its variants.

Cross-References

▸ Hyper-heuristics
▸ Particle Swarm Methods

References

1. Adulyasak Y, Cordeau JF, Jans R (2014) Optimization-based adaptive large neighborhood search for the production routing problem. Transport Sci 48(1):20–45
2. Ai TJ, Kachitvichyanukul V (2009) A particle swarm optimisation for vehicle routing problem with time windows. Int J Oper Res 6(4):519–537
3. Ai TJ, Kachitvichyanukul V (2009) A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. Comput Oper Res 36:1693–1702
4. Ai TJ, Kachitvichyanukul V (2009) Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem. Comput. Ind. Eng. 56:380–387
5. Angelelli E, Speranza MG (2002) The periodic vehicle routing problem with intermediate facilities. Eur J Oper Res 137(2):233–247
6. Archetti C, Speranza MG, Hertz A (2006) A tabu search algorithm for the split delivery vehicle routing problem. Transp Sci 40(1):64–73
7. Archetti C, Speranza MG (2008) The split delivery vehicle routing problem: a survey. In: Golden B, Raghavan S, Wasil E (eds) The vehicle routing problem: latest advances and new challenges. Springer, Boston, pp 103–122
8. Archetti C, Speranza MG, Vigo D (2014) Vehicle routing problems with profits. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications. MOS-SIAM series on optimization. SIAM, Philadelphia, pp 273–298
9. Assad AA, Golden BL (1995) Arc routing methods and applications. In: Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing, handbooks in operations research and management science, vol 8. Elsevier Science B V, Amsterdam, pp 375–483

10. Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing, handbooks in operations research and management science, vol 8. Elsevier Science B V, Amsterdam
11. Banks A, Vincent J, Anyakoha C (2007) A review of particle swarm optimization. Part I: background and development. Nat Comput 6(4):467–484
12. Banks A, Vincent J, Anyakoha C (2008) A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Nat Comput 7:109–124
13. Bard, JF, Nananukul N (2009) The integrated production inventory distribution routing problem. J Sched 12(3):257–280
14. Bashiri M, Fallahzade E (2012) A Particle swarm optimization algorithm for multi-depot capacitated location-routing problem with inventory decisions in supply chain network design. In: CIE42 proceedings, Cape Town, 16–18 July 2012. CIE and SAIIE, pp 25-1–25-9
15. Bektas T, Laporte G (2011) The pollution-routing problem. Transp Res B Methodol 45(8):1232–1250
16. Belmecheri F, Prins C, Yalaoui F, Amodeo L (2013) Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. J Intell Manuf 24(4):775–789
17. Berbeglia G, Cordeau JF, Gribkovskaia I, Laporte G (2007) Static pickup and delivery problems: a classification scheme and survey. TOP 15(1):1–31
18. Bianchi L, Birattari M, Manfrin M, Mastrolilli M, Paquete L, Rossi-Doria O, Schiavinotto T (2006) Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J Math Model Algorithm 5(1):91–110
19. Bodin L, Golden B (1981) Classification in vehicle routing and scheduling. Networks 11: 97–108
20. Bodin L, Golden B, Assad A, Ball M (1983) The state of the art in the routing and scheduling of vehicles and crews. Comput Oper Res 10:63–212
21. Bortfeldt A (2012) A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. Comput Oper Res 39(9):2248–2257
22. Bozorgi-Amiri A, Jabalameli MS, Alinaghian M, Heydari M (2012) A modified particle swarm optimization for disaster relief logistics under uncertain environment. Int J Adv Manuf Technol 60(1–4):357–371
23. Braysy O, Gendreau M (2005) Vehicle routing problem with time windows, Part I: route construction and local search algorithms. Transp Sci 39(1): 104–118
24. Braysy O, Gendreau M (2005) Vehicle routing problem with time windows, Part II: metaheuristics. Transp Sci 39(1): 119–139
25. Brito J, Exposito A, Moreno-Pérez JA (2015) Bi-objective discrete PSO for service-oriented VRPTW (Chapter 29). In: Greiner D et al (eds) Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences. Computational methods in applied sciences, vol 36. Springer International Publishing, Cham, pp 445–460. https://doi.org/10.1007/978-3-319-11541-2_29
26. Campbell A, Clarke L, Kleywegt A, Sawelsberg M (1998) The inventory routing problem. In: Crainic TG, Laporte G (eds) Fleet management and logistics. Kluwer Academic Publishers, Boston, pp 95–113
27. Campbell A, Clarke L, Sawelsberg M (2002) Inventory routing in practice. In: Toth P, Vigo D (eds) The vehicle routing problem. Monographs on discrete mathematics and applications. Siam, Philadelphia, pp 309–330
28. Caceres-Cruz J, Arias P, Guimarans D, Riera D, Juan AA (2015). Rich vehicle routing problem: survey. ACM Comput Surv (CSUR) 47(2):32
29. Caretto C, Baker B (2002) A GRASP interactive approach to the vehicle routing problem with backhauls. In: Ribeiro CC, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer Academic Publishers, Norwell, pp 185–199
30. Casco DO, Golden BL, Wasil EA (1988) Vehicle routing with backhauls: models, algorithms, and case studies. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 127–147

31. Castro JP, Landa-Silva D, Moreno Perez JA (2009) Exploring feasible and infeasible regions in the vehicle routing problem with time windows using a multi-objective particle swarm optimization approach (Chapter 9). In: Krasnogor N et al (eds) Nature inspired cooperative strategies for optimization. SCI, vol 236. Springer, Berlin/Heidelberg, pp 103–114
32. Chao IM, Golden BL, Wasil E (1993) A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions. Am J Math Manag Sci 13(3–4):371–406
33. Chen A-L, Yang G-K, Wu Z-M (2006) Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. J Zhejiang Univ Sci A 7(4):607–614
34. Chen J-Q, Li W-L, Murata T (2013) Particle swarm optimization for vehicle routing problem with uncertain demand. In: Proceedings of 2013 4th IEEE international conference on software engineering and service science (ICSESS), Beijing, 23–25 May 2013, pp 857–860
35. Chen S-K, Wu G-H, Ti Y-W, Wang R-Z, Fang W-P, Lu C-J (2014) Hierarchical particle swarm optimization algorithm of IPSVR problem. In: Pan J-S et al (eds) Genetic and evolutionary computing. Advances in intelligent systems and computing, vol 238. Springer International Publishing, Cham, pp 231–238. https://doi.org/10.1007/978-3-319-01796-9_24
36. Chen MC, Hsiao YH, Reddy RH, Tiwari MK (2016) The self-learning particle swarm optimization approach for routing pickup and delivery of multiple products with material handling in multiple cross-docks. Transp Res E 91:208–226
37. Christiansen M, Fagerholt K, Ronen D (2004) Ship routing and scheduling: status and perspectives. Transp Sci 38(1):1–18
38. Christiansen M, Fagerholt K, Nygreen B, Ronen D (2013) Ship routing and scheduling in the new millennium. Eur J Oper Res 228:467–483
39. Christofides N (1985) Vehicle routing. In: Lawer EL, Lenstra JK, Rinnoy Kan AHG, Shmoys DB (eds) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, Chichester, pp 431–448
40. Christofides N, Mignozzi A, Toth P (1979) The vehicle routing problem. In: Christofides N (ed) Combinatorial optimization. Wiley, Chichester, pp 315–338
41. Clerc M (2006) Particle swarm optimization. ISTE, London
42. Clerc M, Kennedy J (2002) The particle swarm: explosion, stability and convergence in a multi-dimensional complex space. IEEE Trans Evol Comput 6:58–73
43. Cordeau JF, Deasulniers G, Desrosiers J, Solomon MM, Soumis F (2002) VRP with time windows. In: Toth P, Vigo D (eds) The vehicle routing problem. Monographs on discrete mathematics and applications. SIAM, Philadelphia, pp 157–193
44. Dallard H, Lam SS, Kulturel-Konak S (2007) Solving the orienteering problem using attractive and repulsive particle swarm optimization. In: Proceedings of IEEE international conference on Information Reuse and Integration (IRI 2007), Las Vegas, 13–15 Aug 2007, pp 2–17
45. Dang D-C, Guibadj RN, Moukrim A (2011) A PSO-based memetic algorithm for the team orienteering problem. In: Di Chio C et al (eds) EvoApplications 2011, Part II. LNCS, vol 6625. Springer, Berlin/Heidelberg, pp 471–480
46. Dantzig GB, Ramser JH (1959) The truck dispatching problem. Manag Sci 6(1):80–91
47. Daskin M (1995) Network and discrete location. Models, algorithms and applications. Wiley, New York
48. De A, Mamanduru VKR, Gunasekaran A, Subramanian N, Tiwari MK (2016) Composite particle algorithm for sustainable integrated dynamic ship routing and scheduling optimization. Comput Ind Eng 96:201–215
49. Desrochers M, Lenstra JK, Savelsberg MWP, Soumis F (1988) Vehicle routing with time windows: optimization and approximation. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 65–84
50. Desrosiers J, Dumas Y, Solomon MM, Soumis F (1995) Time constraint routing and scheduling. In: Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing, handbooks in operations research and management science, vol 8. Elsevier Science B V, Amsterdam, pp 35–140

51. Di-Ming A, Zhe Z, Rui Z, Feng P (2011) Research of pareto-based multi-objective optimization for multi-vehicle assignment problem based on MOPSO. In: Tan Y et al (eds) ICSI 2011, Part II. LNCS, vol 6729. Springer, Berlin/Heidelberg, pp 10–16
52. Eksioglu B, Vural AV, Reisman, A (2009) The vehicle routing problem: a taxonomic review. Comput Ind Eng 57(4):1472–1483
53. Engelbrecht AP (2007) Computational intelligence: an introduction, 2nd edn. Wiley, Chichester
54. Fedegruen A, Simchi-Levi D (1995) Analysis of vehicle routing and inventory routing problems. In: Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing. Handbooks in operations research and management science, vol 8. Elsevier Science B V, Amsterdam, pp 297–373
55. Fisher ML (1995) Vehicle routing. In: Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing. Handbooks in operations research and management science, vol 8. North Holland, Amsterdam, pp 1–33
56. Francis PM, Smilowitz KR, Tzur M (2008) The period vehicle routing problem and its extensions. In: Golden B et al (eds) The vehicle routing problem: latest advances and new challenges. Springer LLC, Boston, pp 73–102
57. Gan X, Wang Y, Yu Y, Niu B (2013) An emergency vehicle scheduling problem with time utility based on particle swarm optimization. In: Huang D-S et al (eds) ICIC 2013. LNAI, vol 7996. Springer, Berlin/Heidelberg, pp 614–623
58. Gan X, Kuang J, Niu B (2014) Particle swarm optimizations for multi-type vehicle routing problem with time windows. In: Huang D-S et al (eds) ICIC 2014. LNAI, vol 8589. Springer International Publishing, Cham, pp 808–815
59. Gan X, Liu LJ, Chen JS, Niu B (2016) Comprehensive learning PSO for solving environment heterogeneous fixed fleet VRP with time windows. In: Tan Y et al (eds) ICSI 2016, Part II. LNCS, vol 9713. Springer, Cham, pp 424–432
60. Gavalas D, Konstantopoulos C, Mastakas K, Pantziou G (2014) A survey on algorithmic approaches for solving tourist trip design problems. J Heuristics 20(3):291–328
61. Gendreau M, Potvin JY (1998) Dynamic vehicle routing and dispatching. In: Crainic TG, Laporte G (eds) Fleet management and logistics. Kluwer Academic Publishers, Boston, pp 115–125
62. Gendreau M, Laport G, Seguin R (1996) Stochastic vehicle routing. Eur J Oper Res 88:3–12
63. Gendreau M, Laporte G, Potvin J-Y (1997) Vehicle routing: modern heuristics. In: Aarts EHL, Lenstra JK (eds) Local search in combinatorial optimization. Wiley, Chichester, pp 311–336
64. Gendreau M, Laporte G, Musaraganyi C, Taillard ED (1999) A tabu search heuristic for the heterogeneous fleet vehicle routing problem. Comput Oper Res 26:1153–1173
65. Gendreau M, Laporte G, Potvin J-Y (2002) Metaheuristics for the capacitated VRP. In: Toth P, Vigo D (eds) The vehicle routing problem. Monographs on discrete mathematics and applications. SIAM, Philadelphia, pp 129–154
66. Gendreau M, Iori M, Laporte G, Martello S (2008) A Tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 51(1):4–18
67. Glover F, Laguna M, Marti R (2003) Scatter search and path relinking: advances and applications. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. Kluwer Academic Publishers, Boston, pp 1–36
68. Goksal FP, Altiparmak F, Karaoglan I (2010) A hybrid particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. In: Proceedings of 2010 40th international conference on Computers and Industrial Engineering (CIE), Awaji, 25–28 July 2010, pp 1–6
69. Goksal FP, Karaoglan I, Altiparmak F (2013) A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. Comput Ind Eng 65: 39–53
70. Golden BL, Assad AA (1988) Vehicle routing: methods and studies. North Holland, Amsterdam

71. Golden BL, Wasil EA, Kelly JP, Chao IM (1998) The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic TG, Laporte G (eds) Fleet management and logistics. Kluwer Academic Publishers, Boston, pp 33–56
72. Golden BL, Raghavan S, Wasil EA (eds) (2008) The vehicle routing problem: latest advances and new challenges. Operations research/computer science interfaces series, vol 43. Springer LLC, Boston
73. Gong Y-J, Zhang J, Liu O, Huang R-Z, Chung HS-H, Shi Y-H (2012) Optimizing the vehicle routing problem with time windows: a discrete particle swarm optimization approach. IEEE Trans Syst Man Cybern C Appl Rev 42(2):254–267
74. Hansen P, Mladenovic N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130:449–467
75. Hu F, Wu F (2010) Diploid hybrid particle swarm optimization with differential evolution for open vehicle routing problem. In: Proceedings of the 8th world congress on intelligent control and automation, Jinan, 6–9 July 2010
76. Hu W, Liang H, Peng C, Du B, Hu Q (2013) A hybrid chaos-particle swarm optimization algorithm for the vehicle routing problem with time window. Entropy 15:1247–1270. https://doi.org/10.3390/e15041247
77. Jaillet P, Odoni AR (1988) The probabilistic vehicle routing problem. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 293–318
78. Javid AA, Azad N (2010) Incorporating location, routing and inventory decisions in supply chain network design. Transport Res E Log Transp Rev 46(5):582–597
79. Jian L (2009) Solving capacitated vehicle routing problems via genetic particle swarm optimization. In: Proceedings of 2009 third international symposium on intelligent information technology application, Nanchang, 21–22 Nov 2009, pp 528–531
80. Jiang W, Zhang Y, Xie J (2009) A particle swarm optimization algorithm with crossover for vehicle routing problem with time windows. In: IEEE symposium on computational intelligence in scheduling (CI-Sched '09), Nashville, 30 Mar 2009–2 Apr 2009, pp 103–106
81. Jozefowiez N, Semet F, Talbi EG (2008) Multi-objective vehicle routing problems. Eur J Oper Res 189(2):293–309
82. Kachitvichyanukul V, Sombuntham P, Kunnapapdeelert S (2015) Two solution representations for solving multi-depot vehicle routing problem with multiple pickup and delivery requests via PSO. Comput Ind Eng 89:125–136
83. Kanthavel K, Prasad P (2011) Optimization of capacitated vehicle routing problem by nested particle swarm optimization. Am J Appl Sci 8(2):107–112
84. Kechagiopoulos PN, Beligiannis GN (2014) Solving the urban transit routing problem using a particle swarm optimization based algorithm. Appl Soft Comput 21:654–676
85. Kennedy J (1997) The particle swarm: social adaptation of knowledge. In: Proceedings of the IEEE international conference on evolutionary computation, Indianapolis, 13–16 Apr 1997, pp 303–308
86. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of 1995 IEEE international conference on neural networks, vol 4, pp 1942–1948
87. Kennedy J, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of 1997 IEEE international conference on systems, man, and cybernetics, vol 5, pp 4104–4108
88. Kirkpatrick S (1984) Optimization by simulated annealing – quantitative studies. J Stat Phys 34:975–986
89. Khouadjia MR, Alba E, Jourdan L, Talbi E-G (2010) Multi-swarm optimization for dynamic combinatorial problems: a case study on dynamic vehicle routing problem. In: Dorigo M et al (eds) ANTS 2010. LNCS, vol 6234. Springer, Berlin/Heidelberg, pp 227–238
90. Khouadjia MR, Jourdan L, Talbi E-G (2010) Adaptive particle swarm for solving the dynamic vehicle routing problem. In: Proceedings of 2010 IEEE/ACS international conference on computer systems and applications (AICCSA), Hammamet, 16–19 May 2010, pp 1–8

91. Khouadjia MR, Sarasola B, Alba E, Jourdan L, Talbi E-G (2012) A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. Appl Soft Comput 12:1426–1439
92. Kim B-I, Son S-J (2012) A probability matrix based particle swarm optimization for the capacitated vehicle routing problem. J Intell Manuf 23(4):1119–1126
93. Kou M, Ye C, Chen Z (2011) A bee evolutionary particle swarm optimization algorithm for vehicle routing problem. In: Proceedings of 2011 6th IEEE joint international information technology and artificial intelligence conference (ITAIC), Chongqing, vol 2, 20–22 Aug 2011, pp 398–401
94. Kumar RS, Kondapaneni K, Dixit V, Goswami A, Thakur LS, Tiwari MK (2016) Multi-objective modeling of production and pollution routing problem with time window: a self-learning particle swarm optimization approach. Comput Ind Eng 99:29–40
95. Kuo RJ, Zulvia FE, Suryadi K (2012) Hybrid particle swarm optimization with genetic algorithm for solving capacitated vehicle routing problem with fuzzy demand – a case study on garbage collection system. Appl Math Comput 219:2574–2588
96. Lahyani R, Khemakhem M, Semet F (2015) Rich vehicle routing problems: from a taxonomy to a definition. Eur J Oper Res 241:1–14
97. Laporte G (1988) Location routing problems. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 163–198
98. Laporte G, Semet F (2002) Classical heuristics for the capacitated VRP. In: Toth P, Vigo D (eds) The vehicle routing problem. Monographs on discrete mathematics and applications. SIAM, Philadelphia, pp 109–128
99. Laporte G, Nobert Y, Taillefer S (1988) Solving a family of multi-depot vehicle routing and location routing problems. Transp Sci 22:161–172
100. Laporte G, Gendreau M, Potvin, J-Y, Semet F (2000) Classical and modern heuristics for the vehicle routing problem. Int Trans Oper Res 7:285–300
101. Li Y, Li D, Wang D (2012) Quantum-behaved particle swarm optimization algorithm based on border mutation and chaos for vehicle routing problem. In: Tan Y, Shi Y, Ji Z (eds) ICSI 2012, Part I. LNCS, vol 7331, Springer, Berlin/Heidelberg, pp 63–73
102. Lichtblau D (2002) Discrete optimization using mathematica. In: Callaos N, Ebisuzaki T, Starr B, Abe JM, Lichtblau D (eds) World multi-conference on systemics, cybernetics and informatics (SCI 2002), vol 16. International Institute of Informatics and Systemics, pp 169–174
103. Lin C, Choy KL, Ho GTS, Chung SH, Lam HY (2014) Survey of green vehicle routing problem: past and future trends. Expert Syst Appl 41:1118–1138
104. Liu J, Kachitvichyanukul V (2013) A new solution representation for solving location routing problem via particle swarm optimization. In: Lin Y-K et al (eds) Proceedings of the institute of industrial engineers asian conference. Springer Science+Business Media, Singapore. https://doi.org/10.1007/978-981-4451-98-7_12
105. Liu X, Jiang W, Xie J (2009) Vehicle routing problem with time windows: a hybrid particle swarm optimization approach. In: Proceedings of 2009 fifth international conference on natural computation, Tianjin, 14–16 Aug 2009, pp 502–506
106. Liu SC, Lu MC, Chung CH (2016) A hybrid heuristic method for the periodic inventory routing problem. Int J Adv Manuf Technol 85:2345–2352
107. Marinakis Y (2015) An improved particle swarm optimization algorithm for the capacitated location routing problem and for the location routing problem with stochastic demands. Appl Soft Comput 37:680–701
108. Marinakis Y, Marinaki M (2008) A particle swarm optimization algorithm with path relinking for the location routing problem. J Math Model Algorithms 7(1):59–78
109. Marinakis Y, Marinaki M (2010) A hybrid genetic – particle swarm optimization algorithm for the vehicle routing problem. Expert Syst Appl 37:1446–1455
110. Marinakis Y, Marinaki M (2012) A hybrid particle swarm optimization algorithm for the open vehicle routing problem. In: Dorigo M et al (eds) ANTS 2012. LNCS, vol 7461. Springer, Berlin/Heidelberg, pp 180–187

111. Marinakis Y, Marinaki M (2013) Combinatorial neighborhood topology particle swarm optimization algorithm for the vehicle routing problem. In: Middendorf M, Blum C (eds) EvoCOP 2013. LNCS, vol 7832, pp 133–144
112. Marinakis Y, Marinaki M (2013) Combinatorial expanding neighborhood topology particle swarm optimization for the vehicle routing problem with stochastic demands. In: Proceedings of GECCO 2013, genetic and evolutionary computation conference, Amsterdam, 6–10 July 2013, pp 49–56
113. Marinakis Y, Migdalas Á (2002) Heuristic solutions of vehicle routing problems in supply chain management. In: Pardalos PM, Migdalas A, Burkard R (eds) Combinatorial and global optimization. World Scientific Publishing Co, Singapore, pp 205–236
114. Marinakis Y, Migdalas A, Pardalos PM (2005) Expanding neighborhood GRASP for the traveling salesman problem. Comput Optim Appl 32:231–257
115. Marinakis Y, Migdalas A, Pardalos PM (2009) Multiple phase neighborhood search GRASP based on lagrangean relaxation and random backtracking Lin-Kernighan for the traveling salesman problem. J Comb Optim 17:134–156
116. Marinakis Y, Marinaki M, Dounias G (2010) A hybrid particle swarm optimization algorithm for the vehicle routing problem. Eng Appl Artif Intel 23:463–472
117. Marinakis Y, Iordanidou G, Marinaki M (2013) Particle swarm optimization for the vehicle routing problem with stochastic demands. Appl Soft Comput 13(4):1693–1704
118. Marinakis Y, Marinaki M, Migdalas A (2014) An adaptive particle swarm optimization algorithm for the vehicle routing problem with time windows. In: Proceedings of LOT 2014, logistics, optimization and transportation conference, Molde, 1–2 Nov 2014
119. Min H, Jayaraman V, Srivastava R (1998) Combined location-routing problems: a synthesis and future research directions. Eur J Oper Res 108:1–15
120. MirHassani SA, Abolghasemi N (2011) A particle swarm optimization algorithm for open vehicle routing problem. Expert Syst Appl 38:11547–11551
121. Moghaddam BF, Ruiz R, Sadjadi SJ (2012)Vehicle routing problem with uncertain demands: an advanced particle swarm algorithm. Comput Ind Eng 62:306–317
122. Montoya-Torres JR, Franco JL, Isaza SN, Jimenez HF, Herazo-Padilla N (2015) A literature review on the vehicle routing problem with multiple depots. Comput Ind Eng 79: 115–129
123. Mosheiov G (1998) Vehicle routing with pickup and delivery: tour – partitioning heuristics. Comput Ind Eng 34:669–684
124. Muñoz-Zavala A, Hernández-Aguirre A, Villa-Diharce E (2009) Particle evolutionary swarm multi-objective optimization for vehicle routing problem with time windows. In: Coello Coello CA et al (eds) Swarm intelligence for multi-objective problems in data mining. SCI, vol 242. Springer, Berlin/Heidelberg, pp 233–257
125. Muthuswamy S, Lam SS (2011) Discrete particle swarm optimization for the team orienteering problem. Memetic Comput 3:287–303
126. Nagy G, Salhi S (2007) Location-routing: issues, models and methods. Eur J Oper Res 177:649–672
127. Ngueveu SU, Prins C, Calvo RW (2010) An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Comput Oper Res 37(11):1877–1885
128. Norouzi N, Tavakkoli-Moghaddam R, Ghazanfari M, Alinaghian M, Salamatbakhsh A (2012) A new multi-objective competitive open vehicle routing problem solved by particle swarm optimization. Netw Spat Econ 12(4):609–633
129. Norouzi N, Sadegh-Amalnick M, Mehdi A (2015) Evaluating of the particle swarm optimization in a periodic vehicle routing problem. Measurement 62:162–169
130. Norouzi N, Sadegh-Amalnick M, Tavakkoli-Moghaddam R (2016) Modified particle swarm optimization in a time-dependent vehicle routing problem: minimizing fuel consumption. Opt Lett. https://doi.org/10.1007/s11590-015-0996-y
131. Okulewicz M, Mańdziuk J (2013) Application of particle swarm optimization algorithm to dynamic vehicle routing problem. In: Rutkowski L et al (eds) ICAISC 2013, Part II. LNAI, vol 7895. Springer, Berlin/Heidelberg, pp 547–558

132. Okulewicz M, Mańdziuk J (2017) The impact of particular components of the PSO-based algorithm solving the dynamic vehicle routing problem. Appl Soft Comput 58:586–604
133. Olivera AC, Garcí a-Nieto JM, Alba E (2014) Reducing vehicle emissions and fuel consumption in the city by using particle swarm optimization. Applied intelligence. Springer Science+Business Media, New York. https://doi.org/10.1007/s10489-014-0604-3
134. Peng Y (2009) Hybrid particle swarm optimization for vehicle routing problem with reverse logistics. In: Proceedings of 2009 international conference on intelligent human-machine systems and cybernetics, Hangzhou, 26–27 Aug 2009, pp 462–465
135. Peng Y, Chen Z-X (2009) Two-phase particle swarm optimization for multi-depot location-routing problem. In: Proceedings of 2009 international conference on new trends in information and service science, Beijing, 30 June 2009–2 July 2009, pp 240–245
136. Peng Y, Chen J (2010) Vehicle routing problem with fuzzy demands and the particle swarm optimization solution. In: Proceedings of 2010 international conference on management and service science (MASS), Wuhan, 24–26 Aug 2010, pp 1–4
137. Peng Y, Zhu H-Y (2008) Research on vehicle routing problem with stochastic demand and PSO-DP algorithm with inver-over operator. Syst Eng Theory Pract (SETP) 28(10):76–81
138. Pereira FB, Tavares J (2008) Bio-inspired algorithms for the vehicle routing problem. Studies in computational intelligence, vol 161. Springer, Berlin/Heideberg
139. Perwira Redi AAN, Maghfiroh MFN, Yu VF (2013) Discrete particle swarm optimization with path-relinking for solving the open vehicle routing problem with time windows. In: Lin Y-K et al (eds) Proceedings of the institute of industrial engineers Asian conference 2013. Springer Science+Business Media, Singapore, pp 853–859. https://doi.org/10.1007/978-981-4451-98-7_102
140. Pillac V, Gendreau M, Gueret C, Medaglia AL (2013) A review of dynamic vehicle routing problems. Eur J Oper Res 225:1–11
141. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. An overview. Swarm Intell 1:33–57
142. Powell WB, Jaillet P, Odoni A (1995) Stochastic and dynamic networks and routing. In: Ball MO, Magnanti TL, Momma CL, Nemhauser GL (eds) Network routing. Handbooks in operations research and management science, vol 8. Elsevier Science B V, Amsterdam, pp 141–295
143. Prins C (2004) A simple and effective evolutionary algorithm for the vehicle routing problem. Comput Oper Res 31:1985–2002
144. Prodhon C, Prins C (2014) A survey of recent research on location-routing problems. Eur J Oper Res 238:1–17
145. Psaraftis HN (1988) Dynamic vehicle routing problems. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 223–248
146. Psaraftis HN (1995) Dynamic vehicle routing: status and prospects. Ann Oper Res 61: 143–164
147. Qi C (2011) Application of improved discrete particle swarm optimization in logistics distribution routing problem. Proc Eng Adv Control Eng Inf Sci 15:3673–3677
148. Rabbani M, Manavizadeh N, Shamekhi A (2013) A particle swarm optimization method for periodic vehicle routing problem with pickup and delivery in transportation. Adv Railw Eng Int J 1(1):51–60
149. Renaud J, Laporte G, Boctor FF (1996) A Tabu search heuristic for the multidepot vehicle routing problem. Comput Oper Res 23(3):229–235
150. Ronen D (1983) Cargo ships routing and scheduling: survey of models and problems. Eur J Oper Res 12:119–126
151. Ronen D (1993) Ships scheduling: the last decade. Eur J Oper Res 71(3):325–333
152. Sariklis D, Powell S (2000) A heuristic method for the open vehicle routing problem. J Oper Res Soc 51(5):564–573
153. Sevkli Z, Sevilgen FE (2010) Discrete particle swarm optimization for the orienteering problem. In: Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC), Barcelona, 18–23 July 2010, pp 1–8

154. Sevkli AZ, Sevilgen FE (2012) Discrete particle swarm optimization for the team orienteering problem. Turk J Electr Eng Comput Sci 20(2):231–239
155. Shao Z-J, Gao S-P, Wang S-S (2009) A hybrid particle swarm optimization algorithm for vehicle routing problem with stochastic travel time. In: Cao B-Y, Zhang C-Y, Li T-F (eds) Fuzzy information and engineering. ASC, vol 54. Springer, Berlin/Heidelberg, pp 566–574
156. Shen H, Zhu Y, Liu T, Jin L (2009) Particle swarm optimization in solving vehicle routing problem. In: Proceedings 2009 second international conference on intelligent computation technology and automation, Changsha, 10–11 Oct 2009, pp 287–291
157. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: Proceedings of 1998 IEEE world congress on computational intelligence, pp 69–73
158. Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 35(2):254–265
159. Solomon MM, Desrosiers J (1988) Time window constrained routing and scheduling problems. Transp Sci 22(1):1–13
160. Solomon MM, Baker EK, Schaffer JR (1988) Vehicle routing and scheduling problems with time windows constraints. In: Golden BL, Assad AA (eds) Vehicle routing: methods and studies. North Holland, Amsterdam, pp 85–105
161. Sombuntham P, Kachitvichayanukul V (2010) A particle swarm optimization algorithm for multi-depot vehicle routing problem with pickup and delivery Requests. In: Proceedings of the international multiconference of engineers and computer scientists (IMECS 2010), vol III, Hong Kong, 17–19 Mar 2010. ISBN:978-988-18210-5-8
162. Stewart WR, Golden BL (1983) Stochastic vehicle routing: a comprehensive approach. Eur J Oper Res 14:371–385
163. Tang H (2011) Vehicle route optimization in logistics distribution based on extension-coded particle swarm algorithm. In: Proceedings of 2011 international conference on computer science and network technology, Harbin, 24–26 Dec 2011, pp 2350–2354
164. Tang C, Wang T (2011) An improved particle swarm optimization for the vehicle routing problem with simultaneous deliveries and pick-ups. In: Shen G, Huang X (eds) CSIE 2011, Part II. CCIS, vol 153. Springer, Berlin/Heidelberg, pp 294–300
165. Tarantilis CD (2005) Solving the vehicle routing problem with adaptive memory programming methodology. Comput Oper Res 32(9):2309–2327
166. Tavakkoli Moghaddam R, Mohmmad Zohrevand A, Rafiee K (2012) Solving a new mathematical model for a periodic vehicle routing problem by particle swarm optimization. Transp Res J 1:77–87
167. Tavakoli MM, Sami A (2013) Particle swarm optimization in solving capacitated vehicle routing problem. Buletin Teknik Elektro dan Informatika (Bull Electr Eng Inf) 2(4):252–257
168. Ting C-J, Wu K-C, Chou H (2014) Particle swarm optimization algorithm for the berth allocation problem. Expert Syst Appl 41:1543–1550
169. Tlili T, Faiz S, Krichen S (2014) A hybrid metaheuristic for the distance-constrained capacitated vehicle routing problem. Procedia – social and behavioral sciences, 2nd world conference on business, economics and management-WCBEM 2013, vol 109, pp 779–783
170. Toth P, Vigo D (2002) The vehicle routing problem. Monographs on discrete mathematics and applications. SIAM, Philadelphia
171. Toth P, Vigo D (2002) VRP with backhauls. In: Toth P, Vigo D (eds) The vehicle routing problem. Monographs on discrete mathematics and applications. SIAM, Philadelphia, pp 195–224
172. Toth P, Vigo D (2014) Vehicle routing: problems, methods and applications, 2nd edn. MOS-SIAM series on optimization. SIAM, Philadelphia
173. Vahdani B, Tavakkoli-Moghaddam R, Zandieh M, Razmi J (2012) Vehicle routing scheduling using an enhanced hybrid optimization approach. J Intel Manuf 23(3):759–774
174. Vansteenwegen P, Souffriau W, Berghe GV, Oudheusden DV (2009) A guided local search metaheuristic for the team orienteering problem. Eur J Oper Res 196:118–127
175. Varthanan PA, Murugan N, Kumar GM (2012) A simulation based heuristic discrete particle swarm algorithm for generating integrated production-distribution plan. Appl Soft Comput 12:3034–3050

176. Venkatesan SR, Logendran D, Chandramohan D (2011) Optimization of capacitated vehicle routing problem using PSO. Int J Eng Sci Technol (IJEST) 3(10):7469–7469
177. Vidal T, Crainic TG, Gendreau M, Prins C (2013) Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur J Oper Res 231(1):1–21
178. Vigo D (1996) A heuristic algorithm for the asymmetric capacitated vehicle routing problem. Eur J Oper Res 89(1):108–126
179. Wang T-J, Wu K-J (2012) Adaptive particle swarm optimization based on population entropy for MDVRPTW. In: Proceedings of 2012 2nd international conference on computer science and network technology, Changchun, 29–31 Dec 2012, pp 753–756
180. Wang W, Wu B, Zhao Y, Feng D (2006) Particle swarm optimization for open vehicle routing problem. In: Huang D-S, Li K, Irwin GW (eds) ICIC 2006. LNAI, vol 4114. Springer, Berlin/Heidelberg, pp 999–1007
181. Wang S, Wang L, Yuan H, Ge M, Niu B, Pang W, Liu Y (2009) Study on multi-depots vehicle scheduling problem and its two-phase particle swarm optimization. In: Huang D-S et al (eds) ICIC 2009. LNAI, vol 5755. Springer, Berlin/Heidelberg, pp 748–756
182. Wang Z, Li J, Fan J, Fan C (2010) Research on improved hybrid particle swarm optimization for vehicle routing problem with time windows. In: Proceedings of 2010 international conference on artificial intelligence and computational intelligence, Sanya, 23–24 Oct 2010, pp 179–183
183. Wang B, Wang K, Bao F, Zhang L, Shen L (2012) Mixed climbing particle swarm algorithm in the VRP. In: Proceedings of 2012 second international conference on business computing and global informatization, pp 554–557
184. Wei R, Zhang T, Tang H (2010) An improved particle swarm optimization algorithm for vehicle routing problem with simultaneous pickup and delivery. In: Zhu R et al (eds) ICICA 2010, Part I. CCIS, vol 105. Springer, Berlin/Heidelberg, pp 430–436
185. Xu J, Yan F, Li S (2011) Vehicle routing optimization with soft time windows in a fuzzy random environment. Transp Res E 47:1075–1091
186. Yan F, Xu M, Yu H (2015) The vehicle routing optimization with uncertain demands and traveling time. In: Xu J et al (eds) Proceedings of the ninth international conference on management science and engineering Management. Advances in intelligent systems and computing, vol 362, pp 267–274
187. Yang SY, Wang M, Jiao LC (2004) A quantum particle swarm optimization. In: Proceedings of the 2004 IEEE congress on evolutionary computation, vol 1, pp 320–324
188. Yao B, Yu B, Hu P, Gao J (2016) An improved particle swarm optimization for carton heterogeneous vehicle routing problem with a collection depot. Ann Oper Res 242: 303–320
189. Yusoff M, Ariffin J, Mohamed A (2011) A multi-valued discrete particle swarm optimization for the evacuation vehicle routing problem. In: Tan Y et al (eds) ICSI 2011, Part I. LNCS, vol 6728. Springer, Berlin/Heidelberg, pp 182–193
190. Yusoff M, Ariffin J, Mohamed A (2012) DPSO based on random particle priority value and decomposition procedure as a searching strategy for the evacuation vehicle routing problem. In: Huang T et al (eds) ICONIP 2012, Part III. LNCS, vol 7665. Springer, Berlin/Heidelberg, pp 678–685
191. Yusoff M, Ariffin J, Mohamed A (2015) DPSO based on a min-max approach and clamping strategy for the evacuation vehicle assignment problem. Neurocomputing 148:30–38
192. Zeimpekis VS, Tarantilis CD, Giaglis GM, Minis I (eds) (2007) Dynamic fleet management – concepts, systems, algorithms and case studies. Book series: operations research/computer science interfaces series, vol 38. Springer
193. Zhan Z-H, Zhang J (2009) Discrete particle swarm optimization for multiple destination routing problems. In: Giacobini M et al (eds) EvoWorkshops 2009. LNCS, vol 5484. Springer, Berlin/Heidelberg, pp 117–122
194. Zhang W, Ye J (2010) An improved particle swarm optimization for the multi-depot vehicle routing problem. In: Proceedings of 2010 international conference on E-business and E-government, Guangzhou, 7–9 May 2010, pp 3188–3191

195. Zhang N-Z, Sun G-H, Wu Y-H, Geng, F-H (2009) A modified particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. In: Proceedings of the 7th Asian control conference, Hong Kong, 27–29 Aug 2009, pp 1679–1684
196. Zhang T, Chaovalitwongse WA, Zhang Y (2012) Scatter search for the stochastic travel-time vehicle routing problem with simultaneous pick-ups and deliveries. Comput Oper Res 39:2277–2290
197. Zhang L, Li Y, Fei T, Chen X, Ting G (2014) Research of emergency logistics routing optimization based on particle swarm optimization. In: Patnaik S, Li X (eds) Proceedings of international conference on computer science and information technology. Advances in intelligent systems and computing, vol 255, Springer, pp 415–421. https://doi.org/10.1007/978-81-322-1759-6_48
198. Zhao Y, Li C, Zhang J-L, Ren X, Ren W (2011) Research on vehicle routing problem with stochastic demand based on multi-objective method. In: Huang D-S et al (eds) ICIC 2011. LNCS, vol 6838. Springer, Berlin/Heidelberg, pp 153–161
199. Zhen T, Zhu Y, Zhang Q (2009) A particle swarm optimization algorithm for the open vehicle routing problem. In: Proceedings of 2009 international conference on environmental science and information application technology, Wuhan, 4–5 July 2009, pp 560–563
200. Zhu Q, Qian L, Li Y, Zhu S (2006) An improved particle swarm optimization algorithm for vehicle routing problem with time windows. In: Proceedings of 2006 IEEE congress on evolutionary computation, Vancouver, 16–21 July 2006, pp 1386–1390
201. Zong X, Xiong S, Fang Z (2014) A conflict-congestion model for Pedestrian-vehicle mixed evacuation based on discrete particle swarm optimization algorithm. Comput Oper Res 44: 1–12

Scheduling Heuristics

41

Rubén Ruiz

Contents

Introduction 1198
The Importance of Scheduling 1199
Scheduling Is Hard 1200
The Relevance of Heuristic Scheduling 1200
Notation and Classification of Scheduling Problems 1201
Types of Scheduling Problems and Notation 1202
Scheduling Constraints 1203
Scheduling Objectives 1205
Basic Scheduling Heuristics 1207
Some Simple Dispatching Rules 1207
An Application to the Parallel Machine Problem 1208
Elaborated Dispatching Rules 1209
Advanced Heuristic Methods 1210
The NEH Heuristic for the Permutation Flow Shop and Makespan Criterion 1210
A Brief Introduction to the Shifting Bottleneck Heuristic for the Job Shop Problem 1212
Metaheuristics 1213
Main Concepts of Metaheuristics in Scheduling 1213
Iterated Greedy as an Example of a Simple- and High-Performing Method 1215
Conclusions 1216
Cross-References 1217
References 1217

R. Ruiz (✉)
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politécnica de València, València, Spain
e-mail: rruiz@eio.upv.es

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_44

Abstract

The scheduling of operations over resources is a relevant theoretical and practical problem with applications in many fields and disciplines, including the manufacturing industry. Scheduling problems are as varied as the reality they model. Additionally, some scheduling settings are among the hardest combinatorial problems there are. This is a perfect scenario for heuristic methods where high-quality robust solutions can be obtained in a short amount of time. This chapter concentrates on heuristics for production scheduling problems and summarizes the main results that range from simple rules to advanced metaheuristics. The importance of proper scheduling in practice is first highlighted, along with its difficulty and relevance. A summary of the scheduling notation is also given. Basic scheduling techniques, dispatching rules, combined rules, advanced heuristics, and an introduction to metaheuristics are also summarized in the chapter. While necessarily brief and incomplete, this chapter serves as an introductory point to those interested readers seeking to delve in the vast and rich world of scheduling heuristics. Some pointers to fruitful future research avenues are also provided. A large list of journal articles and monographs are provided as a reference for additional details and study.

Keywords

Scheduling · Dispatching rules · Heuristics · Local search · Metaheuristics

Introduction

In a wide sense, the term scheduling deals with the allocation of tasks to resources over a finite time horizon. These resources are usually limited and therefore, tasks have to share or, more precisely, compete for them. Depending on the type of scheduling problem, there might be interrelated decisions of allocation of tasks to resources, sequencing of the tasks assigned to each resource, assignment of start and finish times to each task, and so on. The objective is to find a schedule that optimizes a given criterion or criteria. As can easily be seen, this is a very broad definition. As detailed in [57, chap. 1] and in [56, chap. 1], scheduling is a decision-making process that is applied to a very wide range of situations in theory and in practice. As a result, resources can encompass from machines in a factory to tools or personnel in a plant or project. Resources might also model operation rooms in a hospital or cores in a computing cluster, etc. Similarly, tasks range from manufacturing operations to be performed in a factory to surgical interventions in an operation room. The applications are virtually limitless. During this chapter the focus is on one of the most studied applications of scheduling, which is manufacturing scheduling [24]. In manufacturing or shop scheduling, tasks commonly model client's orders or jobs, and resources are the machines or production lines needed to manufacture the products. While this field inside scheduling is arguably the most studied, other monographs deal with other areas of scheduling in detail. Such

is the case of Pinedo [56] for scheduling in services, Błażewicz et al. [2] for computer scheduling, Grosche [27] for scheduling problems in the airline industry, Randolph [59] for health-care scheduling, or Briskorn [5] for scheduling in sports. This short list is just an extract of the rich and varied literature on scheduling problems.

A salient characteristic of the manufacturing scheduling scheduling field is the sheer variety of real settings found in practice. It is easy to imagine that the scheduling problem found in a factory that produces cars is quite different from a factory that manufactures light bulbs to name just two random examples. Furthermore, the underlying scheduling problems are extraordinarily difficult from a mathematical perspective as most problems are of a combinatorial nature. This variety and the difficulty of the scheduling settings have resulted in a research field that is fragmented and overly specialized. More often than not, exact mathematical procedures are devised for specific scheduling settings where rigor and mathematical elegance are set as a goal in itself, possibly neglecting the general application or practical relevance of the proposed procedures. This has resulted in the well-known "research gap" between scheduling theory and practice, as recognized initially by Graves [26], Ledbetter and Cox [37], and Ford et al. [19]. All these studies pointed out the scant application of scheduling theory into practice. McKay et al. [45] and Olhager and Rapp [50] reached the same conclusions. Dudek et al. [14], MacCarthy and Liu [41], McKay and Wiers [42], McKay et al. [46], or McKay and Wiers [44] directly addressed, at different levels, this research gap. Different state-of-the-art reviews and studies about scheduling research like those of Reisman et al. [60], Ribas et al. [61], and Ruiz and Vázquez-Rodríguez [67] concluded that real scheduling problems are seldom studied. A more complete overview of these issues is detailed in Chap. 15 of [24].

The Importance of Scheduling

Without doubt, the aforementioned research gap is not to be attributed to a lack of interest in scheduling problems from industry. Good scheduling practice is of paramount importance in the manufacturing industry. Good schedules increase machine utilization, decrease manufacturing costs, and improve the fulfillment of delivery dates to customers ([30, 43, 57], among many others). Modern industries face a number of challenges that make scheduling as important as ever, to name just a few: (1) increasing product variety and/or configurations to reach a wider market, (2) reducing time to market, (3) shortening delivery dates to costumers, (4) high pressure in costs from emerging markets, (5) reducing production lot sizes, and (6) increasing frequency of orders from clients but of smaller size. All of the above are indicative of a transition from a mass manufacturing setting to a mass customization situation in a globalized economy. The optimized scheduling of hundreds, if not thousands, of production orders on a busy production shop floor is a must in order to meet these challenges.

Furthermore, good schedules help to identify production bottlenecks, streamline and stabilize production, minimize raw material shortages, and allow dealing with priorities in production orders. The conclusion is clear: nowadays factories must employ advanced scheduling tools in order to stay competitive amidst today's terribly competitive market.

Scheduling Is Hard

Sadly, scheduling models are remarkably hard to solve. In fact, most scheduling problems belong to the $\mathcal{NP}$-Hard class of computational problems [6]. This is also true for most combinatorial optimization problems in many other fields inside operations research like location, routing, planning, etc. However, it can be argued that most real scheduling problems are particularly hard and in practice they seem to be more difficult to solve than problems in other areas. Take for example the well-known capacitated vehicle routing problem (CVRP) for which extensive literature exists. According to Laporte [36] and to the most powerful exact methods recently proposed for the problem [3], instances of up to 135 clients and 7 vehicles can be solved to optimality. In comparison, for the job shop scheduling problem (to be defined later), a wait of more than 20 years was needed to have small instances of 10 jobs and 10 machines solved to optimality by Brucker et al. [7], and no significant advances have been made for the optimal solution of instances with just 20 jobs. By all accounts, 20 jobs are way below the normal number of jobs being processed at any time in a real industry where hundreds of tasks are being processed on a daily basis.

Furthermore, real problems are much more complex than regular job shops and have many side constraints and generalizations. There is little hope of solving real problems to optimality. At this moment it is important to raise the question of whether optimal solutions are required in practice or not. Scheduling models are just that models of real problems that are, in the best case, good approximations of the modeled reality. Real data is often put into estimates that are fed to the models. In these circumstances it makes little sense to solve an approximated model with estimated data to optimality. While it is important to study theoretical problems and constructs and to propose exact approaches for such settings (as the insight gained from those studies is invaluable), real problems often require a different approach. This approach comes in the form of heuristics. Scheduling heuristics provide quick and fast approximated solutions to even the most intricate scheduling models. Moreover, and as it will be shown, such solutions are very close to optimality, at least in the cases where the optimality gap can be calculated or estimated.

The Relevance of Heuristic Scheduling

So far it has been stated that scheduling problems are as varied as the manufacturing reality they model and among the hardest combinatorial problems there are. This

has resulted in a research gap between the theory and practice of scheduling. Furthermore, there are many practical issues that arise when deploying scheduling systems in manufacturing firms. The study of such scheduling systems is not as rich as the theoretical works. Some references are Framinan and Ruiz, Framinan and Ruiz [21, 22], and Framinan et al. [24, Chapters 13 and 14 and references therein]. To cope with the variety of real settings, general and robust heuristics are preferable to ad hoc exact approaches that are often tailored to specific problems and objectives. In particular, metaheuristics are powerful heuristic templates that, with little instantiation, are capable of solving a wide array of differing problems to almost optimality in reduced CPU times. Still, the development of such methods for practical applications is challenging and requires finesse as well as a deep knowledge of the main constructs and results. Such is the focus of this chapter. Examples from the most noteworthy basic heuristic procedures to the most advanced metaheuristics will be highlighted. Our natural inclination is toward general simple methods that include little to no problem-specific knowledge at all as they are the most suited to practical applications. More general approaches to heuristic and metaheuristic algorithms for scheduling problems are given by Morton and Pentico [48], Hoos and Stützle [29, chap. 9], and Framinan et al. [24].

The rest of this chapter is organized as follows: section "Notation and Classification of Scheduling Problems" formalizes scheduling problems and provides a succinct explanation of the common notation used in the literature. Section "Basic Scheduling Heuristics" presents dispatching rules and simple heuristics for some classical and basic scheduling problems. Two advanced scheduling heuristics are presented in section "Advanced Heuristic Methods." A brief introduction to metaheuristics is provided in section "Metaheuristics," along with the description of a successful and simple methodology. Finally, section "Conclusions" concludes this chapter and some pointers for future research are given.

Notation and Classification of Scheduling Problems

Most production scheduling models define a set N of n jobs with consecutive indexes, i.e., $N = \{1, 2, \ldots, n\}$. The resources are modeled in the form of a set M of m machines indexed as $M = \{1, 2, \ldots, m\}$. All sets are assumed to be independent from each other. Subindexes j and k are used to refer to jobs and i and l to machines. Some basic information associated with jobs and machines is as follows:

- A processing route. For a given job j, the route is an ordered list of machines that have to be visited in order for the job to be completed at the factory. If a job has to visit three machines (e.g., 2, 1, and 6), it is said that it has three tasks. Usually, and following this small example, the task in machine 1 cannot start until the previous task of the same job in machine 2 has finished.
- Processing times. These are commonly denoted as p_j or p_{ij}. This is the amount of time that a given job j needs at machine i. This time is fixed and usually it

cannot be interrupted. Machine i is busy while processing job j and cannot be occupied during this time by any other job.
- Due dates and weights. These are denoted by d_j and w_j, respectively. The due date models the time at which job j must be completed in the factory to be delivered to the customer. The weight captures the relative importance of the job. This data will be used later for the objectives to optimize.

Of course, a wealth of additional data might be needed for the jobs and machines, depending on the real problem to solve. However, and for the sake of brevity, the previous data suffices as a bare minimum. Furthermore, in the course of this chapter, it is assumed that all data is known and deterministic. Most figures are usually represented by nonnegative integers for simplicity. The literature on stochastic scheduling is vast. The interested reader is referred to two recent monographs and the references therein for more details [8, 69]. Additionally, the part on stochastic models in Pinedo [57] is an excellent starting point.

Types of Scheduling Problems and Notation

The literature on production scheduling is mostly partitioned according to how the machines are laid out on the production floor and also on the characteristics of the processing routes of jobs. With this in mind, the main scheduling problems relate to the following scenarios:

- Single machine problems. This is the simplest setting. There is a known number n of jobs, and each one has to be processed on a single machine, following a sequence. Most of the time only a vector p_j with the processing times of the jobs on this single machine is needed.
- Parallel machine problems. Picture the previous case but with m machines disposed in parallel. Jobs still have to be processed by one single machine, but in this case it has to be one out of the m available. Therefore, an assignment problem precedes that of the sequencing problem among all jobs assigned to each machine. The m parallel machines might be identical, uniform, or unrelated. In the case of identical parallel machines, the processing times for the jobs are independent of the machines. In the unrelated case, the processing time of a job j might change depending on which machine it is assigned to. Therefore, a processing time matrix p_{ij} is needed. The uniform parallel machine case is an intermediate situation where machines are faster or slower for all jobs by a factor.
- Flow shops. Here the m machines are disposed in series and all jobs must visit all machines in the same order. This means that the processing route is identical for all jobs and is usually assumed to be $1 \ldots, m$. Jobs are broken down into m tasks with known processing times p_{ij}. As stated before, any task for a given job is unable to start before the preceding task at the previous machine has finished. A flow shop is a very common production machine on production or assembly lines.

- Job shops. Here the main difference from flow shops is that each job has a potentially different processing route. Therefore not all jobs must visit all machines and nor do they necessarily have to be in the same order.
- Open shops. This is a much less common setting in which each job has to visit a given number of the m machines, but the route is open for the scheduling method to determine.
- Hybrid shop environments. Most real problems are actually a combination of the previous shop settings and parallel machines. Instead of having m machines disposed in series, there are m stages where at each stage i, there might be a single machine or m_i parallel machines, be these identical, uniform, or unrelated. This is, by far, the most complex and, at the same time, most general production setting.

Note that the previous short list is a massive simplification. Other production settings include assembly shops, manufacturing cells, flexible manufacturing systems (FMS), batch machines, and so on. In any case, the previous classification is the most employed one. Following this classification, the most common notation was proposed by Rinnooy Kan [63] and is a triplet $\alpha/\beta/\gamma$. The first field α covers the previous scheduling types with the values 1, P, Q, R, F, J, and O for single machine, identical parallel machines, uniform parallel machines, unrelated parallel machines, flow shop, job shop, and open shop problems, respectively. For more advanced layouts, like hybrid shops, there are extensions of this notation, like the one provided by Vignier et al. [80] and reviewed in Ruiz and Vázquez-Rodríguez [67]. The second field β is concerned with the constraints present in the problem, and the last field γ captures the objective function to optimize. These two last fields are succinctly summarized in the following sections.

Scheduling Constraints

As mentioned, there are as many scheduling constraints as product varieties and types of factories. Following Framinan et al. [24], the different scheduling constraints can be classified if they affect the processing of the jobs, operations, transportation, storage, or others:

- Process constraints. In this broad category, it might be mentioned all situations that affect the flow of the operations in the factory. Here one can find job/task precedence constraints ($\beta = \{chains, intree, outtree, tree\}$ depending on whether the precedence constraints form chains, convergent trees, divergent trees, or general trees, respectively). Here job/tasks might be subject to precedences, i.e., some jobs might not start before others have finished. Another frequent processing constraint is the setup times. Setup times model all nonproductive operations that have to be performed at machines after finishing jobs and before starting new ones. Cleaning, fixing, and reconfiguration are common setup operations. Setup times might be anticipatory/non-anticipatory and/or sequence

independent/dependent. As such, S_{ijk} models the amount of time that it is needed to setup machine i after processing job j and before processing job k. The literature on scheduling with all sorts and types of setup times is incredibly large. In parallel and hybrid layouts, jobs might have machine eligibility constraints, indicating that not all available machines can process certain jobs. Machines might also be subject to unavailability periods and/or breakdowns. Jobs might also have more general processing routes requiring stage/machine skipping or recirculation (revisiting one or more stages/machines more than once). This last circumstance is indicated as $\beta = recrc$. There are many more process constraints than the ones mentioned here. Consult Pinedo [57] or Framinan et al. [24, chap. 4] for a more detailed account.

- Operations constraints. On many occasions the operation or task of a job on a given machine is more complicated. For example, tasks might be interrupted arbitrarily to be repeated or resumed later, possibly after having processed a task from a different job in between. Tasks from the same job might also be split to be processed by different parallel machines. Tasks/jobs might be subject to release dates, i.e., $\beta = r_j$ indicates that job j might not start processing at its first machine before time r_j due to, for example, unavailability of raw materials. Apart from release dates, there might be due dates, deadlines and due windows, etc. Another frequent characteristic is that jobs might not be allowed to wait in between machines ($\beta = nwt$). This is common in the processing of prepared foods in order to avoid breaking the cold chain. At the same time, there might be forced minimum waits for a job in between stages (cooling down, drying off), maximum waiting times, waiting windows, or even overlaps between consecutive tasks of a job. Similarly, processing times might be subject to all types of additional situations, like learning, deterioration or position dependency, etc.
- Transportation constraints. It is very common in real industries to have means of transporting unfinished jobs between machines. The list here is endless as one might have from simple manually operated pallet jacks for moving products from machine to machine to forklift trucks or even complex automated guided autonomous vehicles. Conveyor belts, handling robots, and other equipment are very common too. Depending on the scenario, transportation might be modeled as a simple fixed transportation time in between machines to very complex scheduling-routing problems when the transportation is an essential part of the process and transportation equipment is limited and expensive.
- Storage constraints. Similar to transportation constraints, production in course has to be stored somewhere. Again, and depending on the real situation, storage might be of no concern as space might be ample and/or the type of product being produced small and easy to store. However, in other situations the handling of unfinished products might be so challenging as to be a central constraint in the factory. Imagine, for example, the production of commercial airplanes where moving an airplane frame from one processing station to another is far from a trivial matter.

- Other constraints. Apart from machines one might have additional resources, like special tools, fixtures, or even the personnel operating machines. This personnel might have different degrees of skill. In some factories, machines might be able to perform more than one operation in parallel. Manufacturing firms might have more than one production center. Processing and/or setup times might be controllable depending on the resources (i.e., personnel) dedicated to them. There might be timing/shift constraints, safety concerns, etc. As indicated, the list of potential situations in practice is very long.

Scheduling Objectives

Finally, field γ is concerned with the scheduling objectives to optimize. In order to understand these, there is a need for some additional notation:

- C_j is the completion time of job j in the shop. It models the time at which the last task is completed at the last machine.
- L_j is a measure of the lateness of job j with respect to its due date d_j. $L_j = C_j - d_j$.
- T_j is a measure of the tardiness of job j where $T_j = \max\{L_j, 0\}$.

This is just a small extract of the many possibilities. Similarly, earliness can be defined when jobs finish before their due dates. From these definitions the two main groups of objective functions are those related with completion times and those with due dates:

- Completion time based. The most common one is the minimization of the maximum completion time among the jobs, usually referred to as makespan or $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$. Makespan is mainly related to maximizing machine utilization. The second most studied objective is the minimization of the total flow time, defined as $\sum C = \sum_{j=1}^{n} C_j$.
- Due date based. The most common is the minimization of the total tardiness of the jobs $\sum T = \sum_{j=1}^{n} T_j$. However, the weighted version or $\sum wT = \sum_{j=1}^{n} w_j T_j$ is more realistic as being tardy for some important jobs should be considered over being tardy for irrelevant orders.

There are tens, if not hundreds, of potential objectives. A much more detailed list is given in Chapter 5 of Framinan et al. [24]. Furthermore, the reality is often multiobjective. The extension of the γ notation for multiple objectives in scheduling is detailed in T'Kindt and Billaut [73].

Let us now give a simple example of the $F/prmu/C_{\max}$ problem or the permutation flow shop problem with makespan criterion. The example with the total flow time objective or $F/prmu/\sum C$ will be used. There are five jobs and four machines. The processing times are shown in Table 1.

Table 1 Processing times (p_{ij}) of a flow shop problem with five jobs and four machines

Machine (i)	Job (j)				
	1	2	3	4	5
1	29	37	95	85	65
2	30	62	59	11	55
3	27	21	70	62	67
4	2	6	82	80	57

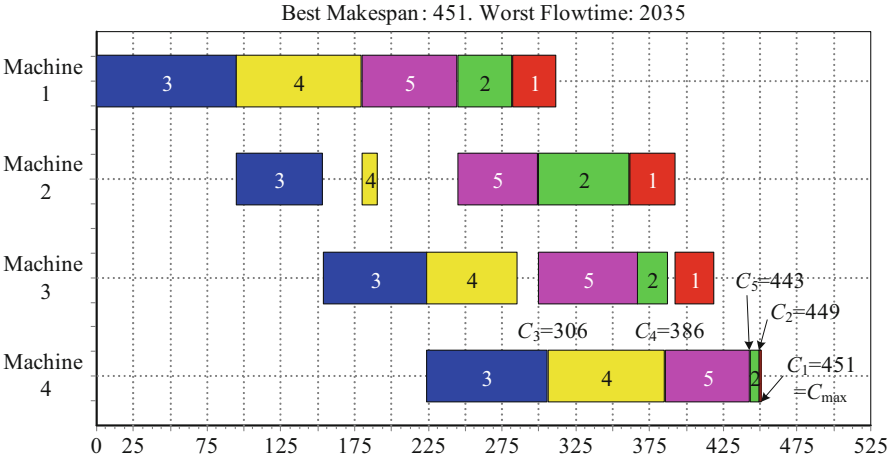


Fig. 1 Gantt chart for the permutation flow shop example with the best $C_{\max}$ which corresponds also with the worst possible $\sum C$ value

The permutation flow shop is a problem of determining a sequence for the jobs so there are $5! = 120$ possible schedules in this small example. If one calculates all of them, the best makespan value we obtain is $C^*_{\max} = 451$ with the permutation or job sequence $\pi^*_{\max} = \{3, 4, 5, 2, 1\}$. The worst makespan value is $C^w_{\max} = 522$ (15.74% worse) resulting from the permutation $\pi^w_{\max} = \{1, 2, 4, 5, 3\}$. What is interesting about this example is that the best flow time value is $\sum C^* = 1{,}464$, resulting from the permutation $\pi^*_{\sum C} = \{1, 2, 4, 5, 3\}$. As one can see, $\pi^w_{\max} = \pi^*_{\sum C}$, i.e., the worst makespan solution is equal to the best flow time solution. Furthermore, the contrary is also true. For this example, the worst flow time value is $\sum C^w = 2,035$ (38.93 % worse than the best), resulting from the permutation $\pi^w_{\sum C} = \{3, 4, 5, 2, 1\}$. As one can see, $\pi^*_{\max} = \pi^w_{\sum C}$. The conclusion is that the best possible makespan value can result in the worst possible flow time and vice versa. The two permutations in this example are depicted in Figs. 1 and 2, respectively. The completion times of all jobs in the last machine have been indicated for further examination. While this is a cherry-picked example, it is easy to find examples such as this for almost any number of jobs and machines.

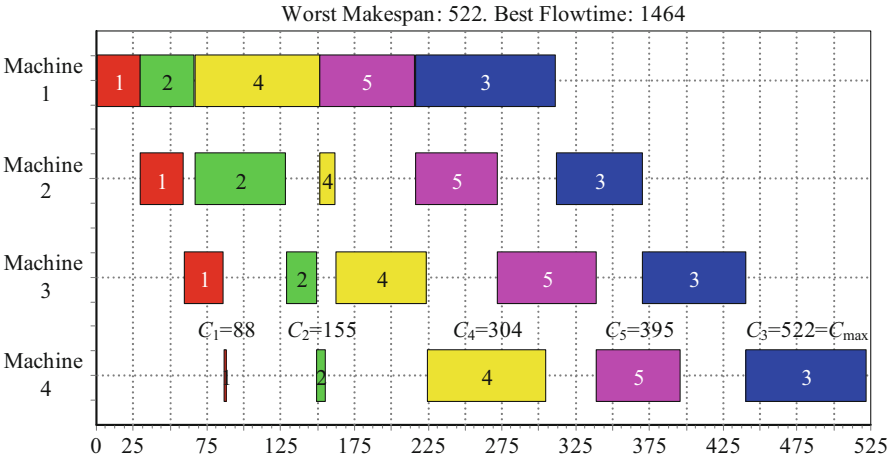


Fig. 2 Gantt chart for the permutation flow shop example with the best $\sum C$ which corresponds also with the worst possible C_{max} value

Basic Scheduling Heuristics

In the early twentieth century, modern production plants started using scheduling techniques to increase productivity. Human schedulers planned production manually without any aid other than pen and paper. The situation today has not changed much for most industries where basically spreadsheets are used to program production, but simple rules are used to obtain effective schedules. These simple rules are often referred to as list scheduling rules, dispatching rules, or priority rules. Some of them will be briefly explained. More advanced dispatching rules will be explained later.

Some Simple Dispatching Rules

Dispatching rules are usually forward heuristics where new tasks or jobs are sequenced as they arrive or as their processing becomes feasible (i.e., all preceding tasks have been already scheduled). A decision is made over a set of eligible tasks, referred to as $\wp$. Some simple dispatching rules are the following:

- First Come First Served (FCFS): Here the job or task that entered the list of eligible tasks first is scheduled, i.e., the eligible tasks are sequenced in a first in first out (FIFO) order.
- Shortest Processing Time (SPT) First: The pending eligible with the shortest processing time goes first, i.e., given m machines, the task k that satisfies the

following expression is chosen: $p_{ik} = \min_{j=1}^{|\wp|} \{p_{ij}\}$. While simplistic, SPT results in the optimal solution for the single machine total flow time problem $(1||\sum C)$. The contrary rule to SPT is the longest processing time (LPT) first rule.

- Earliest Due Date (EDD) First: This is a dispatching rule that considers due dates. Basically the pending task with the earliest due date is scheduled first. EDD is optimal for the problems $1||T_{\max}$ and $1||L_{\max}$.

Note that these dispatching rules are extremely fast; all of them have a computational complexity of $\mathcal{O}(n \log n)$, n being the number of jobs or tasks to be scheduled. They have many other advantages like their simplicity and ease of application. Furthermore, they are easy to understand which is something valued by human schedulers. There are literally hundreds of dispatching rules and they have been comprehensively studied. It is suffice to cite just some of the exiting reviews in Panwalkar and Iskander, Blackstone et al., Haupt, and Jayamohan and Rajendran [4, 28, 31, 55].

An Application to the Parallel Machine Problem

In general, dispatching rules, albeit general, fast, and simple, are overly myopic and produce subpar results, especially when compared to more elaborated constructs. However, in some cases they provide strong solutions. The following example applies to the identical parallel machine problem with makespan criterion or $P//C_{\max}$. In this problem only the assignment of jobs to machines is relevant, as the sequence at each machine has no effect on the maximum completion time. There are n jobs to be scheduled in m identical parallel machines with the only input data being a vector with the processing times p_j. The $P//C_{\max}$ is $\mathcal{NP}$-Hard even for two machines [39].

Real results from an exam scheduling problem of a big company in the USA (Pearson VUE) are presented. This company has 5000 test centers in 175 countries in which all types of certifications, admissions, and exams are taken. The problem solved is the scheduling of different exams every week of a given year at each center in the USA so as to minimize the number of hours that each center is open to the public. Each center has a number of seats in which exams can be taken. Therefore, each center is modelized as an identical parallel machine problem in which the number of seats is the number of machines. The objective is to minimize the makespan as this translates directly into a minimization of the opening hours. The different exams are job types, and registrations for a given exam by users at a given week are the jobs to be scheduled at the given center in that week. All 219 centers in the USA for the 52 weeks of 2012 were solved. This resulted in a total of 11261 different identical parallel machine problems with 1,244,063 jobs in total. The largest center had 308 machines (seats) available, and the biggest number of jobs to solve in a given week and center was 638.

The algorithm of choice is the aforementioned LPT, which for the parallel machine case is simple. Jobs are ordered from longest to shortest processing times, and the first m are assigned to the m parallel machines. Then, each job in the ordered list is assigned to the first freed machine. This LPT is known to have a worst-case performance guarantee of $\frac{4}{3} - \frac{1}{3m}$ [25]. This means that for large m values, the solution given by LPT should be at most around 33 % worse than the optimal makespan. Now the results of the LPT with the solutions obtained by a state-of-the-art solver (CPLEX 12.5) are compared. The experiments were run on a computing cluster with 240 cores in total (XEON E5420 processors running at 2.5 GHz.) and 480 GBytes of RAM memory. Around 93 % of problems could be solved to optimality and the average gap obtained was 0.57 %. However, more than 80 h of CPU time were needed in the cluster (time limit for each problem solved was set to 300 s, which was reached in 851 cases). Comparatively speaking, the LPT resulted in a gap in respect to the MIP optimal solution or lower bound of just 0.91 %. The total CPU time of the application of the LPT for all 11261 problems was just 17.16 s on a single computer and most of this time was the processing of the input data and generation of the solution files.

The conclusion is clear. For some problems, including real ones, the application of simple dispatching rules can result in extraordinary performance in negligible CPU times.

Elaborated Dispatching Rules

Many authors have explored more intricate procedures. A clear example is the apparent tardiness cost (ATC) composite dispatching rule proposed by Vepsalainen and Morton [79]. This rule is excellently explained in Pinedo [57]. The ATC rule was presented for job shop problems with total weighted tardiness criterion but has been successfully applied to different problems as well. The rule is applied to all jobs and an index is calculated. The job with the highest index is scheduled and then the rule is calculated again for the remaining $n - 1$ jobs. In total ATC is computed $n \cdot (n + 1)/2 - 1$ times resulting in a computational complexity of $\mathcal{O}(n^2)$. The index is calculated for each job j, and each time a machine is free as $I_j(t) = \frac{w_j}{p_j} \cdot e^{\left(\frac{\max\{d_j - p_j - t, 0\}}{K\overline{p}}\right)}$ where $\overline{p}$ is the average of the processing times and K is a parameter that needs to be fixed. ATC was expanded for the consideration of setup times by Lee et al. [38] resulting in the apparent tardiness cost with setups or ATCS. Here the index is further complicated: $I_k(t, j) = \frac{w_k}{p_k} \cdot e^{\left(\frac{\max\{d_k - p_k - t, 0\}}{K_1\overline{p}}\right)} \cdot e^{\left(-\frac{s_{jk}}{K_2\overline{s}}\right)}$. As one can easily see, the calculations quickly become cumbersome; the rule is also harder to understand and to apply which basically defeats many of the advantages of dispatching rules. The benefits are better performance and objective function values.

Advanced Heuristic Methods

Advanced heuristics usually include problem-specific knowledge exploiting theoretical properties of scheduling problems. Alternatively, some methods aim at deriving better schedules starting from previous schedules. These latter approaches have been known as improvement heuristics. The distinction between basic or constructive heuristics and improvement heuristics is not so clear on many occasions. In any case, most of the time constructive heuristics have a natural stopping criterion, i.e., the method stops when a complete schedule or sequence has been built. Improvement heuristics don't have such a direct stopping criteria.

Most improvement heuristics and metaheuristics (to be succinctly reviewed in the next section) are strongly based on local search techniques. Local search is a heuristic exploration technique with a strong methodological background ([29], among many others). Local search requires at least the following ingredients: (1) A definition of what is a neighboring solution of a sequence; (2) a way to explore the neighbors of a solution; (3) an acceptance criterion, i.e., when a neighboring solution must replace the current one; and (4) when and how to stop the local search. Usually the main and most important issue is how to represent a solution and how to define a neighborhood from that representation. In most scheduling problems, solutions are indirectly represented as permutations of integers. The most employed neighborhoods are swap and insert where two jobs exchange their positions in the sequence in the case of swap and a job is extracted from its position and inserted into another position in the case of insert. In the following section, a very successful heuristic for the flow shop problem is detailed. It is based on the exploration of the insertion neighborhood in a constructive phase. Later another effective heuristic is briefly described but this time for the job shop problem and based on a different methodology the decomposition of the scheduling problem into smaller subproblems.

The NEH Heuristic for the Permutation Flow Shop and Makespan Criterion

The NEH heuristic of Nawaz et al. [49] is one of the most successful advanced heuristics. It was originally proposed for the permutation flow shop with makespan criterion ($F/prmu/C_{\max}$). It is powerful and capable, with hundreds of applications and references to many scheduling problems. Reviews like those of Turner and Booth [74], Taillard [71] or Ruiz and Maroto [64] recognize its capacity. The NEH exploits the fact that jobs with large total processing times must be sequenced as early as possible. The NEH consists of three steps:

1. The total processing times of the jobs in all machines are calculated: $P_j = \sum_{i=1}^{m} p_{ij}, \quad j = (1, \ldots, n)$.
2. Jobs are sorted in descending order of P_j, so that the first job is the one that has the highest time requirements. These sorted jobs are stored in a list ℓ.

3. The first two jobs of the list ℓ, referred to as $\ell_{(1)}$ and $\ell_{(2)}$, are extracted. The two possible partial sequences made up from these two jobs are constructed and their respective makespan values calculated. In other words, the two following partial sequences are tested: $\{\ell_{(1)}, \ell_{(2)}\}$ and $\{\ell_{(2)}, \ell_{(1)}\}$. From the two sequences, the one with the minimum partial $C_{\max}$ value is retained for the next iterations.
4. The main loop of the NEH starts here. At each iteration, the job at position k of the list ℓ, i.e., $\ell_{(k)}$, $k = (3, \ldots, n)$ is extracted. This job $\ell_{(k)}$ is inserted into all possible solutions of the partial incumbent sequence, and the sequence resulting in the lowest partial $C_{\max}$ value is retained for the next iteration. For example, let us suppose that the best partial sequence obtained up to a given point is $\pi_p = \{\ell_{(1)}, \ell_{(3)}, \ell_{(2)}\}$ and that $k = 4$; therefore, job $\ell_{(4)}$ has to be inserted in the four possible positions of the partial sequence: $\pi^1_{p+\ell_{(4)}} = \{\ell_{(4)}, \ell_{(1)}, \ell_{(3)}, \ell_{(2)}\}$, $\pi^2_{p+\ell_{(4)}} = \{\ell_{(1)}, \ell_{(4)}, \ell_{(3)}, \ell_{(2)}\}$, $\pi^3_{p+\ell_{(4)}} = \{\ell_{(1)}, \ell_{(3)}, \ell_{(4)}, \ell_{(2)}\}$, and $\pi^4_{p+\ell_{(4)}} = \{\ell_{(1)}, \ell_{(3)}, \ell_{(2)}, \ell_{(4)}\}$. For the next step ($k = 5$), the sequence with the lowest partial $C_{\max}$ value from the four previous possible sequences is retained.

As one can see, NEH is basically based upon insertions of jobs into a partial sequence that is constructed step after step. Step 1 has a complexity of $\mathcal{O}(nm)$. The second step is just an ordering and takes $\mathcal{O}(n \log n)$. The third step requires just two calculations of two small sequences and can be disregarded. Most CPU time is needed by step 4. In step 4 there is a loop of $n-2$ iterations where at each step k, k insertions of job $\ell_{(k)}$ are carried out into a partial sequence that contains $k-1$ jobs, and for each insertion one needs to calculate the $C_{\max}$ value of k jobs (including the inserted one). In total, $\frac{n(n+1)}{2} - 3$ insertions are carried out, being n of these complete sequences at the last iteration. Therefore, the computational complexity of the NEH is $\mathcal{O}(n^3 m)$. A closer look shows that when inserting the k-th job in position, let us say, j, all $C_{i,h}$, $h = \{j-1, j-2, \ldots, 1\}$ were already calculated in the previous insertion and there is no need to recalculate. Taillard [71] exploited this idea by which all insertions in a given step can be calculated in $O(nm)$ effectively reducing the complexity of the NEH to $O(n^2 m)$. This faster variant is referred to as NEHT or NEH with Taillard's accelerations.

The NEH was computationally tested against 25 other heuristics, including other more recent and complex heuristics in Ruiz and Maroto [64]. The experiments demonstrated that the NEH is superior to all other tested methods and at the same time much faster. This was supported by statistical analyses. In the results, NEH showed an average percentage deviation from the best solutions known in the 120 instances of Taillard [72] of just 3.33 %. Although this is an average result, NEH rarely goes beyond 7 %. Furthermore, NEH is actively used as a seed sequence for metaheuristic techniques.

The NEHT was also evaluated by Rad et al. [58] with surprising results. For example, in the 10 instances of the largest size (500×20) from the benchmark of Taillard [72], NEHT managed to give an average percentage deviation from the best known solutions (these solutions have been proved to be "almost optimal" with very small gaps between the highest known lower bound and lowest known upper

bound or best solution known) of just 2.24 %. Furthermore, this 2.24 % deviation from a very close to optimum solution was achieved in an average of just 0.0773 s. Extensions of the NEH presented in the same paper of Rad et al. [58] resulted in much better performance, reaching an average percentage deviation from best known solutions of just 1.4 % in CPU times under 2 s.

With these results, it is easy to see why the NEH is such an important and widely studied heuristic. Studies about the NEH have been numerous over the years. The papers of [10, 12, 13, 23, 32–35, 58] and more recently Fernandez-Viagas and Framinan [17] are some examples. Furthermore, the NEH does not rely on specific problem knowledge, and it is relatively easy to extend it and apply it to different scheduling problems. Such extensions are numerous in the scheduling literature, mainly over flow shop problems but also in hybrid settings and other problems.

A Brief Introduction to the Shifting Bottleneck Heuristic for the Job Shop Problem

The shifting bottleneck heuristic or SBH for short was initially proposed by Adams et al. [1] for the job shop problem and makespan minimization ($F//C_{\max}$). Many improvements on the initial SBH for the same problem were presented over the following years. Additionally, SBH has been extended by many authors to other objectives and problems. It is a very interesting method because it mimics the reality of a production floor. In real factories one production stage or machine is usually the bottleneck of the whole plant, i.e., productivity or the measured objective is hard to improve unless the scheduling or capacity of this stage is improved. SBH tries to optimize the schedule of the bottleneck machine first as the contribution of this machine to the final makespan is large. Since in a job shop there are m machines and jobs are made of many tasks with precedence relationships among them, scheduling only the bottleneck machine is not straightforward. The procedure can be simply described as follows:

1. Determine the bottleneck machine.
2. Schedule all previous tasks of the n jobs on the other machines to obtain the release dates (r_j) of the jobs on the bottleneck machine.
3. Schedule all subsequent tasks of the n jobs on the other machines to obtain the due dates (d_j) of the jobs on the bottleneck machine.
4. Schedule the bottleneck machine as a single machine problem with release dates and due dates, minimizing the maximum lateness ($1/r_j, d_j/\max L_j$).
5. Repeat all previous steps for the remaining $m - 1$ machines until all machines have a sequence.

Note that the single machine problem is solved m times. The $1/r_j,d_j/\max L_j$ is in itself a hard problem. However, efficient methods do exist for its solution. In particular, the original SBH heuristic of [1] uses the algorithm of [9]. Additional details about the SBH procedure, including examples, are given in [57] and in [24].

The performance of the SBH is outstanding, especially if one considers that the job shop problem is incredibly hard to solve. Back at the time the authors were able to solve problems of up to 50 jobs and 10 machines to optimality in some cases. As a result, the literature on the SBH is incredibly large. More details about this heuristic and many other related bottleneck decomposition methods are given in the book of [51].

Metaheuristics

While heuristics are usually devised for specific problems with a given objective function, they are, in principle, more generic. This distinction is far from clear as many heuristics are indeed general as well, like the aforementioned NEH. In any case, metaheuristics can be seen as general algorithmic templates that exist without the particular problem or application. Additionally, metaheuristics are commonly more demanding and CPU intensive than regular heuristics. The desired effect of this additional computation is a better solution quality. Other common characteristics of metaheuristics are mechanisms to escape local optimality and to reduce the effect of the myopic nature of heuristic approaches by diversification mechanisms. The field of metaheuristics is interdisciplinary and terribly prolific with literally thousands of papers being published every year in many different fields, not to mention the large number of monographs dedicated to them. Scheduling being an important problem, the application of metaheuristics to scheduling is frequent. A good starting point are the books of [15, 29].

Metaheuristics are roughly grouped into families or classes. Each family stems from a basic metaheuristic template that, with relatively few modifications, can be applied to a specific scheduling problem, and reasonable results might be obtained. To obtain very high-quality solutions, more specific operators and local search schemes are usually needed. This also results in slower methods. The main categories are simulated annealing, tabu search, genetic algorithms, ant colony optimization, scatter search, variable neighborhood search, GRASP and iterated local search, etc. In the following section, the main constructs inside most metaheuristic algorithms are described, to be followed by a brief description of a successful metaheuristic that is at the same time simple: iterated greedy.

Main Concepts of Metaheuristics in Scheduling

Most metaheuristic classes require at least the instantiation of some of the following elements:

- Representation of the solution. At the lowest level, solutions contain variables and these might be continuous, discrete, binary, etc. A representation is just how the variables are represented and arranged. In scheduling problems that require permutations and an array or list of integers, each one representing a job is an

indirect representation of the final solution. Such a solution is "scheduled" (i.e., starting and finish times for each job and/or task are calculated) in order to calculate the desired objective function. For parallel machine problems where only the assignment is relevant (e.g., the $P//C_{\max}$), each job might have an integer indicating the machine to which it is assigned to. The representation is very important for the final performance of the metaheuristic.

- Initial solution or population generation. Most metaheuristics require an initial solution to start the process. Population-based metaheuristics require a set of solutions. The easiest procedure is to randomly generate these initial solutions. Of course, this results in initial solutions of mediocre quality. The most commonly employed alternative is to use high-performance heuristics to generate initial schedules.
- Intensification and diversification mechanisms. Initial solutions are worked on through local search and/or other procedures like selection and crossover in genetic algorithms. The objective is to find new and better solutions. There are many procedures and techniques but all of these fall more or less into what is referred to as intensification. To avoid stagnation in the search process, most metaheuristics include diversification mechanisms by which existing solutions are disrupted or altered (or new solutions are generated) in order to allow the intensification procedures to work over different parts of the solution and search space.
- Acceptance criterion. New solutions generated through the intensification and diversification processes have to be examined in order to decide if they replace current incumbent solutions or not. The simplest scenario is when new solutions are accepted if and only if they improve on the current or best objective function value found so far during the search. This criterion leads to premature convergence and stagnation, and most metaheuristic templates consider more elaborated alternatives where worse solutions might be accepted, often temporarily.
- Termination criterion. Metaheuristics iterate continuously unless a user-given termination criterion is met. For most metaheuristics, optimality is not guaranteed. Furthermore, most applications of metaheuristics do not have an indication of when an optimal solution has been found in order to stop the search. Therefore, common termination criteria include a number of iterations or a consecutive number of iterations without improvements (or small improvements), elapsed CPU time, etc.

The previous operators are only a very small list where not even all of them might be needed to construct a simple method. Furthermore, elaborated metaheuristics might have many operators and procedures. Actually, in the last decade, many natural or man-made processes have inspired a real tsunami of metaheuristic methods. As a result there are algorithms that are referred to as particle swarm, imperialist competitive, honey bees, fruit flies, and even more bizarre methods like intelligent water drops or galaxy optimization. Currently, such "novel" methods are being criticized. The papers of [70, 81] are two examples. In this chapter these criticisms are supported and very successful, but at the same time really simple metaheuristic is described.

Iterated Greedy as an Example of a Simple- and High-Performing Method

The first application of the concepts of iterated greedy (IG) to a scheduling problem was presented by [65]. The authors dealt with the permutation flow shop problem with makespan criterion ($F/prmu/C_{\max}$). Being a metaheuristic, IG starts from an initial heuristic solution. Then three phases are iteratively applied until a termination criterion is met. The first phase is a partial destruction of the current solution. Some jobs are randomly removed from the permutation. The second phase is the reconstruction of the (now) incomplete solution. All removed jobs are inserted, one by one, into all positions in the partial solution using the NEH heuristic. Each job is placed into the position resulting in the best objective function value. An optional step is a local search each time a complete solution is obtained. This local search is carried out in the insertion neighborhood until local optimality. The third last step is an acceptance criterion. A detailed pseudoalgorithm listing is given in Fig. 3.

As one can see, IG carries out intensification with the reconstruction and local search operators and diversification with the destruction. The acceptance criterion is a simplification of simulated annealing. With a constant temperature factor, worse solutions are accepted with a probability that decreases with the quality of the new solution.

The results of the IG method for the permutation flow shop problem are state-of-the-art even today. According to the results of [65], IG was able to reduce the average relative percentage deviation from the best known solutions in the 120 instances of [72] to just 0.44 % in little under a minute on average. It was computationally and statistically demonstrated that IG produced better results than competing methods. Later, [78] were able to further reduce this deviation to 0.22 % using a simple island model parallel version of the IG.

IG has been applied to different scheduling problems with good results. Ruiz and Stützle [66] extended the previous results with the consideration of sequence-dependent setup times. IG for no wait and blocking flow shops were studied by [54,62], respectively. IG showed excellent performance in no-idle and mixed no-idle flow shops in [68] and [53]. Non-permutation flow shops were approached by [83]. Distributed scheduling problems have been solved with IG methods by [18, 40]. IG also gives good results in other objectives aside from makespan like tardiness in [20] or total flow time in [52], to name just a few. Multi-objective extensions of IG have proven effective for Pareto flow shops without and with setups in [47] and in [11], respectively. Parallel machine problems were successfully solved with IG methods by [16]. Finally, hybrid flow shops were studied by [82] and complex realistic hybrid flow shops by [75–77]. There are many other applications of the IG methodology to other scheduling problems and also to several other combinatorial problems as well.

What is noteworthy about all previous studies is that IG algorithms are shown to produce state-of-the-art results that on many occasions are competitive or better than those produced by other much more complex metaheuristics.

procedure *Iterated_Greedy*

$\pi :=$ NEH_Heuristic;

$\pi :=$ Insertion_LocalSearch(π);

$\pi_b := \pi$;

while (termination criterion not satisfied) **do**

$\pi' := \pi$;

for $i := 1$ **to** *destruct* **do** % `Destruction phase`

$\pi' :=$ randomly extract a job of π' and insert it into $\pi_R^{'}$;

for $i := 1$ **to** *destruct* **do** % `Reconstruction phase`

$\pi' :=$ best permutation after inserting $\pi_R(i)$ into all possible positions of π';

$\pi'' :=$ Insertion_LocalSearch(π'); % `Local Search`

if $C_{\max}(\pi'') < C_{\max}(\pi)$ **then** $\pi := \pi''$; % `Acceptance criterion`

if $C_{\max}(\pi) < C_{\max}(\pi_b)$ **then** $\pi_b := \pi$;

elseif $\left(random \leq e^{-\frac{(C_{\max}(\pi'') - C_{\max}(\pi))}{\text{Temperature}}}\right)$ **then** $\pi := \pi''$;

endif

endwhile

return π_b

end

Fig. 3 Iterated greedy (IG) algorithm (Adapted from [65]). $random$ is a random number distributed uniformly in the range [0, 1]. Temperature and *destruct* are the two parameters of IG

There are highly successful applications of different metaheuristics for practically every formulated scheduling problem. Many are listed in [24] and the references therein.

Conclusions

In this chapter the scheduling field has been introduced, pointing out its relevance, difficulty, and hardness. Scheduling problems are varied, hard, and difficult to solve optimally. However, in many settings, obtaining approximated solutions of

sufficient quality is within reach if modern heuristic methodologies are employed. The notation of scheduling problems has been introduced. A succinct explanation of simple dispatching rules is provided. An example that shows the lack of correlation between otherwise seemingly correlated objectives has been presented which again highlights the fact that scheduling problems are neither easy nor intuitive. Furthermore, in a briefly summarized application, it has been shown how a very simple dispatching rule can result in almost optimal solutions for real problems of a large size. Even though elaborated and combined dispatching rules are powerful, advanced heuristic methods are many times better. The NEH method for the flow shop and the SBH for the job shop are the clearest examples. High-performance and close-to-optimal solutions are perfectly possible in negligible CPU times. Finally, in an ever-growing field, metaheuristics propose themselves as modern heralds of flexibility, robustness, and finest solution quality. A summary of the iterated greedy method has been presented which apart from the previous qualities manages state-of-the-art results without unnecessary complexities in its design.

As for the future of scheduling heuristics, it is argued for even simpler and more robust approaches, capable of solving hard and varied scheduling problems with little to no instantiation and adaptation. Such frameworks and methodologies could prove invaluable in closing the gap between the theory and practice of scheduling.

Cross-References

▸ Iterated Greedy
▸ Iterated Local Search

Acknowledgments The author is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds.

References

1. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manag Sci 34(3):391–401
2. Błażewicz J, Ecker KH, Pesch E, Schmidt G, Węglarz J (2001) Scheduling computer and manufacturing processes, 2nd edn. Springer, Berlin
3. Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. Oper Res 59(5):1269–1283
4. Blackstone JH Jr, Phillips DT, Hogg GL (1982) A state-of-the-art survey of dispatching rules for manufactuing job shop operations. Int J Prod Res 20(1):27–45
5. Briskorn D (2008) Sports leagues scheduling. Models, combinatorial properties, and optimization algorithms. Lecture notes in economics and mathematical systems, vol 603. Springer, Berlin/Heidelberg
6. Brucker P (2007) Scheduling algorithms, 5th edn. Springer, New York
7. Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discret Appl Math 49(1–3):107–127

8. Cai X, Wu X, Zhou X (2014) Optimal stochastic scheduling. International series in operations research & management science, vol 207. Springer, New York
9. Carlier J (1982) The one-machine sequencing problem. Eur J Oper Res 11(1):42–47
10. Chakraborty UK, Laha D (2007) An improved heuristic for permutation flowshop scheduling. Int J Inf Commun Technol 1(1):89–97
11. Ciavotta M, Minella G, Ruiz R (2013) Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study. Eur J Oper Res 227(2):301–313
12. Dong XY, Huang HK, Chen P (2006) A more effective constructive algorithm for permutation flowshop problem. In: Intelligent data engineering and automated learning (IDEAL 2006). lecture notes in computer science, vol 4224. Springer, Berlin/New York (Beijing Jiaotong Univ, Sch Comp & IT, Beijing 100044, Peoples R China), pp 25–32
13. Dong XY, Huang HK, Chen P (2008) An improved NEH-based heuristic for the permutation flowshop problem. Comput Oper Res 35(12):3962–3968
14. Dudek RA, Panwalkar SS, Smith ML (1992) The lessons of flowshop scheduling research. Oper Res 40(1):7–13
15. El-Ghazali T (2009) Metaheuristics: from design to implementation. Wiley, New York
16. Fanjul-Peyro L, Ruiz R (2010) Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 207(1):55–69
17. Fernandez-Viagas V, Framinan JM (2014) On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Comput Oper Res 45:60–67
18. Fernandez-Viagas V, Framinan JM (2015) A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. Int J Prod Res 53(4):1111–1123
19. Ford FN, Bradbard DA, Ledbetter WN, Cox JF (1987) Use of operations research in production management. Prod Invent Manag 28(3):59–62
20. Framinan JM, Leisten R (2008) Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm. Int J Prod Res 46(22):6479–6498
21. Framinan JM, Ruiz R (2010) Architecture of manufacturing scheduling systems: literature review and an integrated proposal. Eur J Oper Res 205(2):237–246
22. Framinan JM, Ruiz R (2012) Guidelines for the deployment and implementation of manufacturing scheduling systems. Int J Prod Res 50(7):1799–1812
23. Framinan JM, Leisten R, Rajendran C (2003) Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. Int J Prod Res 41(1):121–148
24. Framinan JM, Leisten R, Ruiz R (2014) Manufacturing scheduling systems. An integrated view on models, methods and tools. Springer, New York
25. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17(2):416–429
26. Graves SC (1981) A review of production scheduling. Oper Res 29(4):646–675
27. Grosche T (2009) Computational intelligence in integrated airline scheduling. Studies in computational intelligence, vol 173. Springer, New York
28. Haupt R (1989) A survey or priority rule-based scheduling. OR Spectr 11(1):3–16
29. Hoos HH, Stützle T (2005) Stochastic local search: foundations and applications. Morgan Kaufmann, San Francisco
30. Hopp WJ, Spearman ML (2011) Factory physics, 3rd edn. Waveland Press Inc., Long Grove
31. Jayamohan MS, Rajendran C (2000) New dispatching rules for shop scheduling: a step forward. Int J Prod Res 38(3):563–586
32. Jin F, Song SJ, Wu C (2007) An improved version of the NEH algorithm and its application to large-scale flow-shop scheduling problems. IIE Trans 39(2):229–234
33. Kalczynski PJ, Kamburowski J (2007) On the NEH heuristic for minimizing the makespan in permutation flow shops. OMEGA Int J Manag Sci 35(1):53–60
34. Kalczynski PJ, Kamburowski J (2008) An improved NEH heuristic to minimize makespan in permutation flow shops. Comput Oper Res 35(9):3001–3008

35. Kalczynski PJ, Kamburowski J (2009) An empirical analysis of the optimality rate of flow shop heuristics. Eur J Oper Res 198(1):93–101
36. Laporte G (2009) Fifty years of vehicle routing. Transp Sci 43(4):408–416
37. Ledbetter WN, Cox JF (1977) Operations research in production management: an investigation of past and present utilisation. Prod Invent Manag 18(3):84–91
38. Lee YH, Bhaskaran K, Pinedo ML (1997) A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Trans 29(1):45–52
39. Lenstra JK, Rinnooy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discret Math 1:343–362
40. Lin SW, Ying KC, Huang CY (2013) Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. Int J Prod Res 51(16):5029–5038
41. MacCarthy BL, Liu J (1993) Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. Int J Prod Res 31(1):59–79
42. McKay KN, Wiers VCS (1999) Unifying the theory and practice of production scheduling. J Manuf Syst 18(4):241–255
43. McKay KN, Wiers VCS (2004) Practical production control. A survival guide for planners and schedulers. J. Ross Publishing Inc., Boca Raton
44. McKay KN, Wiers VCS (2006) The human factor in planning and scheduling, chap 2. In: Herrmann JW (ed) Handbook of production scheduling. International series in operations research & management science, vol 89. Springer, New York, pp 23–57
45. McKay KN, Safayeni FR, Buzacott JA (1988) Job-shop scheduling theory: what is relevant? Interfaces 4(18):84–90
46. McKay KN, Pinedo ML, Webster S (2002) Practice-focused research issues for scheduling systems. Prod Oper Manag 11(2):249–258
47. Minella G, Ruiz R, Ciavotta M (2011) Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. Comput Oper Res 38(11):1521–1533
48. Morton TE, Pentico DW (1993) Heuristic scheduling sysmtems with applications to production systems and project management. Wiley series in engineering & technology management. Wiley, Hoboken
49. Nawaz M, Enscore EE Jr, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA Int J Manag Sci 11(1):91–95
50. Olhager J, Rapp B (1995) Operations research techniques in manufacturing planning and control systems. Int Trans Oper Res 2(1):29–43
51. Ovacik IM, Uzsoy R (1997) Decomposition methods for complex factory scheduling problems. Springer, New York
52. Pan QK, Ruiz R (2012) Local search methods for the flowshop scheduling problem with flowtime minimization. Eur J Oper Res 222(1):31–43
53. Pan QK, Ruiz R (2014) An effective iterated Greedy algorithm for the mixed no-idle flowshop scheduling problem. OMEGA Int J Manag Sci 44(1):41–50
54. Pan QK, Wang L, Zhao BH (2008) An improved iterated Greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. Int J Adv Manuf Technol 38(7–8):778–786
55. Panwalkar SS, Iskander W (1977) A survey of scheduling rules. Oper Res 25(1):45–61
56. Pinedo ML (2009) Planning and scheduling in manufacturing and services, 2nd edn. Springer, New York
57. Pinedo ML (2012) Scheduling: theory, algorithms and systems, 4th edn. Springer, New York
58. Rad SF, Ruiz R, Boroojerdian N (2009) New high performing heuristics for minimizing makespan in permutation flowshops. OMEGA Int J Manag Sci 37(2):331–345
59. Randolph H (ed) (2012) Handbook of healthcare system scheduling. International series in operations research & management science, vol 168. Springer, New York
60. Reisman A, Kumar A, Motwani J (1997) Flowshop scheduling/sequencing research: a statistical review of the literature, 1952–1994. IEEE Trans Eng Manag 44(3):316–329
61. Ribas I, Leisten R, Framinan JM (2010) Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput Oper Res 37(8):1439–1454

62. Ribas I, Companys R, Tort-Martorell X (2011) An iterated greedy algorithm for the flowshop scheduling problem with blocking. OMEGA Int J Manag Sci 39(3):293–301
63. Rinnooy Kan AHG (1976) Machine scheduling problems: classification, complexity and computations. Martinus Nijhoff, The Hague
64. Ruiz R, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165(2):479–494
65. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177(3):2033–2049
66. Ruiz R, Stützle T (2008) An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 187(3):1143–1159
67. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flowshop scheduling problem. Eur J Oper Res 205(1):1–18
68. Ruiz R, Vallada E, Fernández-Martínez C (2009) Scheduling in flowshops with no-idle machines. In: Chakraborty UK (ed) Computational intelligence in flow shop and job shop scheduling. Studies in computational intelligence, vol 230. Springer, Berlin, pp 21–51
69. Sarin SC, Nagarajan B, Liao L (2014) Stochastic scheduling. Expectation-variance analysis of a schedule. Cambridge University Press, Cambridge
70. Sörensen K (2015) Metaheuristics – the metaphor exposed. Int Trans Oper Res 22(1):3–18
71. Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47(1):67–74
72. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285
73. T'Kindt V, Billaut JC (2006) Multicriteria scheduling: theory, models and algorithms, 2nd edn. Springer, New York
74. Turner S, Booth D (1987) Comparison of heuristics for flow shop sequencing. OMEGA Int J Manag Sci 15(1):75–78
75. Urlings T, Ruiz R (2007) Local search in complex scheduling problems. In: Stützle T, Birattari M, Hoos HH (eds) Engineering stochastic local search algorithms. Designing, implementing and analyzing effective heuristics. Lecture notes in computer science, vol 4638. Springer, Brussels, pp 202–206
76. Urlings T, Ruiz R, Sivrikaya-Şerifoğlu F (2010) Genetic algorithms for complex hybrid flexible flow line problems. International Journal of Metaheuristics 1(1):30–54
77. Urlings T, Ruiz R, Stützle T (2010) Shifting representation search for hybrid flexible flowline problems. European Journal of Operational Research 207(2):1086–1095
78. Vallada E, Ruiz R (2009) Cooperative metaheuristics for the permutation flowshop scheduling problem. Eur J Oper Res 193(2):365–376
79. Vepsalainen APJ, Morton TE (1987) Priority rules and lead time estimation for job shop scheduling with weighted tardiness costs. Manag Sci 33(8):1036–1047
80. Vignier A, Billaut JC, Proust C (1999) Les problèmes d'ordonnancement de type flow-shop hybride: État de l'art. RAIRO Recherche opérationnelle 33(2):117–183 (in French)
81. Weyland D (2010) A rigorous analysis of the harmony search algorithm: how the research community can be misled by a "novel" methodology. Int J Appl Metaheuristic Comput 1(2):50–60
82. Ying KC (2008) An iterated Greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. IEEE Trans Evol Comput 60(6):810–817
83. Ying KC (2008) Solving non-permutation flowshop scheduling problems by an effective iterated Greedy heuristic. Int J Adv Manuf Technol 38(3–4):348–354

Selected String Problems

42

Christian Blum and Paola Festa

Contents

Introduction to String Problems . . . 1222
Creating Diagnostic Probes for Bacterial Infections . . . 1222
Primer Design . . . 1223
Discovering Potential Drug Targets . . . 1223
Motif Search . . . 1223
Notation . . . 1224
The Closest String Problem (CSP) . . . 1224
The Close to Most String Problem (CTMSP) . . . 1228
The Farthest String Problem (FSP) . . . 1229
The Far From Most String Problem (FFMSP) . . . 1230
A Simple ILP-Based Heuristic . . . 1232
Experimental Evaluation . . . 1233
Conclusions . . . 1238
Cross-References . . . 1239
References . . . 1239

Abstract

This chapter overviews some string selection and comparison problems, with special emphasis on the optimization and operational research perspective. It also proposes a simple and efficient ILP-based heuristic that can be used for any of the considered problems.

C. Blum (✉)
Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), Bellaterra, Spain
e-mail: christian.blum@iiia.csic.es

P. Festa
Department of Mathematics and Applications "Renato Caccioppoli", University of Napoli FEDERICO II, Napoli, Italy
e-mail: paola.festa@unina.it

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_58

Keywords
Bio-informatics · Optimization · String problems · Strings consensus

Introduction to String Problems

Among string selection and comparison problems, there is a class of problems known as *strings consensus*, where a finite set of strings is given and one is interested in finding their consensus, i.e., a new string that agrees as much as possible with all the given strings. In other words, the objective is to determine a string called consensus, because it represents—in some way—all the given strings. The idea of representation and of being in consensus can be related to several different objectives listed in the following:

(i) the consensus is a new string whose total distance from all given strings is minimum (*closest string problem*);
(ii) the consensus is a new string close to most of the given strings (*close to most string problem*);
(iii) the consensus is a new string whose total distance from all given strings is maximum (*farthest string problem*);
(iv) the consensus is a new string far from most of the given strings (*far from most string problem*).

Computational intractability of the general strings consensus problem was first proved in 1997 by Frances and Litman [7] and in 1999 by Sim and Park [28].

As a constring of the linear coding of DNA and proteins, many molecular biology problems have been formulated as computational optimization problems involving strings and sequences. Biological applications of computing distance/proximity among strings occur mainly in two varieties. Some require that a region of similarity be discovered, while other applications use the reverse complement of the region, such as designing probes or primers. In the following, some relevant biological applications are outlined.

Creating Diagnostic Probes for Bacterial Infections

Probes are strands of either DNA or RNA that have been modified (i.e., made either radioactive or fluorescent) so that their presence can be easily detected. One possible application of string problems arises in creating diagnostic probes for bacterial infections [14, 21]. In this scenario, given a set of DNA strings from a group of closely related pathogenic bacteria, the task is to find a substring that occurs in each of the bacterial strings (as close as possible) without occurring in the host's DNA. Probes are then designed to hybridize to these target strings, so that the detection of their presence indicates that at least one bacterial species is likely to be present in the host.

Primer Design

Primers are short strings of nucleotides designed such that they hybridize to a given DNA string or to all of a given set of DNA strings with the aim of providing a starting point for DNA strand synthesis by polymerase chain reaction (PCR). The hybridization of primers depends on several conditions, including some thermodynamic rules, but it is largely influenced by the number of mismatching positions among the given strings, and this number should be as small as possible [9, 10, 17].

Discovering Potential Drug Targets

Another biological application of string selection and comparison problems is related to discovering potential drug targets. Given a set of strings of orthologous genes from a group of closely related pathogens and a host (such as a human, crop, or livestock), the goal is to find a string fragment that is more conserved in all or most of the pathogens strings but not as conserved in the host. Information encoded by this fragment can then be used for novel antibiotic development or to create a drug that harms several pathogens with minimal effect on the host. All these applications reduce to the task of finding a pattern that, with some error, occurs in one set of strings (closest string problem) and/or does not occur in another set (farthest string problem). The far from most string problem can help to identify a string fragment that distinguishes the pathogens from the host, so the potential exists to create a drug that harms several but not all pathogens [9, 10, 17].

Motif Search

A motif is a string that occurs *approximately preserved* as a substring in some/-several of the DNA strings of a given set. Approximately preserved means that the motif occurs with changes in at most t positions for a fixed nonnegative integer t. The importance of a motif lies in its characteristic of being a candidate for substrings of noncoding parts of the DNA string that have functions related to, e.g., gene expression [9, 10].

For most consensus problems, Hamming distance is used instead of other alternative measures, such as, for example, the editing distance. The biological reasons justifying this choice are very well described and motivated by Lanctot et al. in [14–16] and can be summarized claiming that the "edit distance is more suitable to measure the *amount* of change that has happened, whereas the Hamming distance is more suitable to measure the *effect* of that change."

The remainder of this chapter is organized as follows. The next section lists notation and definitions used throughout the paper. The following four sections are devoted to the closest string, the close to most string, the farthest string, and the

far from most string problem, respectively. All these problems are mathematically formulated and their properties analyzed. The most popular solution techniques for them are surveyed, along with the computational results obtained and analyzed in the literature. The second last section reports computational results which demonstrate empirically the efficiency of the state-of-the-art algorithms. Concluding remarks and future directions are discussed in the last section.

Notation

Throughout this chapter, the following notation and definitions will be used:

- An *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$ is a finite set of elements, called *characters*.
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ denotes a string of m characters (that is, of length m) over alphabet Σ, i.e., $s^i_j \in \Sigma,\ j = 1, 2, \ldots, m$.
- Given two strings s^i and s^l on Σ such that $|s^i| = |s^l|$, $d_H(s^i, s^l)$ denotes their Hamming distance and is given by

$$d_H(s^i, s^l) = \sum_{j=1}^{|s^i|} \Phi(s^i_j, s^l_j), \tag{1}$$

  where s^i_j and s^l_j denote the character at position j in string s^i and in string s^l, respectively, and $\Phi : \Sigma \times \Sigma \to \{0, 1\}$ is a predicate function such that

$$\Phi(a, b) = \begin{cases} 0, \text{ if } a = b; \\ 1, \text{ otherwise.} \end{cases}$$

- For all consensus problems, each string s of length m over Σ is a valid solution.

The Closest String Problem (CSP)

Given a finite set of strings Ω on Σ, the problem is to find a *center string* $s^* \in \Sigma^m$ such that the Hamming distance between s^* and all strings in Ω is minimal; in other words, s^* is a string to which a minimal value d corresponds such that

$$d_H(s^*, s^i) \leq d, \quad \forall\ s^i \in \Omega.$$

The closest string problem can be formulated as an integer linear program (ILP). In fact, let $\Sigma_k \subseteq \Sigma$ be the set of characters appearing at position k in any of the strings from Ω.

For each $k = 1, 2, \ldots, m$ and $j \in V_k$, let us define the following binary variables:

$$x_{ck} = \begin{cases} 1, \text{ if character } c \in \Sigma_k \text{ is used at position } k \text{ of the solution;} \\ 0, \text{ otherwise.} \end{cases}$$

Then, the CSP admits the following ILP formulation:

$$\min d \tag{2}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{3}$$

$$m - \sum_{k=1}^{m} x_{s_k^i k} \leq d \quad \text{for } i = 1, 2 \ldots, n \tag{4}$$

$$d \in \mathbb{N}^+, \tag{5}$$

$$x_{ck} \in \{0, 1\}, \qquad \text{for } k = 1, 2, \ldots, m, \ \forall\, c \in \Sigma_k. \tag{6}$$

Equalities (3) guarantee that only one character is selected for each position $k \in \{1, 2, \ldots, m\}$. Inequalities (4) impose that if a character in a string s^i is not in the solution defined by the x-variables, then that character will contribute to increasing the Hamming distance from solution x to s^i. Finally, (5) forces d assume a nonnegative integer value and (6) define the decision variables.

This problem was first studied in the area of coding theory [27] and has been independently proved computationally intractable in [7, 15, 16].

In 2004, Meneses et al. [26] used a linear relaxation of the above-described mathematical model to design a branch and bound algorithm. At each iteration, the next node in the branching tree to be explored is the one with the smallest linear relaxation objective function value (also known as the *best-bound first strategy*). Once selected the next node and obtained an optimal fractional solution $x'_{ck} \in [0, 1]$, $k = 1, \ldots, m$, $\forall\, c \in \Sigma_k$, for the linear relaxation of the corresponding subproblem, the algorithm branches on the fractional variable x_{ck} with maximum value of x'_{ck}. The bounding phase is very important in any branch and bound algorithm: the better is the computed bound, the smaller is the number of nodes that need to be explored and that can therefore be pruned. For the bounding phase, Meneses et al. computed an initial bound selecting one of the given input strings and modifying it until a local optimal solution is found. The authors have empirically shown that their branch and bound algorithm is able to solve in reasonable running times small-size instances with 10–30 strings, each of which is 300–800 characters long.

Further exact methods proposed for the CSP are fixed-parameter algorithms [11, 20, 30] that are applicable only when the maximum Hamming distance among all pairs of strings is small. In fact, since these algorithms are designed to solve the decision version of the problem, it is necessary to apply them multiple times in order to find an optimal solution that minimizes the Hamming distance.

The first approximation algorithm for the CSP was proposed in [16] with a worst-case performance ratio of 2. It is a simple algorithm that constructs an approximate feasible solution in a pure random fashion. Starting from an empty solution, the algorithm selects at random the next element to be added to the solution under construction.

Better performance approximation algorithms proposed in the literature are based on the linear programming relaxation of the previously described ILP model. The basic idea consists in solving the linear programming relaxation of the ILP model and in using the result of the relaxed problem to find an approximate solution to the original problem. Following this line, [16] also proposed a $\frac{4}{3}(1+\epsilon)$-approximation algorithm (for any small $\epsilon > 0$) that uses the randomized rounding technique for obtaining an integer 0–1 solution from the continuous solution for the relaxed problem. The randomized rounding technique works by defining the value of a Boolean variable $x \in \{0, 1\}$ to be $x = 1$ with a certain probability y, where y is the value of the continuous variable corresponding to x in the relaxation of the original integer programming problem. In 1999, Li et al. [17] used the rounding idea to design a polynomial-time approximation scheme (PTAS). A PTAS is a special type of approximation algorithm that, for each $\epsilon > 1$, yields a performance guarantee of ϵ in polynomial time. Thus, this can be viewed as a way of getting solutions with guaranteed performance, for any desired threshold greater than one. The PTAS proposed in [17] is also based on randomized rounding that here is refined to check results for a large (but polynomially bounded) number of subsets of indices. However, since a large number of iterations involve the solution of a linear relaxation of an ILP model, the algorithm becomes impractical for any instance with large strings. To efficiently deal with real-world scenarios and/or medium- to large-sized problem instances, several heuristic and metaheuristic algorithms have been proposed in the last few years.

In 2005, Liu et al. [18] designed a genetic algorithm and a simulated annealing algorithm, both in their sequential and their parallel versions. Genetic algorithms (GAs) are population-based metaheuristics that have been applied to find optimal or near-optimal solutions to combinatorial optimization problems [8, 12]. They implement the concept of *survival of the fittest* making an analogy between a solution and an *individual* in a *population*. Each individual has a corresponding *chromosome* that encodes the solution. A chromosome consists of a string of *genes*. Each gene can take on a value, called an *allele*, from some alphabet. A chromosome has an associated *fitness level* which is correlated to the corresponding objective function value of the solution it encodes. Over a number of iterations, called *generations*, GAs evolve a population of chromosomes. This is usually accomplished by simulating the process of natural selection through mating and mutation. For the CSP, starting from an initial randomly generated population, at each generation $0 \leq t \leq \texttt{number-generations}$ of the Liu et al.'s GA, a population $P(t)$ of `popsize` strings of length m is evolved, and the fitness function to be maximized is defined as the difference $m - d_{\max}$, where $d_{\max}$ is the largest Hamming distance between an individual of the population $P(t)$ and any string in Ω. The production of offspring in a GA is done through the process of mating or

crossover, and Liu et al. used a multipoint crossover (MPX). In more detail, at a generic generation t, two parental individuals x and y in $P(t)$ are randomly chosen according to a probability which is proportional to their fitness. Then, iteratively until the offspring is not complete, x and y exchange parts between two randomly picked points. The resulting offspring have a new order of the strings, one part from the *first parent* and the other part from the *second parent*. Afterward, a mutation of any individual in the current population $P(t)$ is executed with some given probability. During this phase, two positions are randomly chosen and exchanged in the individual.

In their paper, Liu et al. proposed also a simulated annealing (SA) algorithm for the CSP. Originally proposed in [13], in the optimization and computer science research communities, simulated annealing is commonly said to be the "oldest" among the metaheuristics and surely one of the first techniques that had an explicit strategy to escape from local minima. Its fundamental idea is to allow moves resulting in solutions of worse quality in terms of objective function value than the current solution (uphill moves) in order to escape from local minima. The origin of simulated annealing and the choice of the acceptance criterion of a better quality solution lie in the physical annealing process that can be modeled by methods based on Monte Carlo techniques. One of the early Monte Carlo techniques for simulating the evolution of a solid in a heat bath to thermal equilibrium is due to [24], who in 1953 designed a method that iteratively (until a stopping criterion is met) generates a string of states of the solid in the following way. At a generic iteration k, given a current state i of the solid (i.e., a current solution x),

- E_i is the energy of the solid in state i (objective function value $f(x)$).
- a subsequent state j (solution $\overline{x}$) is generated with energy E_j (objective function value $f(\overline{x})$) by applying a *perturbation mechanism* such as displacement of a single particle ($\overline{x}$ is a solution "close" to x);
- if $E_j - E_i < 0$ (i.e., $\overline{x}$ is a better quality solution), j ($\overline{x}$) is accepted; otherwise, j ($\overline{x}$) is accepted with probability given by

$$\exp\left(-\frac{E_j - E_i}{k_B T_k}\right) \qquad \left[\exp\left(-\frac{f(\overline{x}) - f(x)}{k_B \cdot T_k}\right)\right],$$

  where T_k is the heat bath temperature and k_B is the Boltzmann constant.

As the number of performed iterations increases, the current temperature T_k is decreased, resulting in a smaller probability of accepting not improving solutions. For the CSP, Liu et al.'s SA sets the initial temperature T_0 to $\frac{m}{2}$. The current temperature T_k is reduced every 100 iterations according to the "geometric cooling schedule" that is, $T_{k+100} = \gamma \cdot T_k$, where $\gamma = 0.9$. The stopping criterion is to reach a current temperature less than or equal to 0.001. Despite the interesting ideas proposed in [18], the experimental analysis involves only small instances with up to 40 strings of length 40.

For the special case of $|\Omega| = 3$ and $|\Sigma| = 2$, [19] designed an exact approach called distance first algorithm (DFA), whose basic idea is to let the Hamming distance $d_H(s^*, s^i)$, $i = 1, 2, 3$, be as close as possible. The algorithm decreases the distance between the string that is farthest to the other two strings and solution s^* while increasing the distance between the string that is closest to the other two strings and solution s^*. For the general case, the authors proposed a polynomial-time heuristic resulting from a combination of local search strategies inspired by [26] and an approximation algorithm called Largest Distance Decreasing Algorithm (LDDA) which is based on similar ideas as DFA.

More recently, Tanaka [29] proposed a novel heuristic (TA) based on the Lagrangian relaxation of the ILP model of the problem that allows to decompose the problem into subproblems, each corresponding to a position of the strings. The proposed algorithm combines a Lagrangian multiplier adjustment procedure to obtain feasibility and a tabu search as local improvement procedure. In [4], Della Croce and Salassa described three relaxation-based procedures. One procedure (RA) rounds up the result of continuous relaxation, while the other two approaches (BCPA and ECPA) fix a subset of the integer variables in the continuous solution at the current value and let the solver run on the remaining (sub)problem. The authors also observed that all relaxation-based algorithms have been tested on *rectangular instances*, i.e., with $n \ll m$, and that the instances such that $n \geq m$ are harder to be solved due to the higher number of constraints imposed by the strings, which enlarges the portion of non-integer components in the continuous solution of the problem. In the attempt to overcoming this drawback, Croce and Garraffa [3] designed a multistart relaxation-based algorithm (called the selective fixing algorithm) that for a predetermined number of iterations takes a feasible solution as input and iteratively selects variables to be fixed at their initial value until the number of free variables is small enough that the remaining subproblem can be efficiently solved to optimality by an ILP solver. The new solution found by the solver can then be used as initial solution for the next iteration. The authors have experimentally shown that their algorithm is much more robust compared to the state-of-the-art competitors and is able to solve a wider set of instances of different types, including those with $n \geq m$.

The Close to Most String Problem (CTMSP)

The closest string problem can be seen as a special case of the so-called close to most string problem (CTMSP) that consists in determining a string close to most of the strings in the input set Ω. This can be formalized by saying that, given a threshold t, a string s^* must be found maximizing the variable l such that

$$d_H(s^*, s^i) \leq t, \text{ for } s^i \in P \subseteq \Sigma \text{ and } |P| = l.$$

$$\max \sum_{i=1}^{n} y_i \tag{7}$$

subject to:

$$\sum_{c \in \Sigma} x_{ck} = 1 \quad \text{for } k = 1, \ldots, m \tag{8}$$

$$\sum_{k=1}^{m} x_{s_k^i k} \geq m \cdot y_i - t \quad \text{for } i = 1, \ldots, n \tag{9}$$

$$x_{ck}, y_i \in \{0, 1\}$$

Constraints (8) ensure that for each position k of a possible solution, exactly one character from Σ_k is chosen. Constraints (9) ensure that y_i can only be set to 1 if and only if the number of differences between $s^i \in \Omega$ and the possible solution (as defined by the setting of the variables x_{ck}) is less than or equal to t. Remember, in this context, s_k^i denotes the character at position k in $s^i \in \Omega$.

Despite its similarity with the CSP, the CTMSP has not been widely studied. In [2] it was proved that this problem has no polynomial-time approximation scheme (**PTAS**) unless **NP** has randomized polynomial-time algorithms.

The Farthest String Problem (FSP)

Given a finite set of strings Ω over alphabet Σ, a problem complementary to the CSP is that of finding a string $s^* \in \Sigma^m$ farthest from the strings in Ω. This type of problem can be useful in situations such as finding a genetic string that cannot be associated to a given number of species.

Like the CSP, the farthest string problem can be formulated mathematically in form of an ILP, where both decision variables and constraints have an interpretation which is contrary to the one in the CSP:

$$\max d \tag{10}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{11}$$

$$m - \sum_{k=1}^{m} x_{s_k^i k} \geq d \quad \text{for } i = 1, 2 \ldots, n \tag{12}$$

$$d \in \mathbb{N}^+, \tag{13}$$

$$x_{ck} \in \{0, 1\}, \qquad \text{for } k = 1, 2, \ldots, m, \ \forall\, c \in \Sigma_k. \tag{14}$$

The computational intractability of the FSP was demonstrated in [16], where it was proved that the problem remains intractable even for the simplest case where the alphabet has only two characters.

Despite its inherent computational intractability, it has be shown in [16] that there is a PTAS for the FSP. The algorithm is based on the randomized rounding of the relaxed solution of the above-reported ILP and uses the randomized rounding technique together with probabilistic inequalities to determine the maximum error possible in the solution computed by the algorithm. Note that, the mathematical formulation of FSP is quite similar to the one used for the CSP, with only a change in the optimization objective, and the inequality sign in the constraint

$$m - \sum_{j=1}^{m} x_{s^i_j j} \geq d, \qquad i = 1, 2, \ldots, n.$$

Thus, to solve the problem using an ILP formulation, one can use similar techniques to those employed for solving the CSP. In 2011 [31] and more recently in 2015 Zörnig [32] proposed a few integer programming models for some variants of the farthest string problem and the closest string problem. The number of variables and constraints is substantially less compared with state-of-the-art integer linear programming models, and the solution of the linear programming relaxation contains only a small proportion of non-integer values, which considerably simplifies both a subsequent rounding process and a branch and bound procedure.

The Far From Most String Problem (FFMSP)

A problem closely related to the farthest string problem is the far from most string problem (FFMSP). It consists in determining a string far from most of the strings in the input set Ω. This can be formalized by saying that, given a threshold t, a string s^* must be found maximizing the variable l such that

$$d_H(s^*, s^i) \geq t, \text{ for } s^i \in P \subseteq \Sigma \text{ and } |P| = l.$$

In [1], the FFMSP has been mathematically formulated as an ILP. In fact, by defining a Boolean variable x_{ck} for each position k ($k = 1, 2 \ldots, m$) of a possible solution and for each character $c \in \Sigma_k$ and a Boolean variable y_i ($i = 1, 2 \ldots, n$) for each of the n input strings provided in set Ω, the FFMSP can be stated as the following ILP:

$$\max \sum_{i=1}^{n} y_i \tag{15}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{16}$$

$$\sum_{k=1}^{m} x_{s_k^i k} \leq m - t \cdot y_i \quad \text{for } i = 1, 2 \ldots, n \tag{17}$$

$$x_{ck}, y_i \in \{0, 1\}$$

Constraints (16) ensure that for each position k of a possible solution, exactly one character from Σ_k is chosen. Constraints (17) ensure that y_i can only be set to 1 if and only if the number of differences between $s^i \in \Omega$ and the possible solution (as defined by the setting of the variables x_{ck}) is greater than or equal to t. Remember, in this context, s_k^i denotes the character at position k in $s^i \in \Omega$.

Despite the similarity, it can be shown [16] that the FFMSP is much harder to approximate than the FSP, due to the approximation preserving reduction to FFMSP from the independent set problem, a classical and computationally intractable combinatorial optimization problem. In particular, [16] demonstrated that for strings over an alphabet Σ with $|\Sigma| \geq 3$, approximating the FFMSP within a polynomial factor is **NP**-hard.

The first attempt in the direction of the design of heuristic methods to efficiently solve the FFMSP was done in [22, 23], who proposed a heuristic algorithm consisting of a simple greedy construction followed by an iterative improvement phase. Later, [6] designed a simple GRASP, recently improved in [25]. Mousavi et al. noticed that the search landscape of the FFMSP is characterized by many solutions having the same objective value. Consequently, local search is likely to visit many suboptimal local maxima. To efficiently escape from these local maxima, Mousavi et al. devised a new hybrid heuristic evaluation function and used it in conjunction with the objective function when evaluating neighbor solutions during the local search phase in the GRASP framework.

Ferone et al. [5] designed the following pure and hybrid multistart iterative heuristics:

- a pure GRASP, inspired by [6];
- a GRASP that uses forward path-relinking for intensification;
- a pure VNS;
- a VNS that uses forward path-relinking for intensification;
- a GRASP that uses VNS to implement the local search phase; and
- a GRASP that uses VNS to implement the local search phase and forward path-relinking for intensification.

The algorithms were tested on several random instances, and the results showed that the hybrid GRASP with VNS and forward path-relinking always found much better quality solutions compared with the other algorithms, but clearly with higher running times as compared to the pure GRASP and the hybrid GRASP with forward path-relinking. The best objective function values found by GRASP and its hybrids were when the construction phase was more greedy than random. The integration of forward path-relinking as an intensification procedure in the pure metaheuristics was beneficial in terms of solution quality. A further investigation conducted, studying the empirical distributions of the random variable *time-to-target-solution value*, revealed that, given any fixed amount of computing time, GRASP with forward path-relinking has an empirically higher probability than all competitors of finding a target solution.

In [1], besides the first linear integer programming formulation for the FFMSP described above (15), (16), and (17), a hybrid ant colony optimization approach has been proposed. This hybrid approach consists of two phases. A first phase applies ant colony optimization until the convergence of the pheromone values is reached. After this first phase, the algorithm possibly applies a second phase in which the hybridization with a mathematical programming solver takes place. Both the linear integer programming formulation and the hybrid ant colony algorithm have compared to the most performing hybrid GRASP with path-relinking, and computational results on a large set of randomly generated test instances have indicated that the hybrid ACO is very competitive.

A Simple ILP-Based Heuristic

In the following we present the results of a quite simple ILP-based heuristic which can be applied to all four problems described before: the closest string problem (CSP), the clost to most string problem (CTMSP), the farthest string problem (FSP), and the far from most string problem (FFMSP). The heuristic is based on the ILP models of the four problems. It works as follows. Given a fixed computation time limit (t_{limit}), maximally half of this computation time is given to an ILP solver for tackling the mixed integer linear problem (MILP) that is obtained by relaxing the x_{ck}-variables involved in all four ILP models. The used ILP solver returns the best solution found in the given computation time. Note that this solution may, or may not, correspond to the optimal MILP solution. The fractional values of the x_{ck}-variables after termination of the solver are henceforth denoted by x'_{ck}. In the second phase of the heuristic, the corresponding ILP models are solved in the remaining computation time, with the following additional constraints:

$$x_{ck} = 1 \quad \text{for } k = 1, \ldots, m,\ c \in \Sigma_k,\ x'_{ck} = 1 \tag{18}$$

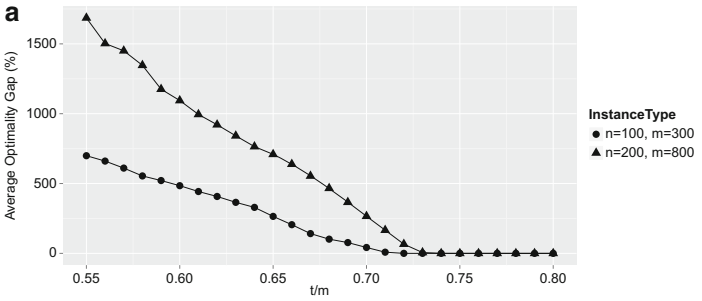
In other words, whenever a variable x_{ck} in the best-found MILP solution has a value of 1, this value is fixed for the solution of the ILP model.

Experimental Evaluation

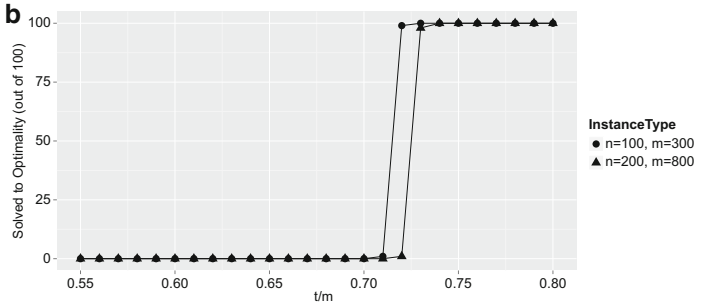
The heuristic described above is compared, in the case of all four problems, with the application of the ILP solver to the original ILP models. Moreover, a simple greedy algorithm was applied. This greedy algorithm generates a solution by selecting for each position of the solution string the letter which, in the case of the CSP and the CTMSP, has the least number of appearances at this position in the set of input strings and which, in the case of the FSP and the FFMSP, has the highest number of appearances at this position in the set of input strings. The greedy algorithm is henceforth denoted by GREEDY. As (M)ILP solver we used IBM ILOG CPLEX V12.1. The experimental results were obtained on a cluster of PCs with "Intel(R) Xeon(R) CPU 5160" CPUs of four nuclei of 3000 MHz and 4 GB of RAM. Moreover, CPLEX was configured for single-threaded execution. Depending on the problems, we used two different computation time limits: $t_{\text{limit}} = 200\,\text{s}$ and $t_{\text{limit}} = 3600\,\text{s}$ per problem instance. The different applications of CPLEX and the ILP-based heuristic, respectively, are named accordingly: CPLEX-200, CPLEX-3600, HEURISTIC-200, and HEURISTIC-3600.

All the algorithms described above were applied, in the context of all four problems, to a set of benchmark instances that was originally introduced in [5] for the FFMSP. This set consists of random instances of different size. More specifically, the number of input strings (n) is in $\{100, 200\}$, and the length of the input strings (m) is in $\{300, 600, 800\}$. In all cases, the alphabet size is four, that is, $|\Sigma| = 4$. For each combination of n and m, the set consists of 100 random instances. This makes a total of 600 instances. Note that, in the context of the CTMSP and the FFMSP, a value for parameter t must be specified before running the algorithm(s). However, a sensible choice of t is not trivial. For example, in the context of the CTMSP, the lower the value of t, the easier it should be, for example, for CPLEX to solve the problem to optimality. In order to be able to choose meaningful values for t, the following experiments were executed. CPLEX was applied to each problem instance—both concerning the CTMSP and the FFMSP—for each value of $t \in \{0.05, 0.02, \ldots, 0.95\,\text{m}\}$. This was done with a time limit of 3600 s per run. The corresponding optimality gaps (averaged over 100 problem instances) and the number of instances (out of 100) that was solved to optimality are graphically presented in Fig. 1. The results reveal that, in the case of the CTMSP, the problem becomes difficult for approx. $t \leq 0.72\,\text{m}$. In the case of the FFMSP, the problem becomes difficult for approx. $t \geq 0.78\,\text{m}$. Therefore, the following values for t were chosen for the final experimental evaluation: $t \in \{0.65, 0.7, 0.75\,\text{m}\}$ in the case of the CTMSP and $t \in \{0.75, 0.8, 0.85\,\text{m}\}$ in the case of the FFMSP.

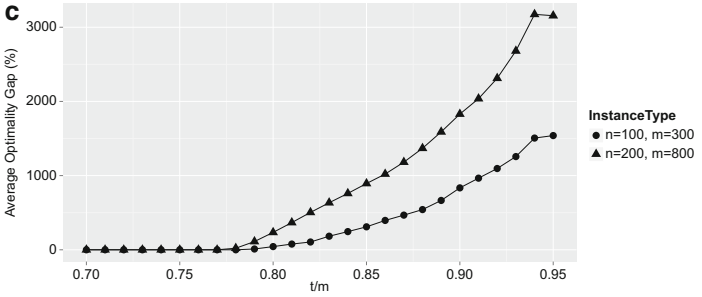
The numerical results for the CSP are shown in Table 1. They are presented as averages over the 100 instances for each combination of n (the number of input strings) and m (the length of the input strings). For all three algorithms, we provide the values of the best-found solutions (averaged over 100 problem instances) and the computation time at which these solutions were found. In the case of CPLEX-200, the average optimality gap is additionally provided. The results clearly show that HEURISTIC-200 outperforms both GREEDY and CPLEX-200.



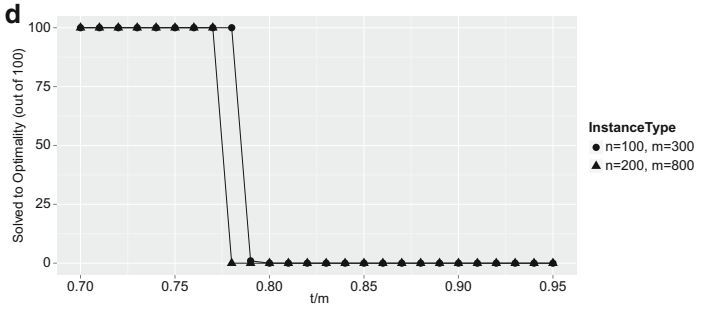
Average optimality gap (in percent) of CPLEX for the CTMSP.



Number of instances solved (out of 100) by CPLEX for the CTMSP.



Average optimality gap (in percent) of CPLEX for the FFMSP.



Number of instances solved (out of 100) by CPLEX for the FFMSP.

Fig. 1 Justification for the choice of parameter t in the context of the CTMSP and the FFMSP. (**a**) Average optimality gap (in percent) of CPLEX for the CTMSP. (**b**) Number of instances solved (out of 100) by CPLEX for the CTMSP. (**c**) Average optimality gap (in percent) of CPLEX for the FFMSP. (**d**) Number of instances solved (out of 100) by CPLEX for the FFMSP

Table 1 Numerical results for the CSP

		GREEDY		HEURISTIC-200		CPLEX-200		
n	m	Value	Time	Value	Time	Value	Time	Gap
100	300	228.55	<0.009	**213.33**	10.79	213.48	10.53	0.88
100	600	446.91	<0.009	**422.90**	7.36	423.06	5.22	0.47
100	800	590.08	<0.009	**562.40**	6.86	562.52	7.19	0.36
200	300	235.02	<0.009	**219.99**	12.74	220.03	33.42	1.38
200	600	457.19	<0.009	**434.33**	14.03	434.36	67.29	0.78
200	800	604.49	<0.009	**576.95**	11.59	577.01	87.03	0.60

Table 2 Numerical results for the FSP

		GREEDY		HEURISTIC-200		CPLEX-200		
n	m	Value	Time	Value	Time	Value	Time	Gap
100	300	221.67	<0.009	**236.53**	4.46	236.41	9.34	0.68
100	600	455.15	<0.009	**476.55**	7.13	476.39	7.10	0.38
100	800	611.61	<0.009	**636.58**	10.13	636.45	9.09	0.27
200	300	215.74	<0.009	230.11	9.29	**230.14**	25.72	1.24
200	600	443.88	<0.009	**465.68**	10.81	465.59	53.41	0.67
200	800	596.41	<0.009	**622.81**	10.52	622.72	60.49	0.51

Similar conclusions can be drawn in the case of the FSP, for which the results are presented in Table 2. Except for one case ($n = 200, m = 300$), HEURISTIC-200 outperforms both GREEDY and CPLEX-200.

In the case of the CTMSP, both the ILP-based heuristic and CPLEX were applied with computation time limits 200 and 3600 CPU seconds. Therefore, Table 3 contains results for HEURISTIC-200, HEURISTIC-3600, CPLEX-200, and CPLEX-3600. The following observations can be made:

- Both CPLEX and the ILP-based heuristic greatly outperform GREEDY.
- When the problem is rather easy—that is, for a setting of $t = 0.75$ m—CPLEX has usually slight advantages over the ILP-based heuristic. This is the case especially for the instances with larger number of input strings ($n = 200$).
- With growing problem difficulty, the ILP-based heuristic starts to outperform CPLEX. In particular, for a setting of $t = 0.65$ m, the differences in the qualities of the obtained solutions between the ILP-based heuristic and CPLEX are significant.

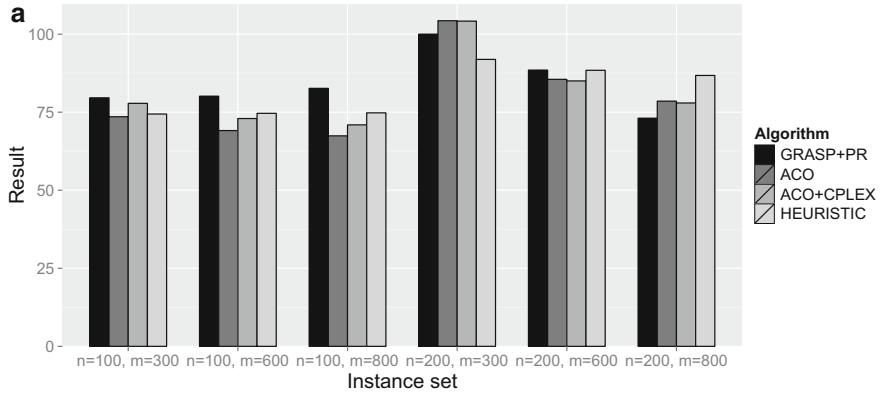
Not surprisingly, the same observations can be made in the context of the FFMSP, for which the results are provided in Table 4. As the considered benchmark instances were originally used for the FFMSP, we are able to compare to current state-of-the-art results for this problem (see [1]). This comparison is graphically presented in Fig. 2 for all instances concerning the interesting cases $t = 0.8$ m and $t = 0.85$ m. The results show that, for $t = 0.8$ m, HEURISTIC-3600 is generally outperformed by the other state-of-the-art methods. However, note that when the instance size

Table 3 Numerical results for the CTMS problem

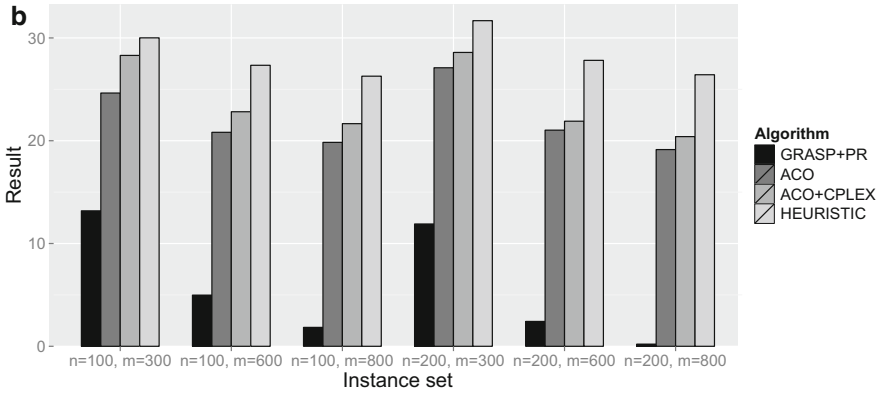
			GREEDY		HEURISTIC-200		HEURISTIC-3600		CPLEX-200			CPLEX-3600		
n	m	t	Value	Time	Value	Time	Value	Time	Value	Time	Gap	Value	Time	Gap
100	300	0.65 m	4.18	<0.009	31.87	140.45	**33.79**	2315.36	23.39	110.15	262.64 %	27.14	1509.07	210.56 %
100	300	0.7 m	55.94	<0.009	72.08	125.04	**74.75**	1642.93	69.95	62.70	41.98 %	72.14	1411.02	37.19 %
100	300	0.75 m	98.30	<0.009	99.98	2.55	**100.00**	4.45	**100.00**	0.81	0.00 %	**100.00**	1.06	0.00 %
100	600	0.65 m	0.70	<0.009	29.28	130.18	**31.45**	2182.52	21.96	100.77	289.23 %	23.16	1165.62	268.94 %
100	600	0.7 m	56.05	<0.009	72.50	77.33	**75.05**	876.32	71.30	109.45	40.03 %	73.44	1106.52	35.80 %
100	600	0.75 m	99.74	<0.009	**100.00**	1.17	**100.00**	1.20	**100.00**	1.72	0.00 %	**100.00**	1.97	0.00 %
100	800	0.65 m	0.18	<0.009	28.13	126.89	**30.35**	2131.77	20.14	70.47	328.33 %	25.86	1866.29	228.77 %
100	800	0.7 m	56.84	<0.009	73.14	51.06	**76.03**	837.32	72.88	119.22	37.34 %	75.23	1052.56	32.93 %
100	800	0.75 m	99.94	<0.009	99.99	2.24	**100.00**	2.91	**100.00**	2.31	0.00 %	**100.00**	2.16	0.00 %
200	300	0.65 m	2.16	<0.009	32.12	146.42	**36.93**	2493.67	23.81	94.62	576.99 %	24.30	735.91	562.71 %
200	300	0.7 m	65.51	<0.009	81.25	145.30	**92.82**	2720.97	85.61	144.05	121.06 %	92.37	1784.17	103.87 %
200	300	0.75 m	186.07	<0.009	192.90	43.00	195.05	595.86	**200.00**	3.22	0.00 %	**200. 00**	3.05	0.00 %
200	600	0.65 m	0.05	<0.009	28.46	129.11	**32.54**	2242.65	20.01	32.28	712.81 %	23.25	1192.03	594.35 %
200	600	0.7 m	50.04	<0.009	70.71	137.05	**88.04**	2570.76	66.95	118.40	185.15 %	81.96	1794.91	132.00 %
200	600	0.75 m	196.29	<0.009	196.23	44.52	198.05	647.22	**200.00**	4.86	0.00 %	**200. 00**	4.75	0.00 %
200	800	0.65 m	0.03	<0.009	27.21	128.22	**30.84**	2141.78	20.13	46.58	709.14 %	22.70	2238.87	615.48 %
200	800	0.7 m	43.27	<0.009	67.18	137.80	**86.59**	2552.02	63.94	122.52	198.62 %	79.93	2266.09	138.14 %
200	800	0.75 m	198.40	<0.009	196.50	44.65	198.27	623.50	**200.00**	7.00	0.00 %	**200. 00**	9.00	0.00 %

Table 4 Results for the FFMS problem

			GREEDY		HEURISTIC-200		HEURISTIC-3600		CPLEX-200			CPLEX-3600		
n	*m*	*t*	Value	Time	Value	Time	Value	Time	Value	Time	Gap	Value	Time	Gap
100	300	0.75 m	98.40	<0.009	**100.00**	0.09	**100.00**	0.16	**100.00**	0.10	0.00 %	**100.00**	0.11	0.00 %
100	300	0.8 m	54.67	<0.009	71.79	134.70	**74.41**	2594.31	69.87	69.91	42.69 %	71.56	1069.58	38.61 %
100	300	0.85 m	1.71	<0.009	28.83	118.05	**30.01**	1831.47	23.41	118.42	298.96 %	26.37	1881.75	249.42 %
100	600	0.75 m	99.82	<0.009	**100.00**	0.29	**100.00**	0.31	**100.00**	0.30	0.00 %	**100.00**	0.34	0.00 %
100	600	0.8 m	53.52	<0.009	71.88	113.65	**74.64**	1976.14	70.93	81.89	40.91 %	71.97	575.00	38.64 %
100	600	0.85 m	0.10	<0.009	25.93	107.92	**27.34**	1808.34	20.37	101.20	360.32 %	24.11	2014.55	285.93 %
100	800	0.75 m	99.99	<0.009	**100.00**	0.49	**100.00**	0.53	**100.00**	0.47	0.00 %	**100.00**	0.56	0.00 %
100	800	0.8 m	53.42	<0.009	71.88	111.76	**74.79**	1685.50	71.17	105.51	40.63 %	72.55	514.75	37.80 %
100	800	0.85 m	0.04	<0.009	25.08	105.12	**26.28**	1804.13	19.82	109.72	374.61 %	22.46	1573.26	317.03 %
200	300	0.75 m	188.58	<0.009	199.99	0.35	199.99	0.36	**200.00**	0.33	0.00 %	**200.00**	0.43	0.00 %
200	300	0.8 m	60.63	<0.009	78.58	124.40	**91.91**	2635.07	82.90	143.63	136.67 %	89.56	1775.12	118.20 %
200	300	0.85 m	0.61	<0.009	28.41	113.99	**31.68**	1869.94	20.13	71.95	813.23 %	24.88	2155.67	635.34 %
200	600	0.75 m	196.81	<0.009	**200.00**	0.56	**200.00**	0.60	**200.00**	0.57	0.00 %	**200.00**	0.68	0.00 %
200	600	0.8 m	43.29	<0.009	69.11	138.76	**88.42**	2459.95	62.67	116.46	213.91 %	77.34	1778.80	153.80 %
200	600	0.85 m	0.00	<0.009	24.93	103.93	**27.82**	1815.27	19.88	128.44	826.66 %	21.53	1173.71	753.46 %
200	800	0.75 m	198.75	<0.009	**200.00**	0.93	**200.00**	0.94	**200.00**	0.93	0.00 %	**200.00**	1.11	0.00 %
200	800	0.8 m	36.03	<0.009	69.45	139.97	**86.77**	2536.96	59.55	110.48	230.10 %	73.93	2565.90	166.09 %
200	800	0.85 m	0.00	<0.009	23.82	103.74	**26.42**	1803.11	18.67	47.26	887.78 %	20.79	808.43	784.97 %



Results for all instances with $t = 0.8m$.



Results for all instances with $t = 0.85m$.

Fig. 2 Graphical representation of the comparison between HEURISTIC-3600 and the current FFMSP state-of-the-art methods (GRASP+PR, ACO, and ACO+CPLEX) for $t = 0.8$ m (see (**a**)) and $t = 0.85$ m (see (**b**))

(in terms of the number of input strings and their length) grows, HEURISTIC-3600 starts to produce better results than the competitors. In the case of $t = 0.85$ m, which results in more difficult instances than $t = 0.8$ m, HEURISTIC-3600 clearly outperforms the current state-of-the-art methods.

Conclusions

The goal of this chapter was to provide an overview of some string selection and comparison problems, with special emphasis on the optimization and operational research perspective. Besides mathematical models that can be used to find exact solutions only up to a certain instance size, there are many approximate techniques,

proposed by researchers from several heterogenous communities, which are more or less efficient, depending on the input data and the type of information they make use of. Not surprisingly, generally there is no single best approach that wins in every aspect. Therefore, we proposed a simple ILP-based heuristic that can be used for any of the four considered problems. We have shown that this heuristic outperforms both a general greedy algorithm and the application of an ILP solver (CPLEX) to the original ILP models. In the case of the far from most string problem, we were even able to show that this simple heuristic is able to produce state-of-the-art results for instances which are intrinsically difficult to be solved.

Cross-References

- Genetic Algorithms
- GRASP
- Variable Neighborhood Descent
- Variable Neighborhood Search

Acknowledgments C. Blum was supported by project TIN2012-37930 of the Spanish Government. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science).

References

1. Blum C, Festa P (2014) A hybrid ant colony optimization algorithm for the far from most string problem. In: Proceedings of the 14th European conference on evolutionary computation in combinatorial optimisation (EvoCOP2014). Lecture notes in computer science, vol 8600. Springer, Berlin, pp 1–12
2. Boucher C, Landau G, Levy A, Pritchard D, Weimann O (2013) On approximating string selection problems with outliers. Theor Comput Sci 498:107–114
3. Croce FD, Garraffa M (2014) The selective fixing algorithm for the closest string problem. Comput Oper Res 41:24–30
4. Croce FD, Salassa F (2012) Improved lp-based algorithms for the closest string problem. Comput Oper Res 39:746–749
5. Ferone D, Festa P, Resende M (2013) Hybrid metaheuristics for the far from most string problem. In: Proceedings of 8th international workshop on hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Berlin, pp 174–188
6. Festa P (2007) On some optimization problems in molecular biology. Math Biosci 207(2): 219–234
7. Frances M, Litman A (1997) On covering problems of codes. Theory Comput Syst 30(2): 113–119
8. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Reading, Massachusetts, Addison-Wesley
9. Gramm J (2003) Fixed-parameter algorithms for the consensus analysis of genomic data. Eberhard Karls University of Tübingen, PhD thesis
10. Gramm J, Hüffner F, Niedermeier R (2002) Closest strings, primer design, and motif search. In: Sixth annual international conference on computational molecular biology, Washington, DC, pp 74–75

11. Gramm J, Niedermeier R, Rossmanith P (2003) Fixed-parameter algorithms for closest string and related problems. Algorithmica 37:25–42
12. Holland JH (1975) Adaptation in natural and artificial systems. Cambridge, Massachusetts, MIT Press
13. Kirkpatrick S (1984) Optimization by simulated annealing: quantitative studies. J Stat Phys 34(5–6):975–986
14. Lanctot J (2004) Some string problems in computational biology. University of Waterloo, PhD thesis
15. Lanctot J, Li M, Ma B, Wang S, Zhang L (1999) Distinguishing string selection problems. In: Proceedings of the annual ACM-SIAM symposium on discrete aLgorithms (SODA), Baltimore, pp 633–642
16. Lanctot J, Li M, Ma B, Wang S, Zhang L (2003) Distinguishing string selection problems. Inf Comput 185(1):41–55
17. Li M, Ma B, Wang L (1999) Finding similar regions in many strings. In: ACM symposium on theory of computing (STOC'99), New York, pp 473–482
18. Liu X, He H, Sýkora O (2005) Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem. In: Proceedings of ADMA 2005. Lecture notes in artificial intelligence, vol 3584. Springer, Berlin Heidelberg New York, pp 591–597
19. Liu X, Liu S, Hao Z, Mauch H (2011) Exact algorithm and heuristic for the closest string problem. Comput Oper Res 38(11):1513–1520
20. Ma B, Sun X (2008) More efficient algorithms for closest string and substring problems. Lecture notes in computer science, vol 4955. Springer, Berlin, pp 396–409
21. Macario A, de Macario EC (eds) (1990) Gene probes for bacteria. Academic Press, San Diego
22. Meneses C, Oliveira C, Pardalos P (2005) Optimization techniques for string selection and comparison problems in genomics. IEEE Eng Med Biol Mag 24(3):81–87
23. Meneses C, Pardalos P, Resende M, Vazacopoulos A (2005) Modeling and solving string selection problems. In: Proceedings of the 2005 international symposium on mathematical and computational biology (BIOMAT 2005), pp 55–65
24. Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
25. Mousavi S, Babaie M, Montazerian M (2012) An improved heuristic for the far from most strings problem. J Heuristics 18:239–262
26. Pardalos P, Oliveira C, Lu Z, Meneses C (2004) Optimal solutions for the closest string problem via integer programming. INFORMS J Comput 16:419–429
27. Roman S (1992) Coding and information theory. Graduate texts in mathematics, vol 134. Springer, New York
28. Sim J, Park K (1999) The consensus string problem for a metric is NP-complete. In: Proceedings of the annual Australasian workshop on combinatorial algorithms (AWOCA), pp 107–113
29. Tanaka S (2012) A heuristic algorithm based on Lagrangian relaxation for the closest string problem. Comput Oper Res 39:709–717
30. Wang L, Zhu B (2009) Efficient algorithms for the closest string and distinguishing string selection problems. Lecture notes in computer science, vol 5598. Springer, Berlin, pp 261–270
31. Zörnig P (2011) Improved optimization modelling for the closest string and related problems. Appl Math Modell 35(12):5609–5617
32. Zörnig P (2015) Reduced-size integer linear programming models for string selection problems: application to the farthest string problem. J Comput Biol 22(8):729–742

Supply Chain Management

43

Helena Ramalhinho Lourenço and Martín Gómez Ravetti

Contents

Introduction 1242
What Is Supply Chain Management? 1243
Decision-Making Problems in SCM 1245
Metaheuristic Algorithms for SCM Problems 1250
Current and Future Research Trends in SCM 1252
Conclusions 1254
Cross-References 1254
References 1255

Abstract

Supply chain management (SCM) is related to the management of all activities along a network of organizations to provide a good or a service to final customers. The efficiency of these activities can have a great impact on customer's satisfaction and cost reduction. However, SCM is not just the sum of activities along the supply chain but, instead, it must consider the organization, supervision, and control of all activities in the chain from an integrated and collaborative perspective aiming to provide a competitive advantage. From this point of view, an increase in the dimension and complexity of the decision problems involved is expected, as several actors with different goals must be considered to administrate efficiently the activities within the supply chain.

H. Ramalhinho Lourenço (✉)
Universitat Pompeu Fabra, Barcelona, Spain
e-mail: helena.ramalhinho@upf.edu

M. G. Ravetti
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
e-mail: martin.ravetti@dep.ufmg.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_54

This chapter briefly reviews the main concepts of SCM, identifying relevant decision and optimization problems and discussing possible solution approaches. Heuristics and metaheuristics are two of the best optimization tools to be used in solving and providing business insights for the SCM problems. This chapter also describes some successful metaheuristic approaches to SCM and it examines future research trends. A large number of applications of metaheuristics to SCM integrating new subjects, such as open and big data, smart cities, and online decision-making, just to mention a few, are foreseen.

Keywords
Supply chain management · Optimization problems · Metaheuristics

Introduction

Supply chain management (SCM) consists in integrating processes by linking major business functions from a particular company as well as across companies with business relationships. SCM is more than just summed up of a series of interconnected activities among business; it consist of integrated and collaborative processes within a clear business model that leads to a more cohesive and efficient performance of companies and better customer services. Being able to optimize the activities and processes along a supply chain is strategical for most businesses.

The activities and processes in the supply chain management include topics like location and network design, sourcing and inventory strategies, manufacturing and resource strategies, sustainability and green strategies, and also transportation and distribution strategies. At the operational level, we can find different topics as for example customer service management, demand management, returns management etc. To efficiently manage these areas, in many situations large-scale complex combinatorial optimization problems must be solved in a very effective way to be able to answer the increasingly demanding market.

Companies like INDITEX, AMAZON, DELL, etc. (just to mention some) have used and applied the concept of supply chain management to improve their processes and customer service with great success. These companies manage all activities along the supply chain in a collaborative way with the final objective to serve efficiently the final customer.

This chapter argues that metaheuristics can be an outstanding tool to help in the SCM decision-making. On the one hand, problems associated with SCM are becoming more complex and bigger causing greater impact on business performance. On the other hand, metaheuristics have attributes (as accuracy, speed, simplicity, and flexibility [1]) that can help dealing with problems that have arisen in SCM.

The objective of the chapter is to emphasize previous and potential metaheuristic's applications to SCM problems. We also revised some of the most relevant work in the literature that combines and apply metaheuristics to problems in the area of supply chain management.

The chapter is organized as follows: The first two sections describe main concepts and decision-making problems in SCM. Next, relevant applications of metaheuristics to different decision-making problems in SCM are presented and discussed. Relevant current and future topics in SCM where heuristics and metaheuristics can have a significant impact on the management of a supply chain are also examined. Finally, conclusions are drawn.

What Is Supply Chain Management?

Supply chain management has been a very popular topic in the last decade not only in business but also in academia. However, it is still possible to find some ambiguity in the concept and definition of SCM [2]. In this section, the most relevant definitions are presented and discussed to clarify SCM concepts. This definition will help us to identify the optimization problems within SCM and consequently the applications of Metaheuristics to this area.

A supply chain is a set of organizations, activities, people, resources, and products that interact to provide a service or a good to a customer, from the raw material initial stages to the end users and backward when reverse logistics is relevant. Therefore, it is a network of elements interacting with each other with the objective of satisfying customers while minimizing costs. The management of all activities along the supply chain is known as supply chain management. These activities include logistics, manufacturing and operations, sourcing, distribution, sales and marketing, accounting and finance, product design, and information technology activities among others that can be specific for each type of industry. The supply chain is related to the flow of products and goods along the chain, but information flow also plays a relevant role in its efficient management. Another related matter in SCM is the collaboration and integration of the activities in the chain since decisions in one element affect directly the whole supply chain. Thus, decision based on suboptimization approaches may lead to enormous inefficiencies in the chain. Consequently, the decision-making problems in SCM are usually more complex and have greater dimensions than other decision problems in traditional individual businesses.

In the literature, several definitions can be found. However, in this chapter, only two of the most commonly mentioned are analyzed.

The Council of Supply Chain Management Professionals (CSCMP) defines SCM as: "Supply Chain Management encompasses the planning and management of all activities involved in sourcing and procurement, conversion, and all logistics management activities. Importantly, it also includes coordination and collaboration with channel partners, which can be suppliers, intermediaries, third-party service providers, and customers. In essence, supply chain management integrates supply and demand management within and across companies" [2].

Simchi-Levi et al. [3] describes SCM as: "... a set of approaches utilized to efficiently integrate suppliers, manufacturers, warehouses, and stores, so that merchandise is produced and distributed in the right quantities, to the right locations,

and at the right time, to minimize system-wide costs while satisfying service level requirements."

Janvier-James [4] presents a discussion on the lack of consensus in the definitions of SCM, a list of definitions found in the literature, and review on the theory and practice of SCM. In chapter 2 of [5], it can be found a history of SCM and also a discussion on its definition.

When focusing on the common aspects of these definitions, it is possible to see that a supply chain can be understood as a set of interconnected entities and activities which are concerned with the plan, coordination, and control of materials, parts, and finished goods from suppliers to customers. An essential aspect of a successful management of these activities is the integration, cooperation, and coordination of the decision-making processes as well as in the sharing of information throughout the entire supply chain. Integrated decisions in SCM are related to cross functional planning. SCM coordination attempts to maintain an efficient and smooth flow of products and services. An effective SCM is not just a sum of management individual activities, but it must take into account the effect of their interrelation, seeking collaborative actions among the chain. But the importance of concepts as performance and quality service cannot be ignored. The ideas around integration and collaboration arise to improve the customer service and the performance by taking advantages of the synergies among the elements of a supply chain and to have a common objective and align the targets to satisfy the final customer.

In the literature, there is still some uncertainty about the definitions of SCM and Logistics Management (LM). Although, these two areas have a serious overlap; LM is a functional area of business while SCM crosses the businesses borders including relationships and interactions from different entities, as providers and customers for example. The CSCMP defines Logistics Management as follows: "Logistics management is that part of SCM that plans, implements, and controls the efficient, effective forward and reverses flow and storage of goods, services and related information between the point of origin and the point of consumption to meet customers' requirements" [2].

Other similar definition can be found in Johnson et al. [6]: "Logistics define the entire process of materials and products moving into, through, and out of a firm. Inbound logistics covers the movement of materials received by the suppliers. Material management describes the movements of materials and components within a firm. Physical distribution refers to the movement of goods outwards from the end of the assembly line to customers. Finally, SCM is a larger concept than logistics, dealing with managing both the flow of materials and the relationships among channel intermediaries from the point of origin of raw materials through to the final consumer."

Supply chain management gives a particular emphasis on the integration of the entire supply chain, including the logistics, marketing, operations and other activities within the organization, and with suppliers, suppliers of suppliers, customers, and customers of customers, etc. Therefore, in this chapter, we focus on cross functional aspects of the business collaboration and integrated activities as well as the decisions in SCM.

Decision-Making Problems in SCM

The decision-making in SCM is very complex as it may involve a large number of elements and processes, different businesses, or economic functions within each business. This section presents some general and relevant decision-making problems in SCM and relates them to optimization problems. Depending on the segment of the economy or industry sector, the SCM may have specific characteristics, sharing common difficulties. However, this chapter focuses on problems dealing with integration and collaboration issues along the supply chain.

The decision-making problems may be categorized into two groups: planning (long-term decision-making) and execution (tactical and short-term decision-making).

In planning, the most relevant decisions relate to the following areas: network design, sourcing and inventory strategies, manufacturing and resource strategies, sustainability and green strategies, transportation and distribution strategies, information strategies, and global supply chain strategies [7–10].

In execution, it is possible to identify several processes related to supply, manufacturing, logistics and distribution, and post-sale [11]. The problems that arise from these processes share cross functional issues along the chain.

How should the supply chain efficiency be evaluated? The main criteria mentioned in the literature are speed, reliability, cost, and customer satisfaction [4]. Other authors suggest resources measures (cost), output measures (customer related), or flexibility measures [10]. Therefore, the decision problems that can occur in SCM are in general related to finding solutions that perform well for the mentioned criterions.

Next, some examples of decision-making problems in a supply chain are described, emphasizing the ones where metaheuristics have been or can be applied. Please note that there is no intention to make an exhaustive list of problems but exemplify some important problems in SCM where heuristics and metaheuristics can be successfully applied. Some of these problems are already well known in the operations research literature; however very few of them have considered the integration and coordination aspect of SCM.

In SCM planning, two of the most relevant subjects are network design and facility location problems. Location problems have been extensively studied in operations research, and there is a large amount of applications of heuristics and metaheuristics to this area. However, the problems in this chapter on *location and network design* are problems from an integrated and collaborative perspective within a supply chain. For a survey on managerial issues in supply chain and network design, see [11]. The authors describe the main issues in network design, as determining its size, number and locations of the facilities, integration with tactical decisions as distribution, transportation, and inventory policies, as well as fulfilling customers' demand. They also evaluate supply chain network design where competition among supply chains is taking into account in the design phase. For the optimization point of view, the authors mention interesting objectives that

should be considered in the network design problem as social, environmental, economical, agility, and uncertainty considerations. Melo et al. in [12] presented a literature review of facility location models within a SCM context. One interesting aspect of this work is the identification of the key features that these models should capture in the strategic planning of the supply chain. They consider not only the common objectives and constraints of standard location problems but also its integration with other decision aspects of the chains as capacity, production, inventory, transportation, procurement, and routing. All of them received little attention in the literature. Other problems still very relevant in network design problems are globalization and internationalization of the supply chain, financial and taxation factors, risk factors, and multiobjective models (due to the consideration of multiple actors).

The *sourcing and inventory strategy* in SCM refers to several elements, for example, inventory positioning in the network, inventory objectives balancing cost and product availability, pull versus push systems, and inventory policies as VMI (vendor management inventory) or restock policies. Inventory and sourcing represent a large percentage of the supply chain's cost, thus the optimization of these elements is of critical importance. Notice that some decisions related to sourcing and inventory must be taken considering the whole supply chain including the production, transportation, and distribution decisions. These problems are well studied in the literature [13–17]. However, less attention has been done to strategic problems considering a collaborative and integrated point of view. There are few studied problems in inventory and sourcing from SCM perspective, the inventory-routing problem [18,19] and the vendor-managed inventory [20] are a few examples.

The *manufacturing and resource strategies* are associated with the process of producing a good or a service. Most problems and relevant decisions are related to the production planning and scheduling, global production assignment, postponement strategies, resource utilization, and resource investments. Also, related issues as market forecasts and product design make a great impact on the manufacturing and resource strategies. Once again, many problems can be defined within this subject; however, only recently the supply chain perspective has been considered when defining business strategies for manufacturing and resource use. This broad strategy leads to larger and complex problems if compared with the traditional single business perspective. Production planning within a supply chain can be a very challenging problem, as its solution may affect several elements of the chain, horizontally and vertically. For example, job scheduling has been mostly applied to deal with operational decision problems, but its integration with the production planning decision [21] and among different production plants is not commonly considered. Their integration with the availability of raw materials, the global forecasting, and distribution decisions can also be of great interest in the SCM. An overview of production and scheduling problems in the supply chain can be found on [22,23].

Sustainability SCM must be taken into consideration from product design and production to transportation and waste management. Good decisions in *sustainabil-*

ity and green strategies have a significant impact on the economics and market value of the business. For a literature review on the implications and interrelationships of SCM, sustainability, and lean management, we refer to [24]. The environmental and waste costs are often quite significative. Thus it should be taken into account in almost all SCM decisions mentioned in this chapter, for example, the impact of carbon dioxide emissions in transportation strategies. The design of green and lower environmental impact routes is attracting many researchers [25]. The sustainability or green strategies and returns management should also be considered in network design strategies, the latter is known as closed-loop network design, for example, Devika et al. in [26] apply metaheuristics to solve a similar type of problem successfully.

Transportation and distribution strategies are key elements in SCM. The connections among the elements are based on transportation activities or information interchange. Transportation elements not only permit to connect components in the chain but also it represents in many cases a substantial overall cost to the organizations in the chain. Therefore, managers need to have a good understanding of the transportation system and develop good transportation strategies. To the development and design of these strategies, several elements must be taken into account, i.e., transportation costs and performance, transit time and variability, loss and damage costs, transportation mode, international issues, transportation outsourcing, and collaborative transportation, just to mention a few. From the metaheuristic literature, a potential application is the study of the impact of transportation decisions in the SCM.

Today's information technologies allow companies to collect vast amounts of data related to several activities in the supply chain: from demand data to stock levels data, from the location of the transportation vehicle data to order delivery status, etc. Information strategies should be carefully considered since the impact on the business' performance is significant [27]. In reality, it is difficult to discuss SCM without mentioning information technologies. The capability of integration, collaboration, and coordination along a supply chain will not be possible without the information flow along the network. It should also be easy to understand that a good use of this information may lead to improvements in the decision-making and consequently on the overall performance of the business and customer satisfaction [28]. SCM strategic decisions on information are related to where and which data should be collected, accessed, storage, analyzed, and shared with other supply chain elements [28]. Notice that these decisions can affect all optimization problems, since all of them have the need for high-quality input data [29]. Other significant problems raised in this activity are related to how to deal with the enormous amount of available data. Feature selection and classification problems, as well as other data mining difficulties, must be formulated and solved to improve the information quality. Thus, metaheuristic techniques are usually an appropriate method to solve such large problems in a reasonable amount of time, avoiding suboptimizations, poor data aggregation, and other simplifications procedures. The availability of huge quantities of data will probably lead to a widespread use of metaheuristics and better decisions on SCM.

Several businesses have a worldwide presence, so global supply chain strategy is a relevant topic in SCM. Companies frequently produce globally but with some local personalization to improve sales numbers. So, strategic decisions related to marketing, financial, logistics, distribution, production, and postponement within a global supply chain can represent high complexity but at the same time make a great impact on the performance of the elements of the supply chain. Some of these SCM strategic decisions are, for example, where to produce or store the goods or how to distribute globally produced products and always considering: global market characteristics, technological aspects, global costs, and economic and political scenes [30]. The consideration of all these issues usually leads to very complex problems and eventually with nonlinear objective functions, where again, the use of metaheuristics can be extremely useful.

The characteristics of metaheuristics [1] make them one of the best tools to evaluate different scenarios during strategic decision-making processes. Managers face complex, integrated, and stochastic problems, and being able to obtain relevant insights by using optimization methods certainly provides them with a competitive advantage.

In the SCM execution area, the well-established supply chain processes as defined in Croxton et al. [31] are considered. These processes are:

- Customer relationship management
- Customer service management
- Demand management
- Order fulfillment
- Manufacturing flow management
- Supplier relationship management
- Product development and commercialization
- Returns management

Many other processes are suggested in the literature. However, they are quite similar since all include functional processes in supply, manufacturing, logistics and distribution, and post-sales.

Next, some examples of decision problems that occur in the abovementioned processes are briefly described. It is important to remember that the list of problems is not exhaustive, as the goal is to mention relevant examples with a focus on the use of metaheuristics.

Customer relationship management (CRM) is a well-established area in SCM and closely related to marketing. The objective of CRM is to identify target markets and implement marketing programs with key customers. Most of the literature has a focus on the marketing area. However, recently, there is an increase number of applications of operations research and metaheuristics to marketing and CRM processes. The main reason of this interest is due the increased complexity and dimensions of the decision problems, as for example: identification and segmentation of target customers and optimization of product portfolio, recommendation systems, and characteristics of the potential clients.

Customer service management (CSM) is the coordination structure for managing the business' relations with current and future clients. It includes the management of the sales department and information providers and it must deal with the communication among all departments and companies within the supply chain that provide products or services to customers. This process has a strong component of information systems, and different processes, from inventory problems, assignment of the corresponding customer support or representative, to optimize the sales team's organization. Most examples of applications of metaheuristics in CSM are found in inventory management, as [32, 33], and also in staff scheduling [34].

The main objective of the demand management process is to meet the customer's demand and balance their requirements with the capacities of the organizations in the chain. This is a crucial process, and one of its essential elements is the synchronization between the procurement, production, and distribution function to satisfy customer's demand and orders. Forecasting plays a major role in this process, as well as its integration into the resource planning and allocation [35].

The *order fulfillment* process provides for timely and accurate delivery of customer orders. One of the most relevant decisions in this process is the well-known *capacitated vehicle routing problem* (CVRP) [36]. The CVRP has been extensively studied in operations research area and is one of the problems with the largest number of applications of metaheuristic algorithms [37–40]. But from the SCM perspective, it is important to go beyond the classical CVRP and extend this problem to take into consideration more aspects within the supply chain. For example, the collaborative CVRP, that is, where several distribution companies collaborate to deliver their products [41, 42]. Another example is the CVRP with environmental and reverse logistics considerations [43, 44]. An important area of research is also the consideration of inventory constraints and costs in the routing decisions [18] or rich routing problems as in [45, 46], where metaheuristics have already been successfully applied.

The *manufacturing flow management* process is related to the production of the goods and services to serve the customers, including the moving of products through the plants and businesses. Within an SCM context, the production planning must take into account providers and customers, as well as the production capacity of the organization, differing significantly from the usual individual standpoint. Aspects as the production flexibility, batch size, cycle time, postponement, make to order, make to stock, customer order forecast, production scheduling, etc. must be considered when setting the production flow. Examples of applications of metaheuristics in this process can be found on [47, 48].

The *supplier relationship management* is associated with the relationships upstream in the supply chain and can be seen as a minor image of the CRM. Some of the important decision problems in this process are the selection of the right suppliers, cross docking systems, inventory coordination, and planning between providers and customers. The *product development and commercialization* process integrates all necessary activities to develop new products or services as their introduction to the market. The decision problems in this process are related to activities in the CRM and fulfillment; therefore issues such as market

and promotion planning, sales force training scheduling, make or buy decisions, inventory decisions, transportation planning, or commercialization planning are relevant. One example is the application of genetic algorithms to design different product structures considering different product parts, production processes, and green strategies [49].

The process of *returns management* is related to the reverse logistics and closed-loop supply chain processes. Over the last decade, business and academia have been increasing their attention to this issue due to three main reasons: impact on the environment and laws forcing the right disposal of products, recognition of the sustainable issues by the society, and social responsibility by the businesses. The forward logistics activities are already difficult to plan and optimize. The returns activities have an increased difficulty due to the stochastic aspect of the "demand." For example, if a company uses reused/recycled components in their production, the collection of these components can vary significantly along the time horizon, leading to an increased complexity of the production planning. Some applications of metaheuristics to design closed-loop supply chains have been addressed by [50]

The metaheuristics have already being applied to SCM at the operational level as it will be discussed in more detail in the next section. However, the need to consider integration and coordination aspects still exists, and future applications of metaheuristics to SCM must examine these issues as well as its computational efficiency and flexibility.

Metaheuristic Algorithms for SCM Problems

In the last years, there have been several applications of heuristics and metaheuristics to SCM decision problems. The reasons are quite clear, the increase in complexity due to the horizontal and vertical integration and the need for quick and flexible algorithms granted to metaheuristics, a special place in the methodologies applied to SCM.

In this section, we review relevant publications of heuristic algorithms applied to SCM problems. Note that the literature review is not exhaustive and readers are also referred to a recent survey on this topic [51].

In [52], the problem of distribution planning of a network of manufacturing plants, depots, and customers is analyzed. A heuristic based on an evolutionary algorithm approach (▸ Chap. 15, "Genetic Algorithms") is proposed where the solution consists in deciding which depots should be considered and how the product should be distributed from depots to clients minimizing fixed and delivery costs. The approach considers a bi-level structure; the upper level decides how the customers are supplied from the depots. The lower level decides on the manufacturing process.

The problem of coordinating the production schedule and the transportation of orders is analyzed in [53]. The problem aims to establish the best allocation of orders specifying, at the same time, their production sequence and completion times with the objective of minimizing the total cost of the supply chain. Two

genetic algorithms (▸ Chap. 15, "Genetic Algorithms") are proposed with different integration strategies, where the best approach allocate more CPU time in solving the scheduling problem.

Considering a seven-layer recovery network, including primary customers, collection/redistribution centers, recovery, recycling and disposal centers, and secondary customers, the problem of recovering products is analyzed in [54]. A mixed integer linear programming model is proposed to determine the collection and recycling centers for the logistic of recovered products, minimizing the total cost. To deal with real-sized instances, the authors propose heuristic approach based on Tabu search (▸ Chap. 25, "Tabu Search"), to design and select the best recovery option while minimizing fixed and variable costs.

Considering the same type of problem, but on a multiobjective approach, Devika et al. [26] analyzes a general closed-loop supply chain network with six echelons. They consider the minimization of total costs, the minimization of environmental impacts, and the maximization of social benefits. Considering two metaheuristics, adapted imperialist competitive algorithm (AICA) and variable neighborhood search (VNS) algorithm (▸ Chap. 26, "Variable Neighborhood Search"), three hybridization approaches are designed and tested against benchmark algorithms and on a case study. Other examples of quantitative approaches to green SCM can be found on [55–58].

There are few studies considering the transportation and distribution decisions at the strategic level. Lourenço and Ribeiro [59] explore the impact of considering logistics and marketing issues in the design of distribution strategies. Schmid et al. in [45] describe basic models for extensions of vehicle routing problems in the context of SCM. These extensions consider other elements in the supply chain as scheduling, packaging, batching, intermodality, etc. They also describe an important extension where inventory is considered. The inventory and routing interconnectivity is a well-known area in the operations research literature; see for example [18, 60, 61].

Considering a two-echelon supply chain, Cardona-Vales et al. [62] work on a bi-objective optimization problem, minimizing the cost for opening warehouses and the expected value of transportation costs and the maximum traveling time through the whole supply chain. Locations for warehouses and the assignment between warehouses and distribution centers have to be decided by the actual value of the demand. To deal with this stochastic problem, a hybrid heuristic based on Tabu search and GRASP (▸ Chap. 16, "GRASP") is designed to consider a multiobjective approach.

The design and evaluation of cooperative purchasing strategies for healthcare supply chains are evaluated in [63]. The authors propose hybrid heuristic algorithms based on *variable neighborhood search* and *Tabu search*. Considering a set of hospitals, they want to determine the best cooperation arrangements, to foresee global and individual savings and organize their joint supply chain to take advantage from cooperation.

Regarding *customer relationship management,* several examples of applications of metaheuristics can be found on market basket analysis [64, 65], product portfolio

optimization [66], store operation within CRM systems [67], or data mining applications in CRM [68]. The CRM process is receiving more attention from the SCM point of view; however, we could observe few publications using metaheuristics. We identify that there is great potential to apply these techniques to complex problems in CRM.

Although it is possible to see an increased number of metaheuristic applications to SCM, there are still several problems to be mathematically formulated. As a consequence, there is a lack of proper benchmarks and test beds leading to a deficiency in algorithm comparison and analysis.

Current and Future Research Trends in SCM

Supply chain management will continue to be a relevant topic in the following years, and the decision-making problems will be growing in quantity, dimension, and complexity. In this section, current and future research trends in metaheuristics applied to SCM are discussed. Many of them have not been yet extensively considered in the metaheuristic research literature.

Main topics:

- Greater integration and collaboration, horizontal and vertical, along with the supply chain
- Uncertainty decision-making in SCM
- Multiobjective approaches
- E-commerce and SCM
- Online decision-making in SCM
- Open and big data in SCM
- Smart cities and SCM

One of the main characteristics of SCM is the relevance of integration and collaboration between the elements of the chain. If this integration is not taken into account, there is not a truly SCM setting. However, most current works do not always consider these aspects. We conjecture that the main problem is that incorporating objectives and constraints to model the integration and collaboration of processes lead to very complex and in many cases nonlinear problems. Still, models and algorithms in the SCM decision-making must consider these aspects to obtain a real impact and add value to the business in general.

As previously mentioned, an increase in the dimension and complexity of the problem is expected. Thus, metaheuristics are certainly a clear option to deal with this new problem profile, especially when accurate solutions are needed in fast time. High-performance computing and hybrid methods are clear options to deal with this increased size and complexity. From the application point of view, cloud computing algorithms seem to be a good option for medium-sized companies willing to have access to low-cost distributed algorithms.

Efficient decisions in SCM must take into account an important element: *uncertainty*. Uncertainty is present all over the supply chain: uncertainty demand, travel times, cost, resource availability, etc. The uncertainty parameters make it difficult to optimize the system-wide costs and performance in SCM. The consideration of scenario strategies, robust methods to deal with the parameter's uncertainty, will play a major role in helping the decision-making. For example, a new line of research called simheuristics [69, 70] extends metaheuristic capabilities through simulation for solving problems with uncertainty parameters.

The need for integration of different elements of the supply chain often leads to optimization problems where many different objective functions must be assessed. Therefore, metaheuristics once again present themselves as an excellent option to efficiently deal with multiobjective problems in a complex environment.

E-commerce is already a reality, and with this new form of commerce had appeared new decision problems [71]. The e-commerce will be shortly a central part in many businesses, and this involves not only the retailing companies but also many other elements integrated into a supply chain, as production and logistics companies. The need to a fast response in e-commerce is also a reality, so the decision-making in a supply chain presents a different set of problems than the traditional ones.

The availability of technology and online data, as well as the need for flexibility and immediate responses, pushes the decision-making processes from an offline to an online perspective. In this new perspective, we do not count with the whole instance data but rather a sequence of input portions that must be solved as they arrive. Thus, online algorithms must take into account the current state of the system and react as a new portion of information arrives. Once again, the trade-off of computer efficiency and solution quality has to be carefully analyzed, but due to the complexity of the problems and the size of real scenarios, metaheuristic approaches are again promising in this new context.

Open email accounts, Internet search browsers, and mobile phones are just some of the new ways to capture online consumer's habit information. This incredible big amount of data represents an excellent opportunity to find commercial advantages. The problem is now how to extract good-quality information from these vast databases that usually cannot fit in regular servers and lacks proper structure [72]. The fact that these databases are relatively open in today's business increases the potential of the impact of this information in the SCM. Due to the characteristics of metaheuristics, these methods have an enormous potential to be applied in SCM decisions involving big and open data.

Although there are several different definitions of a smart city, the main concept behind the idea is to consider a city as an integrated system. The available information from its infrastructure and dynamical systems are continuously analyzed for the improvement of all the systems as energy, transport, healthcare, buildings, etc. [73]. The large amount of sensors and available data have a great impact on urban supply chain management. The availability of the resources of smart cities can provide substantial improvements in mobility, sustainability, distribution, and

many other SCM activities. Notice that in a smart city environment, the decision problems in SCM will become more complex, with a need for a faster and online response. For example, retailing companies can identify the closest transportation vehicles and make collaborative distribution decisions online.

These issues are only some of the actual and future relevant topics that must be addressed in SCM. Metaheuristics, as one of the available optimization tools, can provide better quality insights to these important issues in a faster time, improving significantly the decision making.

Conclusions

An efficient supply chain management is of critical importance in today's businesses; if we take into account the market's globalization and the increasing demand for flexibility and quality, a good SCM will be vital for the survival of any company.

This chapter reviews the main concepts and decision problems of SCM. The challenges for obtaining relevant business insights and decisions are associated with methods in the area of heuristics and metaheuristics.

Decision problems that have arisen in SCM are becoming more complex and larger; the need for integrating all the elements of the supply chain is an enormous challenge, as it relates to different objective functions, online decisions, big databases, and uncertainty data. Thus, this chapter argues that heuristics and metaheuristics are the most promising tools to solve the decision problems in SCM. The attributes of metaheuristics, as well as the successful cases mentioned in this work, support the argument that these methods can address the main challenges of the actual and future SCM problems.

A larger number of applications of heuristics in metaheuristics to problems in SCM are foreseeing shortly. The vast amount of available data, the need for more integration and coordination, the need for online solutions, the consideration of uncertainty, and many other new constraints will produce a significant amount of complex decision problems with the need for a fast and efficient solution method. The use of metaheuristics may provide a significant contribution in SCM from a theoretical and practical perspective.

Cross-References

▸ Genetic Algorithms
▸ GRASP
▸ Tabu Search
▸ Variable Neighborhood Search

Acknowledgments H.R. has been supported by the Spanish Ministry of Economy and Competitiveness (TRA2013-48180-C3-P). M.G.R. acknowledges supports from the National Council for Scientific and Technological Development (CNPq) and FAPEMIG.

References

1. Cordeau JF, Gendreau M, Laporte G et al (2002) A guide to vehicle routing heuristics. J Oper Res Soc 53:512–522. https://doi.org/10.1057/palgrave/jors/2601319
2. CSCMP Supply Chain Management. https://cscmp.org/about-us/supply-chain-management-definitions
3. Simchi-Levi D, Kaminsky P, Simchi-Levi E (2003) Designing and managing the supply chain: concepts, strategies and case studies. McGraw-Hill/Irwin, Boston
4. Janvier-James AM (2011) A new introduction to supply chains and supply chain management: definitions and theories perspective. Int Bus Res 5:194–207. https://doi.org/10.5539/ibr.v5n1p194
5. Ullrich CA (2014) Issues in supply chain scheduling and contracting. Springer Fachmedien Wiesbad, Wiesbaden. https://doi.org/10.1007/978-3-658-03769-7
6. Johnson JC, Wood DF, Wardlow DL, Murphy Paul R (1999) Contemporary logistics. Prentice-Hall, Upper Saddle River
7. Akdogan AA, Demirtas O (2014) Managerial role in strategic supply chain management. Procedia Soc Behav Sci 150:1020–1029. https://doi.org/10.1016/j.sbspro.2014.09.114
8. Lambert DM, Cooper MC (2000) Issues in supply chain management. Ind Mark Manag 29:65–83. https://doi.org/10.1016/S0019-8501(99)00113-3
9. Chopra S, Meindl P (2016) Supply chain management, 6th edn. Pearson Education, Boston
10. Chen IJ, Paulraj A (2004) Understanding supply chain management: critical research and a theoretical framework. Int J Prod Res 42:131–163. https://doi.org/10.1080/00207540310001602865
11. Farahani RZ, Rezapour S, Drezner T, Fallah S (2014) Competitive supply chain network design: an overview of classifications, models, solution techniques and applications. Omega 45:92–118. https://doi.org/10.1016/j.omega.2013.08.006
12. Melo MT, Nickel S, Saldanha-da-Gama F (2009) Facility location and supply chain management – a review. Eur J Oper Res 196:401–412. https://doi.org/10.1016/j.ejor.2008.05.007
13. Gosling J, Purvis L, Naim MM (2010) Supply chain flexibility as a determinant of supplier selection. Int J Prod Econ 128:11–21. https://doi.org/10.1016/j.ijpe.2009.08.029
14. Tachizawa EM, Gimenez C (2010) Supply flexibility strategies in Spanish firms: results from a survey. Int J Prod Econ 124:214–224. https://doi.org/10.1016/j.ijpe.2009.11.020
15. Chikán A (2011) Managers' view of a new inventory paradigm. Int J Prod Econ 133:54–59. https://doi.org/10.1016/j.ijpe.2010.09.009
16. Morton T, Pentico D (1993) Heuristic scheduling systems. Wiley, New York
17. Chan L, Shen Z, Simchi-levi D, Swann JL (2004) Coordination of pricing and inventory decisions: a survey and classification. In: Simchi-Levi D, Wu SD, Shen ZM (eds) Handbook of quantitative supply chain analysis modeling in the E-business era. Kluwer Academic, Boston, pp 335–392
18. Coelho LC, Cordeau J, Laporte G (2014) Thirty years of inventory routing. Transp Sci 48:1–19. https://doi.org/10.1287/trsc.2013.0472
19. Andersson H, Hoff A, Christiansen M et al (2010) Industrial aspects and literature survey: combined inventory management and routing. Comput Oper Res 37:1515–1536. https://doi.org/10.1016/j.cor.2009.11.009
20. Diabat A (2014) Hybrid algorithm for a vendor managed inventory system in a two-echelon supply chain. Eur J Oper Res 238:114–121. https://doi.org/10.1016/j.ejor.2014.02.061
21. Mateus G, Ravetti MG, de Souza M, Valeriano T (2010) Capacitated lot sizing and sequence dependent setup scheduling: an iterative approach for integration. J Sched 13:245–259. https://doi.org/10.1007/s10951-009-0156-2
22. Kreipl S, Dickersbach JT (2008) Scheduling coordination problems in supply chain planning. Ann Oper Res 161:103–122. https://doi.org/10.1007/s10479-007-0293-y
23. Kreipl S, Pinedo M (2009) Planning and scheduling in supply chains: an overview of issues in practice. Prod Oper Manag 13:77–92. https://doi.org/10.1111/j.1937-5956.2004.tb00146.x

24. Martínez-Jurado PJ, Moyano-Fuentes J (2013) Lean management, supply chain management and sustainability: a literature review. J Clean Prod 85:134–150. https://doi.org/10.1016/j.jclepro.2013.09.042
25. Demir E, Bektaş T, Laporte G (2014) A review of recent research on green road freight transportation. Eur J Oper Res 237:775–793. https://doi.org/10.1016/j.ejor.2013.12.033
26. Devika K, Jafarian A, Nourbakhsh V (2014) Designing a sustainable closed-loop supply chain network based on triple bottom line approach: a comparison of metaheuristics hybridization techniques. Eur J Oper Res 235:594–615. https://doi.org/10.1016/j.ejor.2013.12.032
27. Yücesan E (2007) Impact of information technology on supply chain management. In: Jung H, Jeong B, Chen FF (eds) Trends in supply chain design and management SE – 6. Springer, London, pp 127–148
28. Simchi-Levi D, Kaminsky P, Simchi-Levi E (2003) Designing and managing the supply chain: concepts, strategies and case studies. McGraw-Hill/Irwin, Boston
29. Hazen BT, Boone CA, Ezell JD, Jones-Farmer LA (2014) Data quality for data science, predictive analytics, and big data in supply chain management: an introduction to the problem and suggestions for research and applications. Int J Prod Econ 154:72–80. https://doi.org/10.1016/j.ijpe.2014.04.018
30. Kot S (2014) Principles of global supply chain management. Faculty of Management, Czestochowa University of Technology
31. Croxton KL, García-Dastugue SJ, Lambert DM, Rogers DS (2001) The supply chain management processes. Int J Logist Manag 12:13–36
32. Sánchez D, Amodeo L, Prins C (2009) Meta-heuristic approaches for multi-objective simulation-based optimization in supply chain inventory management. Lect Notes Comput Sci 5484:798–807. https://doi.org/10.1007/978-1-84996-119-6_9
33. Duan Q, Liao TW (2013) A new age-based replenishment policy for supply chain inventory optimization of highly perishable products. Int J Prod Econ 145:658–671. https://doi.org/10.1016/j.ijpe.2013.05.020
34. Zolfaghari S, Quan V, El-Bouri A, Khashayardoust M (2010) Application of a genetic algorithm to staff scheduling in retail sector. Int J Ind Syst Eng 5:20–47. https://doi.org/10.1504/IJISE.2010.029755
35. Wang KJ, Chen MJ (2009) Cooperative capacity planning and resource allocation by mutual outsourcing using ant algorithm in a decentralized supply chain. Expert Syst Appl 36:2831–2842. https://doi.org/10.1016/j.eswa.2008.01.089
36. Laporte G (2009) Fifty years of vehicle routing. Transp Sci 43:408–416. https://doi.org/10.1287/trsc.1090.0301
37. Yeun LC, Ismail WANR, Omar K, Zirour M (2008) Vehicle routing problem: models and solutions. J Qual Meas Anal 4:205–218
38. Cordeau JF, Gendreau M, Hertz A et al (2005) New heuristics for the vehicle routing problem. In: Langevin A, Riopel D (eds) Logistics systems: design and optimization. Kluwer Academic, Boston pp 279–297
39. Kumar SN, Panneerselvam R (2012) A survey on the vehicle routing problem and its variants. Intell Inf Manag 4:66–74. https://doi.org/10.4236/iim.2012.43010
40. Vidal T, Crainic TG, Gendreau M, Prins C (2013) Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur J Oper Res 231:1–21. https://doi.org/10.1016/j.ejor.2013.02.053
41. Pérez-Bernabeu E, Juan AA, Faulin J, Barros B (2015) Horizontal cooperation in road transportation: a multi- depot case illustrating savings in distances and pollutant emissions. Int Trans Oper Res 22:585-606. https://doi.org/10.1111/itor.12130
42. Juan AA, Faulin J, Pérez-Bernabeu E, Jozefowiez N (2014) Horizontal cooperation in vehicle routing problems with backhauling and environmental criteria. Proc Soc Behav Sci 111:1133–1141. https://doi.org/10.1016/j.sbspro.2014.01.148
43. Lin C, Choy KL, Ho GTS et al (2014) Survey of green vehicle routing problem: past and future trends. Expert Syst Appl 41:1118–1138. https://doi.org/10.1016/j.eswa.2013.07.107

44. Erdoğan S, Miller-Hooks E (2012) A green vehicle routing problem. Transp Res Part E Logist Transp Rev 48:100–114. https://doi.org/10.1016/j.tre.2011.08.001
45. Schmid V, Doerner KF, Laporte G (2013) Rich routing problems arising in supply chain management. Eur J Oper Res 224:435–448. https://doi.org/10.1016/j.ejor.2012.08.014
46. Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Comput Oper Res 34:2403–2435. https://doi.org/10.1016/j.cor.2005.09.012
47. Liu S, Papageorgiou LG (2013) Multiobjective optimisation of production, distribution and capacity planning of global supply chains in the process industry. Omega Int J Manag Sci 41:369–381. https://doi.org/10.1016/j.omega.2012.03.007
48. Afshin Mansouri S, Gallear D, Askariazad MH (2012) Decision support for build-to-order supply chain management through multiobjective optimization. Int J Prod Econ 135:24–36. https://doi.org/10.1016/j.ijpe.2010.11.016
49. Chu CH, Luh YP, Li TC, Chen H (2009) Economical green product design based on simplified computer-aided product structure variation. Comput Ind 60:485–500. https://doi.org/10.1016/j.compind.2009.02.003
50. Vieira P, Vieira SM, Sousa JMC et al (2014) Designing closed-loop supply chains with nonlinear dimensioning factors using ant colony optimization. Soft Comput https://doi.org/10.1007/s00500-014-1405-7
51. Griffis SE, Bell JE, Closs DJ (2012) Metaheuristics in logistics and supply chain management. J Bus Logist 33:90–106. https://doi.org/10.1111/j.0000-0000.2012.01042.x
52. Calvete HI, Galé C, Iranzo JA (2014) Planning of a decentralized distribution network using bilevel optimization. Omega 49:30–41. https://doi.org/10.1016/j.omega.2014.05.004
53. Delavar MR, Hajiaghaei-Keshteli M, Molla-Alizadeh-Zavardehi S (2010) Genetic algorithms for coordinated scheduling of production and air transportation. Expert Syst Appl 37:8255–8266. https://doi.org/10.1016/j.eswa.2010.05.060
54. Eskandarpour M, Masehian E, Soltani R, Khosrojerdi A (2014) A reverse logistics network for recovery systems and a robust metaheuristic solution approach. Int J Adv Manuf Technol 74:1393–1406. https://doi.org/10.1007/s00170-014-6045-7
55. Boukherroub T, Ruiz A, Guinet A, Fondrevelle J (2015) An integrated approach for sustainable supply chain planning. Comput Oper Res 54:180–194. https://doi.org/10.1016/j.cor.2014.09.002
56. Brandenburg M, Govindan K, Sarkis J, Seuring S (2014) Quantitative models for sustainable supply chain management: developments and directions. Eur J Oper Res 233:299–312. https://doi.org/10.1016/j.ejor.2013.09.032
57. Ahi P, Searcy C (2013) A comparative literature analysis of definitions for green and sustainable supply chain management. J Clean Prod 52:329–341. https://doi.org/10.1016/j.jclepro.2013.02.018
58. Seuring S (2013) A review of modeling approaches for sustainable supply chain management. Decis Support Syst 54:1513–1520. https://doi.org/10.1016/j.dss.2012.05.053
59. Lourenço H, Ribeiro R (2012) Strategies for an integrated distribution problem strategies for an integrated distribution problem. In: Montoya-Torres JR, Juan AA, Huaccho L et al (eds) Hybrid algorithms for service, computing and manufacturing systems: routing and scheduling solutions. IGI Global Books, Hershey, pp 98–121
60. Ribeiro R, Lourenço HR (2005) A new model and heuristic for a multi-period inventory-routing problem. In: Decision science institute of international conference, Barcelona, pp 805–815
61. Caceres-Cruz J, Arias P, Guimarans D et al (2014) Rich vehicle routing problem: survey. ACM Comput Surv 47:1–28. https://doi.org/10.1145/2666003
62. Cardona-Valdés Y, Álvarez A, Pacheco J (2014) Metaheuristic procedure for a bi-objective supply chain design problem with uncertainty. Transp Res B Methodol 60:66–84. https://doi.org/10.1016/j.trb.2013.11.010

63. Rego N, Claro J, Pinho de Sousa J (2014) A hybrid approach for integrated healthcare cooperative purchasing and supply chain configuration. Health Care Manag Sci 17:303–20. https://doi.org/10.1007/s10729-013-9262-y
64. Cavique L (2007) A scalable algorithm for the market basket analysis. J Retail Consum Serv 14:400–407. https://doi.org/10.1109/EPIA.2005.341294
65. Martins P, Ladrón A, Ramalhinho H (2014) Maximum cut-clique problem: ILS heuristics and a data analysis application. Int Trans Oper Res. https://doi.org/10.1111/itor.12120
66. Sadeghia A, Alem-Tabriza A, Zandieha M (2011) Product portfolio planning: a metaheuristic-based simulated annealing algorithm. Int J Prod Res 49:2327–2350. https://doi.org/10.1080/00207540903329338
67. April J, Glover F, Kelly J, Laguna M (2001) Simulation/optimization using "Real-World" applications. In: Proceedings of 2001 winter simulation conference, Arlington, pp 134–138
68. Ngai EWT, Xiu L, Chau DCK (2009) Application of data mining techniques in customer relationship management: a literature review and classification. Expert Syst Appl 36:2592–2602. https://doi.org/10.1016/j.eswa.2008.02.021
69. Grasas A, Juan AA, Lourenço HR (2014) SimILS: a simulation-based extension of the iterated local search metaheuristic for stochastic combinatorial optimization. J Simul https://doi.org/10.1057/jos.2014.25
70. Juan AA, Faulin J, Grasman SE et al (2015, in press) A review of Simheuristics: extending metaheuristics to deal with stochastic optimization problems. Oper Res Perspect 2:62–72. https://doi.org/10.1016/j.orp.2015.03.001
71. Giménez C, Lourenço HR (2004) E-supply chain management: review, implications and directions for future research. In: EUROMA INSEAD, Fontainebleau, pp 1021–1031
72. Davenport TH (2014) Big data at work dispelling the myths, uncovering the opportunities. Harvard Business School Press, Boston
73. Nam T, Pardo TA (2011) Conceptualizing smart city with dimensions of technology, people, and institutions. In: Proceedings of the 12th annual international digital government research conference on digital government innovation in challenging times – dg.o '11, p 282.https://doi.org/10.1145/2037556.2037602

The Maximum Clique and Vertex Coloring

44

Oleksandra Yezerska and Sergiy Butenko

Contents

Introduction 1260
Construction Heuristics 1261
Local Search 1264
Metaheuristics 1265
  Local Search-Based Methods 1265
  Population-Based Methods 1271
Heuristics Based on Continuous Formulations 1276
Other Heuristics 1278
Computational Results 1278
Conclusion 1281
Cross-References 1281
References 1281

Abstract

In this chapter we review heuristic approaches for two classical and closely related problems of finding a maximum clique and an optimal vertex coloring. Both problems have a wide variety of practical applications, and due to their computational intractability, a significant effort has been focused on developing heuristic methods. This chapter discusses construction heuristics, local search strategies, and metaheuristics designed and/or adapted for the maximum clique and vertex coloring problems.

O. Yezerska · S. Butenko (✉)
Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX, USA
e-mail: yaleksa@tamu.edu; butenko@tamu.edu

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_47

Keywords
Maximum clique · vertex coloring · chromatic number

Introduction

Given a simple graph $G = (V, E)$, a *clique* is defined as a subset $C \subseteq V$ of mutually adjacent vertices. A graph is called *complete* if the set of its vertices forms a clique. A *maximal clique* is a clique that cannot be extended to a clique of larger size by simply adding a new vertex to this clique. The *maximum clique problem* asks for a clique of the largest cardinality in the graph. The size of a maximum clique is called the *clique number* of G and is usually denoted as $\omega(G)$.

An *independent set* (or *stable set*) is a subset $I \subseteq V$ of vertices with no edge between them. It is easy to see that a *clique* in G is an *independent set* in the *complement graph* of G, $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(i, j) : (i, j) \notin E \ \forall i, j \in V, i \neq j\}$, and vice versa. Finding a maximum clique in G is equivalent to finding a maximum independent set in $\bar{G}$. The cardinality of a maximum independent set in G is called the *independence* or *stability number* of G, denoted $\alpha(G)$, and $\alpha(\bar{G}) = \omega(G)$.

A *proper vertex coloring* (also referred to as a coloring for simplicity) is an assignment of a color (or a label) to each vertex in a graph such that no adjacent vertex has the same color. More formally, given a set of colors $\{1, 2, \ldots, k\}$, a *proper k-coloring* is a mapping $f : V \to \{1, 2, \ldots, k\}$, where $f(v_i) \neq f(v_j)$ $\forall (i, j) \in E$. Then the vertices with the same value of f belong to the same *color class*. The *chromatic number* of a graph G, denoted $\chi(G)$, is the minimum number of colors necessary to color G. An *optimal coloring* of a graph is a proper coloring that uses exactly $\chi(G)$ colors.

It is easy to see that the vertices that belong to the same color class form an independent set and the vertices of a clique all belong to different color classes. Then it follows that for a given graph G:

$$\chi(G) \geq |V|/\alpha(G),$$

and

$$\chi(G) \geq \omega(G).$$

The maximum clique and vertex coloring problems often arise in similar applications, where they play complementary roles. For example, let the vertices in the graph represent elements of a system and the edges between them – the incompatibility between those elements. Then a maximum clique will represent a set of mutually incompatible elements of the largest size in the system, whereas the color classes will correspond to sets of mutually compatible elements.

Both problems find a wide range of practical applications in various domains. Some of the earliest and best-known applications of the maximum clique problem

are in computer vision, experimental design, information retrieval, coding theory, fault diagnosis, social network analysis, and computational biochemistry and genomics, among other areas [32,150,182]. As for the coloring, it is commonly used for scheduling [125, 140], timetabling [173, 174, 178], frequency assignment [74, 176], circuit board testing [76], register allocation [38, 42], and, more recently, various problems in transportation [35, 177]. A discussion of both clique and vertex coloring applications in telecommunications is presented in [12].

The maximum clique and vertex coloring problems are both well-known NP-hard problems [75, 119] that are hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, where $n = |V|$ [6,7,59,60,97,187]. Such intractability results, along with the practical interest, have led to a considerable effort in devising numerous heuristic approaches. Almost all known types of heuristic methods have been applied to these problems. Some of the early surveys on heuristic methods for the vertex coloring problem can be found in [151, 175] and more recent in [41, 71, 73, 132, 153], among which [41, 71] review just the local search methods and [73] focuses on the recent approaches. As for the maximum clique problem, the first extensive surveys date back to the 1990s [22, 150], while the most recent one [182] focuses mostly on local search techniques. Other brief discussions on heuristic approaches for the maximum clique problem can be found in [150, 153]. It should be noted that most of these surveys also discuss exact algorithms, many of which take advantage of heuristics to enhance their performance.

The remaining sections of this chapter are organized as follows. The most common construction heuristics for the maximum clique and vertex coloring problems are discussed in section "Construction Heuristics." In section "Local Search," we briefly outline the local search strategies used for both problems. Section "Metaheuristics" covers various metaheuristic approaches. It is followed by Sections "Heuristics Based on Continuous Formulations," "Other Heuristics," "Computational Results," and "Conclusion," respectively.

Construction Heuristics

Construction heuristics are used to build an initial feasible solution. The most intuitive and common construction heuristics are *sequential* ones, which recursively update the current, partial (and not necessarily feasible) solution one step at a time. Based on how the next step is selected, most of these heuristics can be classified as either greedy, random, or a combination of both. Greedy selection rules typically aim to ensure the highest level of flexibility (i.e., the largest possible "candidate set") in the future steps.

The most common sequential greedy heuristics for the maximum clique problem are *best in* and *worst out*, as named by Kopf et al. in [123]. Best in algorithms are those that start with an empty set as an initial solution and iteratively add a vertex with the largest degree among the candidate vertices, whereas worst out are the ones that start with a whole graph and recursively delete vertices with the lowest degree until the remaining graph is complete. The best in heuristic applied for finding the

maximal independent set yields a solution of size at least $\frac{|V|}{\delta+1}$, where $\delta = \frac{2|E|}{|V|}$ [58]. The best in heuristic was also shown to be "provably best" for the maximum clique problem in [115]. A heuristic is called provably best if no other polynomial-time algorithm can guarantee a better solution whenever one exists (assuming $\mathcal{P} \neq \mathcal{NP}$).

As for the vertex coloring problem, any heuristic that finds a coloring using at most $\Delta(G)$ colors (referred to as Brooks' coloring), where $\Delta(G)$ is the maximum vertex degree in G, is provably best [115]. According to Brooks' theorem [24], a simple, connected graph G that is neither complete nor an odd cycle satisfies $\chi(G) \leq \Delta(G)$. Brooks' coloring can be obtained using several heuristics [85, 91, 116, 118, 128, 163].

Many popular heuristics for the vertex coloring problem are based on sequential construction approaches. Sequential construction heuristics usually iteratively assign each vertex with the smallest feasible color such that no conflicts with already colored vertices occur. The order in which vertices are colored is either static and determined beforehand (preorder) or dynamic and updated on the fly. It is easy to see that the quality of the solution depends entirely on how the vertices are ordered and that there exists such vertex ordering that produces an optimal coloring [175]. Therefore, a lot of attention has been placed on studying and designing different ordering schemes.

Two examples of preorder sequential coloring heuristics are the *largest first* (LF) [173] and the *smallest last* (SL) [139]. LF sorts the vertices in decreasing order of their degree, and the ordering $(v_1, \ldots, v_n)$, produced in such manner, results in the coloring with at most $1 + \max_{1 \geq j \geq n} d_j(v_j)$ colors, where $d_j(v_j)$ is the degree of vertex v_j in the induced subgraph $G_j = G[\{v_1, \ldots, v_j\}]$. SL is similar to LF, but it sorts the vertices in the reverse order. First, a vertex of the minimum degree in graph G is selected and labeled v_n. Then, iteratively for each $j = n-1, \ldots, 1$, a vertex of the minimum degree in G_j is selected and labeled v_j.

A notable sequential coloring approach that does not preorder vertices is the DSATUR [23] heuristic. A decision on what vertex to color next is based on the value of the vertex *saturation degree* – a number of the differently colored neighbors of this vertex. A vertex with the maximum saturation degree is selected. Another example of a heuristic that does not preorder vertices is the *recursive largest first* (RLF) [125]. This heuristic generates a color class one at a time and does not proceed to another color class until no more vertices can be assigned to the current class. Such an assignment is implemented in the following manner. When a new color class is generated, set U_1 contains all uncolored vertices, and set U_2 is empty. Iteratively, a vertex $v \in U_1$ is selected, assigned to the current color class, and removed from U_1. If v has any neighbors in U_1, they are removed from U_1 and placed in U_2. The vertex assignment proceeds while U_1 is not empty. The very first vertex to be assigned to the new color class is the one with the maximum degree in $G[U_1]$, and all the rest are the ones with the maximum degree in $G[U_2]$. When U_1 becomes empty, the next color class is generated, and the process repeats.

In another study [20], a maximal independent set is iteratively extracted from the set of uncolored vertices, and all the vertices forming this independent set are

assigned the same color. The process repeats until the whole graph is colored. Similar technique is used in the algorithm called XRLF [114], which is a version of RLF [125]. Several independent sets are first constructed, and then the one whose removal yields a residual graph with the smallest edge density is selected as a new color class. This process is repeated until a threshold number of uncolored vertices are left, which are then colored using an exhaustive search. Independent set extraction approach has been used to construct initial colorings within various metaheuristic frameworks [39,66,73,99,142]. Recently, a few enhanced approaches based on independent set extraction were proposed. Namely, in [179], the authors suggest to preprocess large graphs by iteratively extracting a maximal number of pairwise disjoint independent sets of the same size. Such strategy was shown to assign more vertices to the same number of color classes than that with a one-at-a-time independent set extraction. This yields smaller in size residual graphs of uncolored vertices, which are easier to color. In [96, 181], the authors observe that an independent set extracted may not necessarily define a color class in an optimal coloring and that removing such sets during preprocessing may prevent one from reaching an optimal solution. To mitigate this drawback, the authors of [96,181] suggested a framework that reconsiders a certain number of the removed independent sets and allows some of the vertices forming those sets to change a color. Different strategies on how many and which extracted sets to reexamine were studied, and the experiments on some large and hard graph instances reported in [96, 181] demonstrate an encouraging performance of the proposed approach.

To improve the quality of solutions generated by the greedy sequential methods, different techniques ranging from randomization, multiple restarts, and local search (section "Local Search") to sophisticated metaheuristic frameworks (section "Metaheuristics") can be used. We will cover here a few simple strategies and discuss the rest in the later sections.

For the maximum clique problem, Jagota et al. [109] proposed several enhancements to a basic greedy sequential heuristic, which include randomization, multiple restarts, and an adaptive mechanism. A randomization is embodied by a probabilistic greedy vertex selection rule. Three kinds of adaptation are studied: an update of the initial state (AI), an update of the probability distribution used by a selection rule (AW), and no adaptation at all (NA). The corresponding updates take place at each restart and are based on the information obtained in the previous restart. NA heuristic was shown to perform poorly compared to AI and AW. Grosso et al. [87] used similar techniques and developed a two-phase heuristic, called a *deep adaptive greedy search* (DAGS). In the first phase of DAGS, the modified variant of a sequential greedy heuristic is applied for all the vertices in the graph. The modification consists of allowing to swap already added vertices with the better candidates. The second phase is used for diversification purposes. It is a restart-adaptive greedy heuristic performed on a set of vertices that appear less frequently in the cliques obtained during the first phase. Each vertex is assigned a weight associated with its quality as a potential candidate for a solution. Such weight is updated at each restart by means of a learning-like mechanism.

For the vertex coloring problem, an improvement procedure called an *iterative greedy* (IG) heuristic was proposed in [46, 47]. It is based on the observation that given a feasible coloring and an ordering in which vertices belonging to the same color class are grouped together, applying a sequential greedy heuristic to such an ordering will produce a solution at least as good as the previous one.

Local Search

Local search is a technique of improving a current solution by exploring its local neighborhood and moving to better neighboring solutions until no more improvement can be achieved. A local neighborhood of a solution is defined by a *neighboring function* which can also be thought of as a distance or a difference between neighboring solutions. Local search methods vary based on the definition of a neighboring function and a corresponding local neighborhood, as well as the choice of a search space and a function that evaluates the quality of a solution.

As suggested in [182], local search strategies for the maximum clique problem can be categorized into *legal* and *k-fixed penalty* ones. A legal strategy is a traditional local search, where a search space consists of legal (feasible) solutions, cliques; an evaluation function is the size of a clique; and a neighboring function is a list of vertex operations to be performed to move from one clique to another. The most common neighboring functions are (a, b)-interchanges, where a is the number of vertices removed from the current solution and b is the number of added vertices, with a usually smaller than b to ensure that such interchange increases the size of the solution by $(b - a)$.

A k-fixed penalty strategy, on the other hand, can be thought of as a technique that solves the decision version of the problem that answers the question "is there a clique of size k?" This type of a local search deals with infeasible solutions, sets that are not necessarily cliques. An evaluation function is defined as the number of edges in the set. By moving to sets with larger number of edges, a local search attempts to eventually reach a feasible solution – a clique. For this type of a local search, a (1,1)-interchange is usually used, where a vertex to be removed is usually the one with the smallest number of neighbors in the solution and a vertex to be added is the one adjacent to the largest number of solution members. Such a technique does not guarantee finding a feasible solution; therefore, it is usually used as a part of a more sophisticated local search framework (section "Metaheuristics").

For the vertex coloring problem, the local search strategies can also be categorized into legal and k-fixed ones. Moreover, as discussed in [71], they can be further grouped into *legal*, *k-fixed partial legal*, *penalty*, and *k-fixed penalty* ones. Here, the legal and the k-fixed penalty strategies, similarly to the local search strategies for the maximum clique problem, are the local searches on either feasible (proper) colorings or infeasible k-coloring, respectively. For the legal strategy, the local neighborhood is explored in an attempt of finding a feasible coloring with fewer number of colors. An infeasible coloring is such that contains at least one pair of adjacent vertices assigned the same color. An edge connecting a pair of such

vertices is referred to as *conflicting*. Then for this strategy, a local search aims at performing moves resulting in the minimization of the number of conflicting edges in the solution.

Under a k-fixed partial legal strategy, the search space contains partial feasible k-colorings. A partial feasible k-coloring corresponds to a coloring of a subset of vertices $V' \subset V$ that uses k colors and is feasible with respect to already colored vertices in V'. The rest of the vertices in $V \setminus V'$ are not yet colored. The local neighborhood is explored in an attempt to find a complete feasible k-coloring. For a pure penalty strategy, the search space contains infeasible colorings, and the number of colors used is unfixed. The local search under this strategy tries to detect a feasible coloring with the minimum number of colors used. The quality function usually combines both the number of conflicting edges and the number of colors used.

The most common neighboring functions for coloring are also interchanges [139]. They are performed by moving vertices from one color class to another, in an attempt of emptying one of the classes and, thus, reducing the number of colors used. Under the penalty and k-fixed penalty strategies, such interchanges are allowed to result in infeasible solutions.

Local search approaches also vary based on the rules of determining when to move to a better solution. By the *best improvement strategy*, the entire local neighborhood is explored, and the best neighbor is selected as a new solution, whereas the *first improvement strategy* updates the current solution with the first found neighbor of a better quality. However, with either of these strategies, a local search tends to get trapped in local optima of poor quality. To overcome this drawback and explore more regions of a search space, different techniques, e.g., allowing non-improving moves and recording the search process to direct the future exploration, have been devised and will be discussed in the following section.

Metaheuristics

In this section, we will review the most common metaheuristics for the maximum clique and vertex coloring problems. The metaheuristics can be classified with respect to various criteria [19]. Based on the number of solutions maintained at the same time, the algorithms can be divided into single-point search ones, encompassing most local search-based heuristics (section "Local Search-Based Methods"), and population-based ones (section "Population-Based Methods"), which deal with a pool of solutions at a time and perform evolution-like search processes.

Local Search-Based Methods

The most successful techniques to enhance local search have proven to be the ones that allow non-improving moves in the process of local neighborhood exploration. One of them, *simulated annealing* (section "Simulated Annealing"), allows moving to a worse solution with a certain probability, while another one, *tabu search*

(section "Tabu Search"), records the moves and forbids (tabu) their future usage for a certain number of iterations to prevent cycling. Other successful local search-based methods have been devised and will be discussed in section "Other Local Search-Based Methods".

Simulated Annealing

Simulated annealing (SA) method is a randomized neighborhood search introduced by Kirkpatrick et al. [121] and covered in a separate chapter of this book. SA is analogous to a physical annealing – a process of heating up a solid material and slowly cooling it down in order to obtain a low energy configuration. The main idea of SA is to iteratively select random candidate solution in the local neighborhood and, if it is of better quality, move to the candidate solution; otherwise perform the move with a certain probability of acceptance. Such probability is usually defined as $\exp(-\Delta)/T$, where Δ is a difference between the objective function values of the new and the current solutions and T is a parameter called the *temperature*. As the heuristic progresses, the probability is being decreased, ensuring fewer random walks and directing the search toward improvement. The rate at which the probability decreases is usually referred to as a "cooling schedule." The cooling schedule that follows the logarithmic law guarantees the convergence of the algorithm to a global optimum [19]; however, it may result in an exponential running time. Hence, faster cooling schedules are usually used in practice.

The most influential papers devoted to application of SA to the maximum clique problem date back in the late 1980s and mid-1990s and are all covered in [22]. Since, to the best of our knowledge, no novel SA methods for the maximum clique problem have been designed since then, we will follow [22] and briefly review here the same studies.

An application of SA to the maximum clique (independent set) problem was first described by Aarts et al. in 1988 [1]. Without providing any experimental results, they proposed a *penalty function* method (k-fixed penalty strategy), for which a possible solution can be any set, not necessarily independent, and the quality of the solution is determined by the function $f(V') = |V'| - \lambda|E'|$, where V' and E' are the vertex and edge sets of the solution and λ is a certain weighting factor. A few years later, Jerrum et al. [111] have theoretically proven a poor performance of SA application to the maximum clique problem. In particular, they studied a variant of SA with the temperature parameter fixed and a solution space consisting of legal cliques (legal strategy). Feo et al. [64] implemented SA for the maximum independent set problem utilizing the penalty method and observed that it was inferior to the greedy randomized adaptive search procedure (GRASP) they proposed. However, the computational experiments presented by Homer et al. [104, 113] in the Second DIMACS Implementation Challenge showed that SA can be very effective. In their work, Homer et al. used the idea of penalty method described by Aarts and implemented it for the maximum clique problem. The results of the experiments on large graphs, reported in [104, 113], were quite promising, and SA was ranked one of the best heuristics in the Second DIMACS Implementation Challenge.

One of the first attempts to apply SA to the vertex coloring problem dates back to 1987, when Chams et al. [39] considered the following variant of the method. The neighborhood structure consists of k-colorings that are not necessarily legal (k-fixed penalty strategy), and the objective function to be minimized is defined by the number of conflicting edges. According to the experimental results reported in [39], the pure SA did not perform as strongly as when it was combined with other methods. In particular, when SA was combined with RLF [23], where RLF was used to generate color classes and SA to color the rest of the uncolored vertices, the performance of such combined method dominated all other methods tested in the paper (DSATUR and RLF). In 1991, Johnson et al. [114] conducted a detailed study of different schemes of SA adopted to the vertex coloring problem. The first scheme tested is a penalty function method, where the neighborhood structure is defined by infeasible solutions (penalty strategy), i.e., colorings with conflicting edges. The number of colors used in a particular solution is not fixed. Let C_i denote a color class, let E_i be an edge set in the subgraph induced by C_i ($i = 1, \ldots, k$), and let $\Pi = C_1, \ldots, C_k$ ($1 \leq k \leq |V|$) be a solution. Then the cost (penalty) function associated with Π is defined as follows:

$$f(\Pi) = \sum_{i=1}^{k} 2|C_i| \cdot |E_i| - \sum_{i=1}^{k} |C_i|^2.$$

The first component of $f(\Pi)$ favors independent sets, while the second one favors large color classes. Minimization of $f(\Pi)$ results in eliminating the conflicting edges and reaching a legal coloring and also, as the authors argued, in potentially reducing the number of color classes as a side effect. The second scheme is by Morgenstern and Shapiro [143], and it utilizes so-called *Kempe chains*, which are defined in [114] as the connected components of the union of two disjoint independent sets. In this method, the same cost function as in the penalty function method is retained, but the search space contains legal k-colorings. The new candidate solution is generated in the following way. Two color classes C_i and C_j ($i \neq j$) are picked at random, and a Kempe chain defined by a set $H \subset C_i \cup C_j$ is constructed. Next C_i is replaced by $C_i \ \Delta \ H$ and C_j – by $C_j \ \Delta \ H$, where $X \ \Delta \ Y$ denotes a symmetric difference $(X - Y) \cup (Y - X)$ between X and Y. The third scheme uses the same strategy as in [39], where the number k of colors is fixed and the objective is to minimize the number of conflicting edges in an attempt to find a feasible k-coloring. According to the results of the experiments reported by Johnson et al., none of the three methods was found dominant over the other.

A simplified version of SA with the temperature parameter remaining fixed was used by Morgenstern [113, 142] for the Second DIMACS Implementation Challenge. In this paper, a new neighborhood structure, called *impasse neighborhood*, was defined. It turned out to be very effective and many approaches have used it thereafter. This neighborhood structure operates with a fixed number k of colors and aims to improve a partial coloring, in which not all vertices are colored, to a complete coloring with k colors. A feasible solution is a partition of the set of

vertices into $k + 1$ subsets $\{V_1, \ldots, V_k, V_{k+1}\}$, where $V_1, \ldots, V_k$ are independent sets representing a partial k-coloring and V_{k+1} is the set of uncolored vertices that needs to eventually be emptied in order to obtain a proper k-coloring of all vertices. A neighbor of a solution is obtained by moving a vertex v from V_{k+1} to one of the k independent sets, say V_i, and then moving to V_{k+1} all vertices from V_i that are adjacent to v.

Tabu Search

Tabu search was developed by Glover [81] and, independently, by Hansen and Jaumard [93]. See the ▸ Chap. 25, "Tabu Search" for more details. It is a modified local search which allows performing moves yielding worse quality solutions, thus diversifying the search path. The algorithm uses the so-called *tabu lists* to store certain information about the previously visited solutions to forbid future local search moves that can result in cycling. The length of the tabu lists, called *tabu tenures*, allows to control the level of diversification and intensification of the search. Sometimes the tabu restriction is relaxed if a solution meets certain *aspiration level* condition.

In 1989, Friden et al. [68] used tabu search ideas to construct a heuristic called STABULUS for finding stable (independent) sets. STABULUS is based on the k-fixed penalty local search strategy (see section "Local Search"), where the search space consists of the sets of size k, and the algorithm tries to minimize the number of edges in the set. To find an independent set of the largest cardinality, STABULUS is applied iteratively, incrementing k each time an independent set of size k is detected. Three tabu lists were used in STABULUS, one for storing visited solutions, and the other two contained added and deleted vertices.

A few years later, Gendreau et al. [78] proposed two simple variants of tabu search for the maximum clique problem, deterministic and probabilistic, where in the former, at each iteration, the entire set of the solution neighbors is explored and, in the latter, the sampling of the neighborhood is applied. In both variants the legal local search strategy was exploited, where the search space contains the feasible cliques and the objective is to find a clique of the maximum size. In the deterministic approach, it is allowed to remove only one vertex per iteration, while in the probabilistic it is a few at a time. Later these heuristics were enhanced with the diversification strategies [166], such as greedy restart and continuous diversification. The first one is a very simple approach of restarting the algorithm with the new solution after a certain number of iterations without improvement have been performed. A new solution is constructed from vertices that have not been visited yet or have been visited less frequently. The continuous diversification works in "on-the-fly" mode. It guides the search path by evaluating each vertex addition/removal using the information on how frequently the vertex has been added to the solution and how long it has stayed in the solution. Both diversification techniques and their combination have proven to be beneficial if used along with the simple basic search. The more efficient and sophisticated the search is, the less improvement the diversification is able to add to it. It also did not appear to work

well with the probabilistic version of the tabu search heuristic. The heuristics were tested on DIMACS instances, and the results were reported in [167].

A similar idea of prohibiting the future moves in order to prevent cycling was used in reactive local search (RLS) [14]. However, contrary to the traditional tabu search, where certain parameters (tabu tenure, depth of the search, etc.) are determined by the user through numerous preliminary experiments and are usually sensitive to certain specific structures of the instances, RLS determines all of this information dynamically within its execution. The length of tabu lists is adjusted based on previously stored information on how often the algorithm cycles, etc. Also, the algorithm is equipped with a memory-influenced restart procedure to provide additional long-term diversification. RLS showed excellent performance on DIMACS instances and is considered one of the best local search-based heuristics for the maximum clique problem.

Recent efforts of applying tabu search to the maximum clique (independent set) problem include works by Wu et al. [112, 180, 183]. In [180], the authors developed an adaptive multistart tabu search approach that uses a k-fixed penalty local search strategy. The exploration of a search space is performed by moves that swap a vertex from the current solution with a vertex from the solution constrained neighborhood. The algorithm employs a restart strategy, under which a new solution is constructed from the vertices that were less frequently used throughout the algorithm, that way visiting unexplored search space regions. The algorithm showed excellent performance when compared against several other state-of-the-art algorithms. The study of the relationship between the number of restarts and the quality of the evaluation function performed by the authors suggested that for structured instances, more frequent restarts yield better results, while the reverse holds for random graphs.

For the vertex coloring problem, tabu search techniques were first applied by Hertz and de Werra [99]. In their proposed algorithm, named TABUCOL, the search space consists of the complete k-colorings, where k is fixed. Some colorings might have conflicting edges, and, hence, the algorithm's goal is to move to the solutions with smaller number of conflicting edges by changing the color assignment of the endpoints of conflicting edges. The tabu lists store these color assignments. The algorithm uses the same local search strategy as in [39] but yields better results than those obtained by SA in [39].

Recently, Blöchliger and Zufferey [18] designed several variations of the tabu search framework for the vertex coloring problem. The main ingredients they used were the partial k-coloring local search strategy and reactive tabu tenure, adjusted based on the fluctuations of the objective function.

Tabu search has also been applied to both the maximum clique and vertex coloring problems as a part of more complex metaheuristics, such as evolutionary algorithms, yielding the so-called *hybrid algorithms* (discussed in section "Population-Based Methods").

Other Local Search-Based Methods

The greedy randomized adaptive search procedure (GRASP) [62, 63], described in the ▸ Chap. 16, "GRASP" in this volume, is a simple metaheuristic that combines

a constructive heuristic, a local search, and a restart policy. Its construction part uses both greedy and random selection rules. More precisely, the initial solution is built by randomly selecting a vertex from a so-called *restricted candidate list* (RCL), which consists of a certain number of the best candidates. This number ranges between 1 and the size of the candidate list, where the closer it is to 1, the greedier the selection rule is, and the closer it is to the candidate list size, the more randomness is involved. Every time a solution is constructed, an attempt to improve it is made using an appropriate local search technique (see section "Local Search"). The algorithm restarts a certain number of times, and the best found solution is recorded. GRASP has been successfully applied to the maximum clique (maximum independent set) [2, 64, 161] and the vertex coloring problems [124].

Variable neighborhood search (VNS) [94, 141] is a metaheuristic that systematically explores different neighborhood structures of the same problem in the search for a better solution. This method is introduced in the ▸ Chap. 26, "Variable Neighborhood Search". VNS approach was applied to the maximum clique problem in [95]. For the vertex coloring, an algorithm based on VNS was proposed in [8]. It uses more than ten different neighborhoods and utilizes tabu search techniques. An extension of VNS, called *variable space search* (VSS) was proposed for the vertex coloring problem by Hertz et al. in [101]. In VSS, when the search is trapped in a local optimum, the entire search space, including the neighborhood searched and the objective function, is changed. The proposed algorithm, called VSS-Col, explores three different search spaces for the vertex coloring problem and yields excellent results. An *iterated local search* (see the ▸ Chap. 19, "Iterated Local Search" in this handbook for more information), which upon reaching a local optimum perturbs it so that the local search can keep going, was applied to the vertex coloring problem in [40, 148]. In [40], the authors analyzed the performance of different types of solution perturbations.

An algorithm based on *variable depth search* for the maximum clique problem, based on k-*opt* local search (KLS), was proposed by Katayama et al. [120]. The k-opt search is essentially an (a, b)-interchange with $a > b$ and $a + b = k$, where a vertices are removed from the current clique and b vertices from the local neighborhood are added to the solution yielding a clique of a larger size. The value k is determined dynamically throughout the execution of the algorithm. The reported results on selected DIMACS instances show the algorithm's competitiveness with RLS [14].

Pullan and Hoos [159] introduced a dynamic local search (DLS) algorithm for the maximum clique problem. DLS is based on alternating between a clique expansion phase, during which vertices are added to the current clique, and plateau search, during which vertices of the current clique are swapped with vertices outside of the current clique. The selection of vertices is based on the penalties assigned to them dynamically during the search in a similar way used in DAGS [87]. When no more expansion or swapping is possible, a perturbation mechanism is applied to overcome search stagnation. The algorithm achieves substantial improvements over some state-of-the-art algorithms on many DIMACS instances. However, the algorithm has a disadvantage of being sensitive to the penalty delay parameter,

which controls the frequency at which vertex penalties are decreased and needs to be determined beforehand by preliminary experiments and thorough calibration. This issue has been addressed in [158], where a modified algorithm, called a Phased Local Search (PLS), was proposed. Similarly to DLS [159], the algorithm is a combination of a clique expansion and a plateau search. Moreover, it operates three sub-algorithms, which differ in their vertex selection rule: random selection, random selection within vertex degree, and random selection within vertex penalties. In the latter one, the parameter responsible for the frequency of penalty decreases does not have to be fixed and defined externally as in DLS but is modified adaptively. PLS has shown to have a comparable and sometimes improved performance to DLS. Grosso et al. [88] performed a detailed computational study of the main components of DLS, such as usage of vertex penalties as a selection rule, plateau search, etc., in an attempt to design simple and efficient iterated local search techniques for the maximum clique problem. Another study that analyzes the key ingredients of RLS and DLS was performed by Battiti et al. in [13] and yielded several enhanced modifications of the mentioned algorithms.

Recently, Pullan et al. [160] introduced a parallelized hyper-heuristic algorithm for the maximum clique problem, called a cooperating local search (CLS). Hyper-heuristics [29] are heuristic search methods that manage the low-level heuristics. In particular, CLS is a methodology that controls the four PLS-based heuristics and dynamically reconfigures their allocation to cores, based on information retrieved from a trial in order to ensure the appropriateness of selected heuristic for a particular instance. In [5], Andrade et al. introduce efficient implementations of $(1, 2)$ and $(2, 3)$-interchange legal strategy local searches for the maximum independent set problem. These two local searches are integrated into an iterated local search metaheuristic, which according to the results reported in [5] yield great results, especially on large and difficult instances. Benlic and Hao [15] devised an algorithm, called breakout local search (BLS). BLS is an iterated local search, which, when a local optimum is discovered, applies a certain diversification strategy to perturb the current solution, so it can be used as a starting point to explore another region of the search space. The magnitude of perturbation is adapted dynamically according to the current search state. The results of the experiments showed that BLS competes favorably with RLS [14], VNS [95], and PLS [158].

Population-Based Methods

As opposed to local search algorithms, where one solution is maintained at each iteration, the population-based algorithms deal with several solutions (a *population*) at a time. The new population is produced by certain interactions between the members of a current population. Namely, the population generation techniques in *evolutionary* and *genetic algorithms* (section "Evolutionary and Genetic Algorithms") are based on imitating the mechanisms of evolution, whereas in *ant colony algorithms* (section "Ant Colony Optimization"), the solution construction is inspired by the behavior of ants seeking paths to food sources.

Evolutionary and Genetic Algorithms

Evolutionary and genetic algorithms [82, 103] (see also the corresponding chapters in this volume) are inspired by the evolution and natural selection and are based on the following evolution principles: "the fittest survives," "two fit parents will have even fitter offsprings," and mutation. In the genetic algorithms, these principles are embodied with the help of respective mechanisms: *reproduction*, *crossover*, and *mutation* operators. At each iteration (in evolutionary terminology – *generation*) of an algorithm, the population of solutions (*individuals*) is subjected to all or some of those operators. Reproduction operator selects the "fittest" solutions, where the fitness level is represented by a *fitness function* (or objective function in optimization terms). Then, with the help of the crossover operator, the "offspring" solutions are produced as a result of merging one or several parts of one parent solution with one or several parts of another parent solution. The mutation operator changes a small part of a solution to provide randomization.

For the maximum clique problem, the most intuitive way to encode a solution is to use a binary array of the size equal to the cardinality of the given graph's vertex set, where an entry contains 1, if the corresponding vertex is included in the current solution, and 0, otherwise. The definition of the solution fitness function can vary. A fairly simple fitness function assesses the size of a vertex subset if it is a clique, or equals 0, otherwise. It can also be a little more complex. For example, it can combine several terms, such as the density of the induced subgraph and its size. It can also include a term to penalize the infeasible solutions.

The early work on adapting genetic algorithms to the maximum clique problem is covered in great detail in [22]. We will review in the following the same studies and also discuss several recent advancements.

One of the earliest studies on the effectiveness of genetic algorithms at solving the maximum clique problem was done by Carter et al. [37]. The authors showed that the pure genetic algorithm performs poorly and designed several modifications, including an annealed fitness function and the modified crossover operators, to improve its performance. However, their results were still not satisfactory and led to a conclusion that genetic algorithms have to be significantly customized to be able to perform well and that they are very computationally expensive. Their later studies in [152] report that genetic algorithms are ineffective for solving combinatorial optimization problems and are inferior to other heuristics, such as simulated annealing. However, in another study [9], Back et al. showed quite the contrary results. They designed a genetic algorithm called GENEsYs with a fitness function which included a component for penalizing infeasible solutions. The performance of GENEsYs for the maximum independent set problem on graphs with up to 200 vertices was promising and suggested that with the right fitness function, genetic algorithms are capable of yielding satisfactory results. In another approach, Murthy et al. [146] introduced a new crossover mechanism, called a *partial copy crossover*, and suggested to split a mutation operator into a *deletion* and *addition* operators. The results presented in the paper are based on experiments with small graphs (up to 50 vertices), making it difficult to evaluate the algorithm's effectiveness.

In [67], Foster and Soule developed two variations of genetic algorithms for the maximum clique problem. One of them uses a different problem-encoding approach, referred to as a *grouping representation*, which instead of encoding each solution as a binary string, uses an array that stores information about all the solutions in the current population. Such encoding scheme showed to increase effectiveness of the crossover operation. The other variant of the algorithm uses the time-weighted fitness function, which showed to improve the algorithm's performance. Hifi [102] adopted a genetic algorithm for solving the maximum weighted independent set problem. The necessary algorithm modifications included an introduction of a so-called *two-fusion crossover operator* and a replacement of a mutation operator by a *heuristic-feasibility operator*. The two-fusion crossover operator takes into account both the structure and the fitness of two parent solutions and produces two children, while the heuristic-feasibility operator transforms infeasible solutions into feasible ones. The main contribution of the designed approach is that it is easily parallelizable.

Aggarwal et al. [4] designed an optimized crossover mechanism (inspired by a heuristic CLIQMERGE proposed in [10] by Ballas and Niehaus) and applied it to the maximum independent set problem. The new crossover operator exploits the structure of the solution rather than its encoding. More precisely, when a crossover operator is applied to two solutions from the population, one of the new solutions is generated in such a way that it has the best fitness function value, while the other one is constructed in a way that ensures the diversity of the search space. Promising results of the proposed mechanism inspired authors of CLIQMERGE, Balas and Niehaus, to examine this strategy even further [11]. The authors also considered a different way of population replacement called a *steady-state replacement* and a different selection method. The results of the experiments reported in [11] were even more encouraging than those by Aggarwal et al. in [4].

Since most of pure genetic algorithms have shown to be not very effective for solving hard combinatorial optimization problems, they are usually enhanced by incorporating different techniques. Bui and Eppley [26] designed a hybrid strategy with vertex preordering phase and a local optimization at each iteration of the genetic algorithm. The authors claim that the performance of their hybrid algorithm is comparable to a continuous-based approach by Gibbons et al. [79] but inferior to tabu search- and simulated annealing-based approaches. Fleurent et al. [65] used tabu search and other techniques as alternative mutation operators. The performance of their approach was satisfactory with respect to the quality of the obtained solutions but rather disappointing with respect to the computational effort. Marchiori [136] developed an approach that combines a genetic algorithm and a greedy heuristic, referred to as HGA. At each iteration of genetic algorithm, the greedy heuristic is applied to each member of the population to extract maximal cliques. That way, the genetic part of the approach provides exploration of the search space, while the greedy heuristic provides exploitation. Despite its simplicity, the approach has shown very good results [136, 137]. Zhang et al. [186] developed an approach called EA/G which uses similar techniques as HGA but also incorporates a guided mutation operator. The results of the experiments indicated that the algorithm

was superior to HGA. Singh and Gupta [162] designed a framework which consists of two phases: generation of cliques by a steady-state genetic algorithm and extending them to maximal cliques by a heuristic which combines a randomized sequential greedy approach and the exact algorithm of Carraghan and Pardalos [36]. The results of the experiments show that the algorithm outperformed three other evolutionary algorithms, of Balas and Niehaus [11], Marchiori [137], and Fenet and Solnon [61], but its performance is inferior to that of one of the most effective local search-based techniques (RLS) of Battiti and Protasi [14].

More recently, Guturu and Dantu [90] designed an approach that combines an impatient evolutionary algorithm and a probabilistic tabu search. Brunato and Battiti [25] presented a hybrid algorithm similar to EA/G by Zhang et al. in [186]. In the proposed algorithm, which they refer to as R-EVO, the new solutions are initialized according to an estimated distribution which is based on knowledge extracted from the previous generations of the population. The solutions are then evolved through a simplified version of RLS technique.

Despite such a significant effort in studying and designing different variations of evolutionary and genetic algorithms, these approaches are still considered to be far less effective for the maximum clique problem than the local search-based ones. Perhaps, their inability to compete with the simple local search techniques lies in the fact that there is no intuitive relationship between the crossover operator and a discovery of improved cliques, and no such a meaningful operator has been devised yet [182].

For the vertex coloring problem, the solutions are usually encoded either as permutations of the vertices and referred to as *order-based encoding* or as color partitions and referred to as *direct encoding*. The direct encoding can further be categorized into the *assignment based* and *partition based*. Under the former one, a solution is represented as an array, where each entry is associated with a certain vertex and contains an index of the class the corresponding vertex belongs to. The latter one, on the other hand, represents a solution as a partition of the vertices into the color classes. Even though more satisfactory results have been achieved using direct encoding techniques, the order-based encoding has a powerful advantage of always corresponding to a feasible solution. Hence, it has been somewhat exploited as well [48, 57, 145].

One of the first attempts to apply evolutionary algorithms for vertex coloring problem is by Davis [48]. The author used order-based encoding, and a solution was evaluated using a greedy sequential heuristic, which colors the vertices in the order defined by the permutation. The algorithm yielded unsatisfactory results, as was also shown in [66]. All of the later efforts of applying evolutionary algorithms to the vertex coloring problem involved the incorporation of other techniques, such as local search. The pioneers of this direction were Costa et al. [44] and Fleurent and Ferland [66], replacing the mutation operator by a simple descent method in the former and tabu search in the latter, respectively. In both papers the uniform crossover operator or its slightly enhanced modification [66] is used. The algorithms yielded slightly better results than their pure counterparts; however, the importance of designing better crossover operators was concluded in [66].

In later developments, the focus shifted to designing more advanced crossover operators tailored to the specifics of the problem of interest. More precisely, Dorne and Hao [52] used a specialized crossover based on the notion of the union of independent sets, which, with the combination of tabu search, led to a simple yet very powerful algorithm. Galinier and Hao [70] devised a new class of crossover mechanisms based on the partition of vertices into color classes rather than assignment of colors to vertices. The crossover operator called *greedy partition crossover* (GPX) transmits the subsets of color classes from parent solutions to the offspring solution. The power of this operator lies in its ability to retain some of each parent's structure in the offspring solution. The hybrid algorithm that uses this crossover operator and a tabu search was tested on large and difficult DIMACS instances and showed to be one of the best performing compared to other coloring algorithms [70]. A few years later, Glass and Prügel-Bennett [80] examined the hybrid algorithm of Galinier and Hao [70] by replacing the tabu search part of the algorithm with a steepest descent. The results of the experiments showed that the algorithm still remained powerful, which led to a conclusion that its success is due to the crossover operator. Hamiez and Hao [92] presented a *scatter search algorithm* devised for the vertex coloring problem. A scatter search is a subclass of evolutionary approaches, where replacements of the population are based not only on the improvement of the fitness function but also on the improvement of the population diversity; see the corresponding chapter in this handbook for more detail. The algorithm proposed in [92] was able to obtain results similar to the best-known approaches, but it was more expensive computationally.

More recently, Galinier et al. [72] proposed an *adaptive memory algorithm*, called AMACOL, where portions of solutions (color classes) are stored in a central memory and are then used to build new solutions. The algorithm uses a tabu search and yields results comparable to those obtained by an algorithm proposed in [70]. Malaguti et al. [133] designed a two-phase metaheuristic algorithm, called MMT, for the vertex coloring problem. The first phase is a combination of an evolutionary algorithm with a crossover similar to GPX from [70] and a tabu search with an impasse neighborhood, proposed in [142]. The second phase of MMT is a post-optimization procedure based on the set covering formulation of the problem. The algorithm yields promising results.

Ant Colony Optimization

Ant colony optimization, discussed in more detail in a separate chapter of this book, was originally introduced by Dorigo [51] and was inspired by the way the ants use pheromone to communicate with each other and search for better paths to a source of food. As they search for food, each ant leaves a pheromone trail. All subsequent ants will follow the path with stronger pheromone. Since the pheromone evaporates over time, it will last longer on better (shorter) paths. This way, these paths will be followed by more ants and will subsequently have more pheromone laid. Eventually all the ants will follow one single best path. The main idea of ant colony optimization algorithms is to mimic this process. Each ant is thought of as a constructive heuristic,

which builds a solution step by step using a greedy force and a *trail* information on the history of the search obtained from other ants.

First attempt to investigate ant colony capabilities to tackle the maximum clique problem was by Fenet and Solnon [61]. They introduced an algorithm, called Ant-Clique, that generates maximal cliques by repeatedly adding vertices to partial solutions, where each vertex is chosen according to the probability that depends on the level of pheromone. The pheromone deposition is proportional to the quality of previously constructed cliques. The performance of the algorithm was compared to that of [137] and showed that on average Ant-Clique is capable of reaching better solutions on the majority of tested instances. Youseff and Elliman [184] introduced an algorithm that combines the capabilities of ant colony optimization and local search techniques with prohibition rules. The algorithm was compared to the genetic local search of [137] and has shown to be somewhat competitive. Incorporation of local search techniques into the ant colony algorithm has also been addressed in [165] and has shown to improve the solution process.

One of the first ant colony algorithms for the vertex coloring problem was designed by Costa et al. [43]. Their algorithm, called ANTCOL, considers each ant as a constructive heuristic derived from DSATUR [23] and RLF [125]. Even though the algorithm was not superior of the best graph coloring heuristics of that time, its performance was still promising enough to encourage more research in this direction. Dowsland and Thompson further applied the same algorithm for examination scheduling [53] and later enhanced it by strengthening the construction phase and incorporating it with a tabu search improvement phase [54].

Hertz and Zuffrey [100] argued that an algorithm, where each ant has only a minor role, is still competitive with the other ant algorithms. Namely, in their proposed algorithm, each single ant does not build an entire solution but only colors a single vertex. A similar idea was explored by Bui et al. [27]. In their algorithm, instead of coloring the entire graph, each ant colors only a portion of the graph. The performance of the algorithm seemed to be rather encouraging. Plumetta et al. [155] proposed an ant local search algorithm, where each ant is considered as a local search, rather than a constructive heuristic. The algorithm showed to be competitive with other evolutionary methods available at that time.

Recently, Zufferey [188] conducted an analysis of importance of different ant roles, ranging from insignificant ones, used to make a minor decision, to more crucial ones, by performing a refined local search. It was shown experimentally that the ant algorithms are more efficient when ants have strong roles like local search procedures.

Heuristics Based on Continuous Formulations

A remarkable result by Motzkin and Straus [144], which provides an elegant connection between maximum cliques and a certain quadratic program, has been extensively explored to devise efficient heuristics for the maximum clique problem. Some of the early methods are discussed in detail in [22].

Given a graph $G = (V, E)$, let A_G be the adjacency matrix of G. Consider the following quadratic problem:

$$\begin{aligned} \max\ & g(x) = x^T A_G x, \\ \text{s.t. } & e^T x = 1, \\ & x \in R^{|V|}, \end{aligned}$$

where e is a unit vector.

Motzkin and Straus [144] showed that the size of a maximum clique in G is equal to:

$$\omega(G) = \frac{1}{1 - g(x^*)},$$

where x^* is a global maximizer of the above problem.

One drawback associated with this result is that a solution vector x^* does not always correspond to a maximum clique C, hence, making it hard to extract the clique's vertices. Bomze [21] proposed a regularization of the Motzkin-Straus formulation by adding $\frac{1}{2} x^T I x$ to the objective, where I is the corresponding identity matrix. He has shown that every local maximum of the modified formulation corresponds to a maximal clique in the graph as follows: C is a maximal clique of G if and only if x^*, such that $x_i^* = 1/|C|$ if $x_i \in C$ and $x_i^* = 0$ otherwise, is a local maximum of the regularized Motzkin-Straus formulation.

One of the early approaches based on the Motzkin-Straus result was an iterative clique retrieval procedure [149], which turned out to be of very high computational cost and was only able to solve instances on less than 75 vertices. Gibbons et al. [79] proposed to modify the Motzkin-Straus formulation so that it becomes a problem of optimizing a quadratic function over a sphere. Even though such problem is polynomially solvable, it, however, yields approximate solutions that need to be rounded. Similar approaches, with a few advances, can be found in [30,31]. Simple heuristics based on formulations of the maximum independent set problem as maximization of polynomial functions over a unit hypercube were studied in [3].

Gruzdeva [89] proposed to add a non-convex quadratic constraint represented by the difference of two convex functions (DC contraints) [105] to a continuous formulation of the maximum clique problem. The author showed that the proposed approach is competitive with other methods based on continuous formulations.

Massaro et al. [138] transformed the Motzkin-Straus formulation of the maximum clique problem into its corresponding linear complementary problem (LCP) [45]. To deal with the inherent degeneracy of the derived LCP, the authors designed a variation of a classical Lemke's method [126] with an effective "look-ahead" pivot rule.

Recently, Butenko et al. [34] proposed variable objective search, a metaheuristic framework that performs a local search with respect to different alternative formulations of the same combinatorial optimization problems. To test their method, authors considered the maximum independent set problem and its two equivalent non-convex programs discussed in [3].

Another notable result – the "sandwich theorem" [122] – has inspired appearance of several heuristic algorithms for both the maximum clique (independent set) [28, 55] and vertex coloring problems [56, 84, 117]. For a given graph G, the sandwich theorem states that the following relationship holds:

$$\omega(G) \leq \theta(\bar{G}) \leq \chi(G),$$

where $\theta(\bar{G})$ is the Lovász theta, which can be computed in polynomial time [129].

In [28], the authors study rank-one and rank-two formulations of the semidefinite programming (SDP) formulation of the Lovász theta number. Based on the obtained further continuous formulations, they designed heuristics for the maximum independent set problem. As for the vertex coloring problem, the proposed solution algorithms also used a semidefinite programming (SDP) formulation of the Lovász theta and were capable of obtaining near-optimal solutions for problems of medium sizes [56].

Other Heuristics

Many other interesting heuristic approaches have been devised to tackle the maximum clique and the vertex coloring problems. Balas and Niehaus [10] developed a heuristic, called CLIQMERGE, that generates large cliques by repeatedly finding a maximum clique in the subgraph induced by the union of two cliques. Such procedure is performed by finding the bipartite matching of the complement subgraph. A heuristic for the maximum independent set problem based on the operation of *edge projection*, which is a specialization of Lovász and Plummer's clique projection [130], has been designed in [135]. Goldberg and Rivenburgh [83] used a *restricted backtracking* for detecting cliques in graphs. DNA computing was applied to the maximum clique problem in [147, 185]. Neural network approach has been very popular in tackling the maximum clique problem since the mid-1980s. A detailed discussion on neural network-based methods applied to the maximum clique problem developed before 1999 is presented in [22]. The interested readers are also referred to [86, 106, 107, 110].

For the vertex coloring problem, the neural network approach has also been applied in [16, 17, 77, 108, 154, 168, 169]. An algorithm which inherits ideas from quantum mechanics was proposed in [171].

Computational Results

In this section we summarize best computational results achieved by heuristic algorithms on selected hard benchmark instances for the problems of interest. We disregard running times and focus on the quality of the reported solutions. More

specifically, we only report the best found solutions for the instances with no known optima and list the methods that were able to attain these solutions.

The results for the maximum clique instances are presented in Table 1. First three columns contain information about the graph instance (name, number of vertices $|V|$, and edges $|E|$). In the next columns, we report the size of the largest detected clique (ω_{lb}) and list references of the methods that were able to attain such solution. Note that the first two graph instances in Table 1 are from the Second DIMACS Implementation Challenge [49], the next three instances are from the Tenth DIMACS Implementation Challenge [50], and the rest of them are from the so-called CODE family [164], which is a collection of challenging graph instances arising from the coding theory.

The computational results related to the coloring problem are contained in Table 2. Here, the first three columns are the same as in Table 1. The fourth and fifth columns contain the size of the lower (χ_{lb}) and upper (χ_{ub}) bounds on the chromatic number. The majority of the results related to the lower bound size is by [98]. The last column in Table 2 lists the references to the algorithms which have achieved χ_{ub}. The first 27 instances presented in this table are from the Second DIMACS Implementation Challenge [49]; the next 12 are from Stanford Large Network Dataset Collection [127], referred to as the SNAP; and the last 8 are from the Tenth DIMACS Implementation Challenge [50].

More detailed discussions on computational performance of various heuristics can be found in some of the recent surveys [73, 132, 182].

Table 1 Approximate solutions to the maximum clique problem

Instance	$\|V\|$	$\|E\|$	ω_{lb}	ω_{lb} obtained by
Keller6	3,361	4,619,898	59	[5, 14, 15, 90, 95, 112, 158–160, 180, 183]
Hamming10-4	1,024	434,176	40	[14, 15, 90, 95, 112, 158, 159, 180, 183]
c500.9	500	112,332	57	[14, 15, 90, 95, 112, 158–160, 180, 183]
c1000.9	1,000	450,079	68	[5, 14, 15, 90, 95, 112, 158–160, 180, 183]
c2000.9	2,000	1,799,532	80	[15, 112, 180, 183]
1dc.1024	1,024	24,063	94	[5, 33, 112]
1dc.2048	2,048	58,367	172	[5, 33, 112]
1et.1024	1,024	9,600	171	[5, 33, 112]
1et.2048	2,048	22,528	316	[5, 33, 112]
1tc.1024	1,024	7,936	196	[5, 33, 112]
1tc.2048	2,048	18,944	352	[5, 33, 112]
1zc.1024	1,024	33,280	112	[5, 33, 112]
1zc.2048	2,048	78,848	198	[5, 33, 112]
1zc.4096	4,096	184,320	379	[5, 33, 112]
2dc.1024	1,024	169,162	16	[5, 33, 112]
2dc.2048	2,048	504,451	24	[5, 33, 112]

Table 2 Approximate solutions to the vertex coloring problem

Instance	$\|V\|$	$\|E\|$	χ_{lb}	χ_{ub}	χ_{ub} obtained by
DSJC250.5	250	15,668	26	28	[40, 69, 70, 72, 84, 131, 133, 142, 157, 171]
DSJC500.1	500	12,458	9	12	[18, 40, 52, 69, 72, 84, 101, 131, 133, 142, 155–157, 171]
DSJC500.5	500	62,624	43	48	[18, 52, 70, 72, 101, 131, 133, 142, 155–157, 170, 171]
DSJC500.9	500	1,124,367	123	126	[40, 40, 52, 72, 101, 131, 155–157, 170, 171]
DSJC1000.1	1,000	49,629	10	20	[18, 52, 70, 72, 96, 101, 131, 133, 155–157, 170, 171, 179]
DSJC1000.5	1,000	249,826	73	83	[52, 70, 96, 131, 133, 142, 156, 170, 171, 179]
DSJC1000.9	1,000	449,449	216	222	[96, 170, 171, 179]
r1000.1c	1000	485090	96	98	[18, 69, 96, 131, 133, 142, 156, 157, 171]
latin_square_10	900	307,350	90	97	[170]
1-Insertions_5	202	1,227	4	6	[72, 134]
1-Insertions_6	607	6,337	4	7	[72, 134]
2-Insertions_4	149	541	4	5	[40, 72, 134]
2-Insertions_5	597	3,936	3	6	[72, 134]
3-Insertions_4	281	1,046	3	4	[40]
3-Insertions_5	1,406	9,695	3	6	[40, 134]
4-Insertions_4	475	1,795	3	5	[72, 134]
4-FullIns_5	4,146	77,305	7	9	[40, 134]
5-FullIns_4	1,085	11,395	8	9	[134]
wap01	2,368	110,871	41	42	[40, 96]
wap02	2,464	111,742	40	41	[96]
wap03	4,730	286,722	40	44	[96]
wap04	5,231	294,902	40	42	[96]
wap07	1,809	103,368	40	41	[96]
wap08	1,870	104,176	40	42	[96, 134]
c2000.5	2,000	999,836	99	145	[96]
c2000.9	2,000	1,799,532	80	408	[96]
c4000.5	4,000	4,000,268	107	259	[96]
Wiki-Vote	7,115	100,762	19	24	[172]
p2p-Gnutella04	10,876	39,994	4	6	[172]
p2p-Gnutella25	22,687	54,705	4	6	[172]
p2p-Gnutella24	26,518	65,369	4	6	[172]
Cit-HepTh	27,770	352,285	23	25	[172]
Cit-HepPh	34,546	420,877	19	21	[172]
p2p-Gnutella30	36,682	88,328	4	6	[172]
p2p-Gnutella31	62,586	147,892	4	6	[172]
soc-Epinions1	75,879	405,740	25	30	[172]
Email-EuAll	265,214	364,481	18	20	[172]
WikiTalk	2,394,385	4,659,565	31	51	[172]
cit-Patents	3,774,768	16,518,947	11	12	[172]

(*continued*)

Table 2 (continued)

Instance	$\|V\|$	$\|E\|$	χ_{lb}	χ_{ub}	χ_{ub} obtained by
kron_g500-simple-logn16	65,536	2,456,071	136	155	[172]
G_n_pin_pout	100,000	501,198	4	8	[172]
smallworld	100,000	499,998	6	8	[172]
wave	156,317	1,059,331	6	9	[172]
audikw1	943,695	38,354,076	36	44	[172]
ldoor	952,203	22,785,136	21	35	[172]
333SP	3,712,815	11,108,633	4	5	[172]
cage15	5,154,859	47,022,346	6	13	[172]

Conclusion

In this chapter, we discussed various heuristics for approximating the maximum clique and the vertex coloring problems that were developed in the past 20–30 years. The local search-based metaheuristics proved to be more effective than the population-based ones. Nevertheless, the latter ones still have gained a lot of attention among the researchers. Indeed, when combined with effective local search techniques, they can actually yield competitive results; moreover, they can be easily parallelized. Heuristics based on continuous optimization represent an interesting and promising direction for future research.

Cross-References

▸ GRASP
▸ Tabu Search
▸ Variable Neighborhood Search

Acknowledgments We would like to thank the anonymous reviewers and the book editors, whose suggestions helped to improve the manuscript. Partial support of NSF grant CMMI-1538493 is also gratefully acknowledged.

References

1. Aarts E, Korst J (1988) Simulated annealing and Boltzmann machines. Wiley, Chichester
2. Abello J, Pardalos PM, Resende MGC (1999) On maximum clique problems in very large graphs. In: Abello J, Vitter J (eds) External memory algorithms and visualization. American Mathematical Society, Boston, pp 119–130
3. Abello J, Butenko S, Pardalos PM, Resende MGC (2001) Finding independent sets in a graph using continuous multivariable polynomial formulations. J Glob Optim 21(2):111–137
4. Aggarwal CC, Orlin JB, Tai RP (1997) Optimized crossover for the independent set problem. Oper Res 45(2):226–234

5. Andrade DV, Resende MGC, Werneck RF (2012) Fast local search for the maximum independent set problem. J Heuristics 18(4):525–547
6. Arora S, Safra S (1998) Probabilistic checking of proofs: a new characterization of NP. J ACM (JACM) 45(1):70–122
7. Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1998) Proof verification and the hardness of approximation problems. J ACM (JACM) 45(3):501–555
8. Avanthay C, Hertz A, Zufferey N (2003) A variable neighborhood search for graph coloring. Eur J Oper Res 151(2):379–388
9. Back T, Khuri S (1994) An evolutionary heuristic for the maximum independent set problem. In: Proceedings of the first IEEE conference on evolutionary computation, pp 531–535
10. Balas E, Niehaus W (1996) Finding large cliques in arbitrary graphs by bipartite matching. DIMACS Ser Discrete Math Theor Comput Sci 26:29–52
11. Balas E, Niehaus W (1998) Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. J Heuristics 4(2):107–122
12. Balasundaram B, Butenko S (2006) Graph domination, coloring and cliques in telecommunications. In: Handbook of optimization in telecommunications. Springer, Berlin, pp 865–890
13. Battiti R, Mascia F (2010) Reactive and dynamic local search for max-clique: engineering effective building blocks. Comput Oper Res 37(3):534–542
14. Battiti R, Protasi M (2001) Reactive local search for the maximum clique problem. Algorithmica 29(4):610–637
15. Benlic U, Hao JK (2013) Breakout local search for maximum clique problems. Comput Oper Res 40(1):192–206
16. Berger MO (1994) k-coloring vertices using a neural network with convergence to valid solutions. In: Proceedings of IEEE international conference on neural networks, vol 7, pp 4514–4517
17. Blas AD, Jagota A, Hughey R (2002) Energy function-based approaches to graph coloring. IEEE Trans Neural Netw 13(1):81–91
18. Blöchliger I, Zufferey N (2008) A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput Oper Res 35(3):960–975
19. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv (CSUR) 35(3):268–308
20. Bollobás B, Thomason A (1985) Random graphs of small order. North-Holland Math Stud 118:47–97
21. Bomze IM (1997) Evolution towards the maximum clique. J Glob Optim 10:143–164
22. Bomze IM, Budinich M, Pardalos PM, Pelillo M (1999) The maximum clique problem. In: Handbook of combinatorial optimization. Springer, Boston, pp 1–74
23. Brélaz D (1979) New methods to color the vertices of a graph. Commun ACM 22(4):251–256
24. Brooks RL (1941) On colouring the nodes of a network. In: Mathematical proceedings of the Cambridge philosophical society. Cambridge University Press, vol 37, pp 194–197
25. Brunato M, Battiti R (2011) R-EVO: a reactive evolutionary algorithm for the maximum clique problem. IEEE Trans Evol Comput 15(6):770–782
26. Bui TN, Eppley PH (1995) A hybrid genetic algorithm for the maximum clique problem. In: Proceedings of the 6th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc., pp 478–484
27. Bui TN, Nguyen TH, Patel CM, Phan KAT (2008) An ant-based algorithm for coloring graphs. Discrete Appl Math 156(2):190–200
28. Burer S, Monteiro RD, Zhang Y (2002) Maximum stable set formulations and heuristics based on continuous optimization. Math Program 94(1):137–166
29. Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. Int Ser Oper Res Manag Sci 57:457–474
30. Busygin S (2006) A new trust region technique for the maximum weight clique problem. Discrete Appl Math 154(15):2080–2096
31. Busygin S, Butenko S, Pardalos PM (2002) A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. J Comb Optim 6(3):287–297

32. Butenko S, Wilhelm WE (2006) Clique-detection models in computational biochemistry and genomics. Eur J Oper Res 173(1):1–17
33. Butenko S, Pardalos PM, Sergienko IV, Shylo V, Stetsyuk P (2009) Estimating the size of correcting codes using extremal graph problems. In: Pearce C, Hunt E (eds) Optimization: structure and applications. Springer, New York, pp 227–243
34. Butenko S, Yezerska O, Balasundaram B (2013) Variable objective search. J Heuristics 19(4):697–709
35. Caprara A, Kroon L, Monaci M, Peeters M, Toth P (2007) Passenger railway optimization. Handb Oper Res Manag Sci 14:129–187
36. Carraghan R, Pardalos PM (1990) An exact algorithm for the maximum clique problem. Oper Res Lett 9(6):375–382
37. Carter R, Park K (1993) How good are genetic algorithms at finding large cliques: an experimental study. Technical report, Computer Science Department, Boston University
38. Chaitin GJ, Auslander MA, Chandra AK, Cocke J, Hopkins ME, Markstein PW (1981) Register allocation via coloring. Comput Lang 6(1):47–57
39. Chams M, Hertz A, de Werra D (1987) Some experiments with simulated annealing for coloring graphs. Eur J Oper Res 32(2):260–266
40. Chiarandini M, Stützle T et al (2002) An application of iterated local search to graph coloring problem. In: Proceedings of the computational symposium on graph coloring and its generalizations, pp 112–125
41. Chiarandini M, Dumitrescu I, Stützle T (2007) Stochastic local search algorithms for the graph colouring problem. In: Handbook of approximation algorithms and metaheuristics. Chapman & Hall/CRC, Boca Raton, pp 63-1
42. Chow FC, Hennessy JL (1990) The priority-based coloring approach to register allocation. ACM Trans Program Lang Syst (TOPLAS) 12(4):501–536
43. Costa D, Hertz A (1997) Ants can colour graphs. J Oper Res Soc 48(3):295–305
44. Costa D, Hertz A, Dubuis C (1995) Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. J Heuristics 1(1):105–128
45. Cottle RW, Pang JS, Stone RE (1992) The linear complementarity problem. SIAM, Philadelphia
46. Culberson JC (1992) Iterated greedy graph coloring and the difficulty landscape. Technical report. TK 92-07, Department of Computing Science, University of Alberta
47. Culberson JC, Luo F (1996) Exploring the k-colorable landscape with iterated greedy. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 245–284
48. Davis L (1991) Order-based genetic algorithms and the graph coloring problem. In: Handbook of genetic algorithms. Van Nostrand Reinhold, New York, pp 72–90
49. DIMACS (1993) NP hard problems: maximum clique, graph coloring, and satisfiability. The second DIMACS implementation challenge. http://dimacs.rutgers.edu/Challenges/. Accessed 10 Jan 2018
50. DIMACS (2012) Algorithm implementation challenge: graph partitioning and graph clustering. The tenth DIMACS implementation challenge. http://dimacs.rutgers.edu/Challenges/. Accessed 10 Jan 2018
51. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B Cybern 26(1):29–41
52. Dorne R, Hao JK (1998) A new genetic local search algorithm for graph coloring. In: Parallel problem solving from nature. Springer, Berlin/Heidelberg, pp 745–754
53. Dowsland KA, Thompson JM (2005) Ant colony optimization for the examination scheduling problem. J Oper Res Soc 56(4):426–438
54. Dowsland KA, Thompson JM (2008) An improved ant colony optimisation heuristic for graph colouring. Discrete Appl Math 156(3):313–324
55. Dukanovic I, Rendl F (2007) Semidefinite programming relaxations for graph coloring and maximal clique problems. Math Program 109(2–3):345–365

56. Dukanovic I, Rendl F (2008) A semidefinite programming-based heuristic for graph coloring. Discrete Appl Math 156(2):180–189
57. Eiben ÁE, Van Der Hauw JK, van Hemert JI (1998) Graph coloring with adaptive evolutionary algorithms. J Heuristics 4(1):25–46
58. Erdös P (1970) On the graph theorem of Turán. Mat Lapok 21(249–251):10
59. Feige U, Kilian J (1996) Zero knowledge and the chromatic number. In: Proceedings of eleventh annual IEEE conference on computational complexity, pp 278–287
60. Feige U, Kilian J (1998) Zero knowledge and the chromatic number. J Comput Syst Sci 57:187–199
61. Fenet S, Solnon C (2003) Searching for maximum cliques with ant colony optimization. In: Applications of evolutionary computing. Springer, Berlin, pp 236–245
62. Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71
63. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
64. Feo TA, Resende MGC, Smith SH (1994) A greedy randomized adaptive search procedure for maximum independent set. Oper Res 42(5):860–878
65. Ferland J, Fleurent C (1996) Object-oriented implementation of heuristic search methods for graph coloring, maximum clique and satisfiability. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 619–652
66. Fleurent C, Ferland JA (1996) Genetic and hybrid algorithms for graph coloring. Ann Oper Res 63(3):437–461
67. Foster JA, Soule T (1995) Using genetic algorithms to find maximum cliques. Technical report. LAL95-12, Department of Computer Science, University of Idaho
68. Friden C, Hertz A, de Werra D (1989) STABULUS: a technique for finding stable sets in large graphs with tabu search. Computing 42:35–44
69. Funabiki N, Higashino T (2000) A minimal-state processing search algorithm for graph coloring problems. IEICE Trans Fundam Electron Commun Comput Sci 83(7):1420–1430
70. Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. J Comb Optim 3(4):379–397
71. Galinier P, Hertz A (2006) A survey of local search methods for graph coloring. Comput Oper Res 33(9):2547–2562
72. Galinier P, Hertz A, Zufferey N (2008) An adaptive memory algorithm for the k-coloring problem. Discrete Appl Math 156(2):267–279
73. Galinier P, Hamiez JP, Hao JK, Porumbel D (2013) Recent advances in graph vertex coloring. In: Handbook of optimization. Springer, Berlin/Heidelberg, pp 505–528
74. Gamst A (1986) Some lower bounds for a class of frequency assignment problems. IEEE Trans Veh Technol 35(1):8–14
75. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Company, New York
76. Garey MR, Johnson DS, So H (1976) An application of graph coloring to printed circuit testing. IEEE Trans Circuits Syst 23(10):591–599
77. Gassen DW, Carothers JD (1993) Graph color minimization using neural networks. In: Proceedings of IEEE international joint conference on neural networks, vol 2, pp 1541–1544
78. Gendreau M, Soriano P, Salvail L (1993) Solving the maximum clique problem using a tabu search approach. Ann Oper Res 41(4):385–403
79. Gibbons LE, Hearn DW, Pardalos PM (1996) A continuous based heuristic for the maximum clique problem. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 103–124
80. Glass CA, Prügel-Bennett A (2003) Genetic algorithm for graph coloring: exploration of Galinier and Hao's algorithm. J Comb Optim 7(3):229–236

81. Glover F (1989) Tabu search. Part I. ORSA J Comput 1(3):190–206
82. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
83. Goldberg MK, Rivenburgh RD (1996) Constructing cliques using restricted backtracking. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 285–307
84. Govorčin J, Gvozdenović N, Povh J (2013) New heuristics for the vertex coloring problem based on semidefinite programming. Cent Eur J Oper Res 21(1):13–25
85. Grable DA, Panconesi A (1998) Fast distributed algorithms for Brooks-Vizing colourings. In: Proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, pp 473–480
86. Grossman T (1996) Applying the inn model to the maximum clique problem. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 125–146
87. Grosso A, Locatelli M, Della Croce F (2004) Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. J Heuristics 10(2):135–152
88. Grosso A, Locatelli M, Pullan W (2008) Simple ingredients leading to very efficient heuristics for the maximum clique problem. J Heuristics 14(6):587–612
89. Gruzdeva TV (2013) On a continuous approach for the maximum weighted clique problem. J Glob Optim 56(3):971–981
90. Guturu P, Dantu R (2008) An impatient evolutionary algorithm with probabilistic tabu search for unified solution of some np-hard problems in graph and set theory via clique finding. IEEE Trans Syst Man Cybern B Cybern 38(3):645–666
91. Hajnal P, Szemerédi E (1990) Brooks coloring in parallel. SIAM J Discret Math 3(1):74–80
92. Hamiez JP, Hao JK (2002) Scatter search for graph coloring. In: Artificial evolution. Springer, Berlin/Heidelberg pp 168–179
93. Hansen P, Jaumard B (1990) Algorithms for the maximum satisfiability problem. Computing 44(4):279–303
94. Hansen P, Mladenović N (1999) An introduction to variable neighborhood search. In: Voss S et al (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Springer, Boston, pp 433–458
95. Hansen P, Mladenović N, Urošević D (2004) Variable neighborhood search for the maximum clique. Discret Appl Math 145(1):117–125
96. Hao JK, Wu Q (2012) Improving the extraction and expansion method for large graph coloring. Discret Appl Math 160(16):2397–2407
97. Håstad J (1999) Clique is hard to approximate within $n^{1-\epsilon}$. Acta Math 182:105–142
98. Held S, Cook W, Sewell EC (2012) Maximum-weight stable sets and safe lower bounds for graph coloring. Math Program Comput 4(4):363–381
99. Hertz A, de Werra D (1987) Using tabu search techniques for graph coloring. Computing 39(4):345–351
100. Hertz A, Zufferey N (2006) A new ant algorithm for graph coloring. In: Workshop on nature inspired cooperative strategies for optimization, NICSO, pp 51–60
101. Hertz A, Plumettaz M, Zufferey N (2008) Variable space search for graph coloring. Discret Appl Math 156(13):2551–2560
102. Hifi M (1997) A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. J Oper Res Soc 48(6):612–622
103. Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, Cambridge
104. Homer S, Peinado M (1996) Experiments with polynomial-time clique approximation algorithms on very large graphs. DIMACS Ser Discret Math Theor Comput Sci 26: 147–168
105. Horst R, Thoai NV (1999) Dc programming: overview. J Optim Theory Appl 103(1):1–43
106. Jagota A (1992) Efficiently approximating MAX-CLIQUE in a Hopfield-style network. In: International joint conference on neural networks, vol 2, pp 248–253

107. Jagota A (1995) Approximating maximum clique with a Hopfield network. IEEE Trans Neural Netw 6(3):724–735
108. Jagota A (1996) An adaptive, multiple restarts neural network algorithm for graph coloring. Eur J Oper Res 93(2):257–270
109. Jagota A, Sanchis LA (2001) Adaptive, restart, randomized greedy heuristics for maximum clique. J Heuristics 7(6):565–585
110. Jagota A, Sanchis L, Ganesan R (1996) Approximately solving maximum clique using neural networks and related heuristics. DIMACS Ser Discret Math Theor Comput Sci 26: 169–204
111. Jerrum M (1992) Large cliques elude the metropolis process. Random Struct Algorithm 3(4):347–359
112. Jin Y, Hao JK (2015) General swap-based multiple neighborhood tabu search for the maximum independent set problem. Eng Appl Artif Intell 37:20–33
113. Johnson DS, Trick MA (eds) (1996) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence
114. Johnson DS, Aragon CR, McGeoch LA, Schevon C (1991) Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. Oper Res 39(3):378–406
115. Kahruman-Anderoglu S, Buchanan A, Butenko S, Prokopyev O (2016) On provably best construction heuristics for hard combinatorial optimization problems. Networks 67:238–245. https://doi.org/10.1002/net.21620
116. Karchmer M, Naor J (1988) A fast parallel algorithm to color a graph with Δ colors. J Algorithm 9(1):83–91
117. Karger D, Motwani R, Sudan M (1998) Approximate graph coloring by semidefinite programming. J ACM (JACM) 45(2):246–265
118. Karloff HJ (1989) An NC algorithm for Brooks' theorem. Theor Comput Sci 68(1):89–103
119. Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum, New York, pp 85–103
120. Katayama K, Hamamoto A, Narihisa H (2005) An effective local search for the maximum clique problem. Inf Process Lett 95(5):503–511
121. Kirkpatrick S, Vecchi M et al (1983) Optimization by simulated annealing. Science 220(4598):671–680
122. Knuth DE (1994) The sandwich theorem. Electron J Comb 1:1–48. http://www.combinatorics.org/Volume_1/Abstracts/v1i1a1.html. Accessed 10 Jan 2018
123. Kopf R, Ruhe G (1987) A computational study of the weighted independent set problem for general graphs. Found Control Eng 12(4):167–180
124. Laguna M, Martí R (2001) A grasp for coloring sparse graphs. Computat Optim Appl 19(2):165–178
125. Leighton FT (1979) A graph coloring algorithm for large scheduling problems. J Res Natl Bur Stand 84(6):489–506
126. Lemke CE (1965) Bimatrix equilibrium points and mathematical programming. Manag Sci 11(7):681–689
127. Leskovec J, Krevl A (2014) SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data. Accessed 10 Jan 2018
128. Lovász L (1975) Three short proofs in graph theory. J Comb Theory Ser B 19(3):269–271
129. Lovász L (1979) On the shannon capacity of a graph. IEEE Trans Inf Theory 25(1):1–7
130. Lovász L, Plummer MD (2009) Matching theory. American Mathematical Society, Providence
131. Lü Z, Hao JK (2010) A memetic algorithm for graph coloring. Eur J Oper Res 203(1): 241–250
132. Malaguti E, Toth P (2010) A survey on vertex coloring problems. Int Trans Oper Res 17(1): 1–34
133. Malaguti E, Monaci M, Toth P (2008) A metaheuristic approach for the vertex coloring problem. INFORMS J Comput 20(2):302–316

134. Malaguti E, Monaci M, Toth P (2011) An exact approach for the vertex coloring problem. Discret Optim 8(2):174–190
135. Mannino C, Sassano A (1996) Edge projection and the maximum cardinality stable set problem. DIMACS Ser Discret Math Theor Comput Sci 26:205–219
136. Marchiori E (1998) A simple heuristic based genetic algorithm for the maximum clique problem. In: Proceedings of the ACM symposium on applied computing, vol 27, pp 366–373
137. Marchiori E (2002) Genetic, iterated and multistart local search for the maximum clique problem. In: Applications of evolutionary computing. Springer, Berlin/Heidelberg, pp 112–121
138. Massaro A, Pelillo M, Bomze IM (2002) A complementary pivoting approach to the maximum weight clique problem. SIAM J Optim 12(4):928–948
139. Matula DW, Marble G, Isaacson JD (1972) Graph coloring algorithms. In: Read R (ed) Graph theory and computing. Academic, New York, pp 109–122
140. Mehta NK (1981) The application of a graph coloring method to an examination scheduling problem. Interfaces 11(5):57–65
141. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
142. Morgenstern C (1996) Distributed coloration neighborhood search. Discret Math Theor Comput Sci 26:335–358
143. Morgenstern CA, Shapiro HD (1986) Chromatic number approximation using simulated annealing. Technical report, Department of Computer Science, University of New Mexico
144. Motzkin TS, Straus EG (1965) Maxima for graphs and a new proof of a theorem of turán. Canad J Math 17(4):533–540
145. Mumford CL (2006) New order-based crossovers for the graph coloring problem. In: Parallel problem solving from nature. Springer, Berlin, pp 880–889
146. Murthy AS, Parthasarathy G, Sastry V (1994) Clique finding – a genetic approach. In: Proceedings of the first IEEE conference on evolutionary computation, pp 18–21
147. Ouyang Q, Kaplan PD, Liu S, Libchaber A (1997) Dna solution of the maximal clique problem. Science 278(5337):446–449
148. Paquete L, Stützle T (2002) An experimental investigation of iterated local search for coloring graphs. In: Applications of evolutionary computing. Springer, Berlin/Heidelberg, pp 122–131
149. Pardalos PM, Phillips A (1990) A global optimization approach for solving the maximum clique problem. Int J Comput Math 33(3–4):209–216
150. Pardalos PM, Xue J (1994) The maximum clique problem. J Glob Optim 4(3):301–328
151. Pardalos PM, Mavridou T, Xue J (1999) The graph coloring problem: a bibliographic survey. In: Handbook of combinatorial optimization. Springer, Boston, pp 1077–1141
152. Park K, Carter B (1995) On the effectiveness of genetic search in combinatorial optimization. In: Proceedings of the ACM symposium on applied computing, pp 329–336
153. Pattillo J, Butenko S (2011) Clique, independent set, and graph coloring. In: Encyclopedia of operations research and management science. Wiley, Hoboken, pp 3150–3163
154. Philipsen W, Stok L (1991) Graph coloring using neural networks. In: IEEE international symposium on circuits and systems, pp 1597–1600
155. Plumettaz M, Schindl D, Zufferey N (2010) Ant local search and its efficient adaptation to graph colouring. J Oper Res Soc 61(5):819–826
156. Porumbel DC, Hao JK, Kuntz P (2010) An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Comput Oper Res 37(10):1822–1832
157. Porumbel DC, Hao JK, Kuntz P (2010) A search space "cartography" for guiding graph coloring heuristics. Comput Oper Res 37(4):769–778
158. Pullan W (2006) Phased local search for the maximum clique problem. J Comb Optim 12(3):303–323
159. Pullan W, Hoos HH (2006) Dynamic local search for the maximum clique problem. J Artif Intell Res 25:159–185

160. Pullan W, Mascia F, Brunato M (2011) Cooperating local search for the maximum clique problem. J Heuristics 17(2):181–199
161. Resende MGC, Feo TA, Smith SH (1998) Algorithm 787: fortran subroutines for approximate solution of maximum independent set problems using grasp. ACM Trans Math Softw (TOMS) 24(4):386–394
162. Singh A, Gupta AK (2006) A hybrid heuristic for the maximum clique problem. J Heuristics 12(1–2):5–22
163. Skulrattanakulchai S (2006) Δ-list vertex coloring in linear time. Inf Process Lett 98(3): 101–106
164. Sloane N (2000) Challenge problems: independent sets in graphs. https://oeis.org/A265032/a265032.html. Accessed 10 Jan 2018
165. Solnon C, Fenet S (2006) A study of ACO capabilities for solving the maximum clique problem. J Heuristics 12(3):155–180
166. Soriano P, Gendreau M (1996) Diversification strategies in tabu search algorithms for the maximum clique problem. Ann Oper Res 63(2):189–207
167. Soriano P, Gendreau M (1996) Tabu search algorithms for the maximum clique problem. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. American Mathematical Society, Providence, pp 221–242
168. Takefuji Y, Lee KC (1991) Artificial neural networks for four-coloring map problems and k-colorability problems. IEEE Trans Circuits Syst 38(3):326–333
169. Talaván PM, Yáñez J (2008) The graph coloring problem: a neuronal network approach. Eur J Oper Res 191(1):100–111
170. Titiloye O, Crispin A (2011) Graph coloring with a distributed hybrid quantum annealing algorithm. In: Agent and multi-agent systems: technologies and applications. Springer, Berlin/Heidelberg, pp 553–562
171. Titiloye O, Crispin A (2011) Quantum annealing of the graph coloring problem. Discret Optim 8(2):376–384
172. Verma A, Buchanan A, Butenko S (2015) Solving the maximum clique and vertex coloring problems on very large sparse networks. INFORMS J Comput 27(1):164–177
173. Welsh DJ, Powell MB (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. Comput J 10(1):85–86
174. de Werra D (1985) An introduction to timetabling. Eur J Oper Res 19(2):151–162
175. de Werra D (1990) Heuristics for graph coloring. In: Computational graph theory. Springer, Berlin, pp 191–208
176. de Werra D, Gay Y (1994) Chromatic scheduling and frequency assignment. Discret Appl Math 49(1):165–174
177. Woo TK, Su SY, Newman-Wolfe R (1991) Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. IEEE Trans Commun 39(12):1794–1801
178. Wood D (1969) A technique for colouring a graph applicable to large scale timetabling problems. Comput J 12(4):317–319
179. Wu Q, Hao JK (2012) Coloring large graphs based on independent set extraction. Comput Oper Res 39(2):283–290
180. Wu Q, Hao JK (2013) An adaptive multistart tabu search approach to solve the maximum clique problem. J Comb Optim 26(1):86–108
181. Wu Q, Hao JK (2013) An extraction and expansion approach for graph coloring. Asia Pac J Oper Res 30(05):1350018
182. Wu Q, Hao JK (2015) A review on algorithms for maximum clique problems. Eur J Oper Res 242(3):693–709
183. Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196:611–634
184. Youssef SM, Elliman DG (2004) Reactive prohibition-based ant colony optimization (rpaco): a new parallel architecture for constrained clique sub-graphs. In: 16th IEEE international conference on tools with artificial intelligence, pp 63–70

185. Zhang BT, Shin SY (1999) Code optimization for dna computing of maximal cliques. In: Advances in soft computing. Springer, Heidelberg, pp 588–599
186. Zhang Q, Sun J, Tsang E (2005) An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Trans Evol Comput 9(2):192–200
187. Zuckerman D (2007) Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput 3:103–128
188. Zufferey N (2012) Optimization by ant algorithms: possible roles for an individual ant. Optim Lett 6(5):963–973

The Multi-plant Lot Sizing Problem with Multiple Periods and Items

45

Mariá C. V. Nascimento, Horacio H. Yanasse, and Desiree M. Carvalho

Contents

Introduction . . . 1292
The Lot Sizing Problem . . . 1292
Demands . . . 1293
Planning Horizon . . . 1293
Items . . . 1294
Levels . . . 1294
Plants . . . 1294
Machines . . . 1294
Setup Structure . . . 1294
Inventory and Backlogging . . . 1295
Network Flow-Based Formulation . . . 1299
Computational Experiments . . . 1301
Setup of the Experiments . . . 1302
Results of the Experiments . . . 1303
Conclusions . . . 1304
Cross-References . . . 1305
References . . . 1305

Abstract

The lot sizing problem consists in determining lot sizes and their scheduling to meet the required demands. This chapter focuses on a multi-plant lot sizing problem with planning for multiple items, each plant with their own demands and multiple periods. In spite of not been widely investigated, their theoretical and practical importance are supported by applications from private and public sectors. This chapter pays special attention to the multi-plant uncapacitated lot

M. C. V. Nascimento (✉) · H. H. Yanasse · D. M. Carvalho
Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo – UNIFESP, São Paulo, Brazil
e-mail: mcv.nascimento@unifesp.br; horacio.yanasse@unifesp.br; dmcarvalho@unifesp.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_41

sizing problem (MPULSP). In particular, a novel network flow formulation is introduced, and some computational experiments were carried out to assess the performance of commercial solvers in solving a number of large instance problems. Moreover, a comparative analysis is performed by considering two other formulations for the MPULSP found in the literature.

Keywords
Lot sizing problem · Multi-plant · Network flow

Introduction

Lot sizing plays an important role in improving the profits of the companies in addition to other goal effects, as, e.g., the environmental impacts. Many features and requirements of the companies influence and impose additional constraints and limitations on a lot sizing problem. As a consequence, a number of variants on this subject, each of them suitable for particular applications, can be found in the literature.

Karimi et al. [9] and Buschkühl et al. [3] defined, in a comprehensive literature review, the primary characteristics and existing variants of lot sizing problems. Nevertheless, Karimi et al. [9] do not mention lot sizing problems involving more than one plant, best known as multi-plant lot sizing problems (MPLSP). Buschkühl et al. [3] briefly discuss the MPLSP, but because of the general scope of their review of dynamic lot sizing problems, they do not go into detail on this problem. Highly relevant in certain industry sectors such as mattress companies [14], the MPLSP has received a reasonable amount of attention in the last two decades even though a very small number of solution methods has been studied to approach it. This chapter, therefore, focuses on the MPLSP presenting the state-of-the-art solution methods and mathematical models.

There are two cases of the MPLSP in the literature: the capacitated case (MPCLSP) [14] and the uncapacitated case (MPULSP) [13]. The studies presented in [4, 14, 17] approach two mathematical formulations for the capacitated case. To approach the uncapacitated case, it is enough to delete the capacity constraints of the capacitated models. Both problems have been proved to be NP-hard [13]. The solution methods of the literature mostly focus on the MPCLSP [11, 14, 15].

This chapter also introduces a network flow-based mathematical model for the MPULSP and computational experiments with benchmark MPCLSP instances ignoring the capacity limitations.

The Lot Sizing Problem

Lot sizing is a research topic widely studied due to its importance for the production planning. According to Karimi et al. [9], lot sizing belongs to the medium-term decision making, when it is necessary to determine the timing and how much of the

items to produce to meet the demands and minimize the total costs. This tactical planning is responsible for the productivity of the company, where the decisions must consider the requirements and limitations of the manufacturing process.

Many variants of the lot sizing problem and their solution methods can be found in the literature. Among the numerous features that define the different variations of the problem, we can highlight the finite planning horizon divided into periods, single or multiple production levels, single or multiple items, single or multiple machines in each plant, single or multiple plants, uncapacitated or capacitated, stochastic or deterministic demand, with or without setup time, and limited or unlimited inventory capacity or not. These characteristics define the primary traits of the classical lot sizing problems.

The next section further describes these features loosely based on [9].

Demands

The demand of an item is the required amount to produce in the desired production planning. This amount comprises the estimated value according to the history of the productive process or the quantity of the order of clients. The types of demands are stationary (static or dynamic). Stationary demands mean they do not vary over time. The values of dynamic demands may change over time [19].

Apart from being stationary or dynamic, the demands can be deterministic or non-deterministic. The determinism of demands means that they are known beforehand, and most of the existing studies in the literature consider this type of demand. The non-deterministic or stochastic demands depend on a probability function or an information subject to uncertainty.

Planning Horizon

Scheduling the production up to a certain time in the future is a relevant characteristic of lot sizing problems. In general, one defines this planning as finite, infinite, and rolling. The main characteristics regarding these types of planning horizon are enumerated as follows:

- The time interval of a finite planning horizon is limited.
- There is no time limit imposed in an infinite planning horizon. Demands, in this case, are generally stationary.
- The rolling planning horizon is a strategy that projects the actual planning horizon that has been scheduled to part of or to the entire next interval of time. This planning aims at keeping running the production of a firm based on past plannings.

Lot sizing problems with rolling planning horizon are hard to evaluate as discussed in [6].

Items

The production system can produce a single or multiple types of items. In single-item systems, only one type of item is considered as the final product. Multi-item systems involve the production of more than one type of item as the final product.

Multi-item production may be an item independent problem. In problems with item dependency, an item can only be manufactured after the production of another item. Dependencies on items are usually found in problems with more than one level.

Levels

The dependence between tasks performed for the transformation of the raw material into a final product is modeled by levels in the production system.

- Single level: In a single-level production system, the transformation of raw material into the final product (item) is direct. This means that there are no intermediate steps, that is, the production of items are independent.
- Multilevel: The production of some final product (item) depends on the operations performed at different levels of the production system. In other words, the production of an item depends on the availability of items produced at previous levels.

Plants

A production system must have a facility (plant) where the items are produced. There are problems in which there is more than one production facility. In these multi-plant lot sizing problems, each plant has its own demand and capacity, and lot transfers between plants are possible subject to the transportation costs.

Machines

A production facility may have a single or multiple machines to produce items. In particular, multiple identical parallel machines produce the same items with the same setup costs and times. The problem can be uncapacitated and capacitated when the machine production is limited by an upper bound and is located at the same facility.

Setup Structure

A setup cost is incurred to produce an item type, and a setup time is required to prepare the machine. These two components characterize the machine setup

structure in a production system, which can be classified as simple or complex. In the simple setup, the time and cost of preparation are not influenced by production in previous periods. In the complex setup, both time and cost of setup may depend on the sequencing of tasks and decisions taken in previous periods, as follows:

- Setup carry-over: If the preparation of the machine is ready at the end of a period for the production of a certain type of item, there is no need to re-prepare the machine in the beginning of the following period, if the same type of item is the first to be produced.
- Setup crossover: This type of setup enables a setup starting at the end of a period be interrupted and resumed in the beginning of the next period.
- Setup by similarity: The preparation is carried out taking into consideration the group of items to be produced. Similar items can be produced without additional preparation if they are produced in the sequence.
- Sequence-dependent setup: The preparation is dependent on the sequence determined for the production of the items.

Inventory and Backlogging

When the amount of production of an item exceeds the demand for the item in the period produced, the production stock may be subject to costs, and it may be restricted to the capacity of the storage warehouse, among other factors [14, 15]. In another case, a delay (backlogging) in the supply of demand is allowed, subject to some penalty costs.

On the Complexity

Wagner and Whitin [19] proved that the uncapacitated lot sizing problem with multiple items (or a single item), setup time, and deterministic demand can be solved in polynomial time by a dynamic programming algorithm. Bitran and Yanasse [1] proved that the capacitated lot sizing problem is NP-hard. Maes et al. [10] proved that to decide whether an instance of the capacitated lot sizing problem with setup times is feasible is NP-complete. The existing studies include new problems, reviews, and surveys, and they present methods, their computational complexity, and mathematical formulations [2, 3, 6, 8, 9, 12]. The computational complexity of the different variants of the lot sizing problem depends on their features, such as limited capacity and existence of setup times.

Among the plethora of variants and possibilities to approach this problem, this chapter limits the attention to the description of the multi-plant lot sizing problem with setup time and unlimited inventory.

Different from most uncapacitated lot sizing problems that, even with setup times have polynomial time solution, the MPULSP has been proved NP-hard [13].

The Multi-plant, Multi-item Lot Sizing Problem

The multi-plant lot sizing problem (MPLSP), where all plants produce the same items and have limited capacity, is known as multi-plant capacitated lot sizing problem (MPCLSP). For this problem, the literature on mathematical formulations presents two different mixed integer programs. First, in this chapter, the formulation proposed in [13] is presented.

Let one define the following parameters of the problem:

n : number of items;
m : number of plants;
p : number of periods;
cs_{ij} : setup cost for producing item i at plant j;
c_{ij} : unitary production cost of item i at plant j;
h_{ij} : unitary inventory cost of item i at plant j;
r_{jk} : unitary transportation cost of items from plant j to plant k;
d_{ijt} : demand of item i at plant j in period t;
b_{ij} : processing time of item i at plant j;
f_{ij} : setup time for preparing the machine for producing item i at plant j;
C_{jt} : capacity of production at plant j in period t.

To formulate the MPCLSP, Sambasivan and Schimidt [14] considered the following variables:

x_{ijt} : the quantity of item i produced at plant j in period t
y_{ijt} : binary setup variable that receives 1 if item i is produced at plant j in period t and 0, otherwise;
I_{ijt} : variable that controls the inventory of item i at plant j at the end of period t;
w_{ijkt} : variable that specifies the amount of item i transferred from plant j to plant k in period t.

$$\min \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{p}\left(cs_{ij}\,y_{ijt} + c_{ij}\,x_{ijt} + h_{ij}\,I_{ijt} + \sum_{k\neq j} r_{jk}\,w_{ijkt}\right) \tag{1}$$

subject to:

$$x_{ijt} + I_{ij,t-1} - I_{ijt} - \sum_{k\neq j} w_{ijkt} + \sum_{l\neq j} w_{iljt} = d_{ijt} \qquad \forall i, j, t \tag{2}$$

$$\sum_{i=1}^{n}(b_{ij}\,x_{ijt} + f_{ij}\,y_{ijt}) \leq C_{jt} \qquad \forall j, t \tag{3}$$

$$x_{ijt} \leq L\,y_{ijt} \qquad \forall i, j, t \tag{4}$$

$$y_{ijt} \in \{0,1\}, x_{ijt} \geq 0, I_{ijt} \geq 0, w_{ijkt} \geq 0 \qquad \forall i, j, t; j \neq k \tag{5}$$

The objective function (1) is the minimization of the sum of setup, production, inventory, and transfer costs. Constraints (2) balance inventory production transference and demand of items at each plant in each period t. Constraints (3) limit the time production capacity of plant-period (j, t). Constraints (4) ensure the production of item i at plant j in period t can occur only if the corresponding setup is performed. Finally, constraints (5) define the domain of the variables.

Silva [16] proposed a mathematical formulation for the MPCLSP based on the study found in [7]. Carvalho and Nascimento [4] refined this integer program and the formulation, presented next.

This formulation possesses the following parameters:

$$\overline{c}_{ijtku} = \begin{cases} 0, \text{ if } \quad u < t, \\ c_{ij} + \chi_{ijtku}, \text{ otherwise.} \end{cases}$$

where

χ_{ijtku}: is the inventory and transfer costs for producing one unit of item i at plant j in period t to meet the unitary demand of item i at plant k in period u. It is given by

$$\chi_{ijtku} = \min\{(u-t)h_{ij} + r_{jk}, \min_{1 \le v \le p}\{(u-t)h_{iv} + r_{jv} + r_{vk}\}, (u-t)h_{ik} + r_{jk}\} \tag{6}$$

$\overline{c}_{ijtku}$ is the cost of producing one unity of item i at plant j in period t plus the least inventory and transportation costs of this unity of i from plant j to plant k from period u to t. In addition to y_{ijt} defined as previously, the following variables are used:

x_{ijtku}: is the production of item i at plant j in period t to meet the demand of item i at plant k in period u.

The model of [4] for the MPCLSP is:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{p}\sum_{k=1}^{m}\sum_{u=1}^{p}(\overline{c}_{ijtku}x_{ijtku}) + \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{p}(cs_{ij}y_{ijt}) \tag{7}$$

subject to

$$\sum_{j=1}^{m}\sum_{t=1}^{u} x_{ijtku} = d_{iku} \qquad \forall(i,k,u) \tag{8}$$

$$\sum_{i=1}^{n}\left(f_{ij}y_{ijt} + \sum_{k=1}^{m}\sum_{u=t}^{p} b_{ij}x_{ijtku}\right) \le C_{jt} \qquad \forall(j,t) \tag{9}$$

$$x_{ijtku} \leq \min \left\{ d_{iku}; \left\lfloor \frac{C_{jt} - f_{ij}}{b_{ij}} \right\rfloor \right\} y_{ijt} \qquad \forall (i, j, t, k, u) \qquad (10)$$

$$x_{ijtku} \in \mathbb{N} \qquad \forall (i, j, t, k, u) \qquad (11)$$

$$y_{ijt} \in \{0, 1\} \qquad \forall (i, j, t) \qquad (12)$$

The objective function aims at minimizing the total costs: production, setup, inventory, and transference between plants. Constraints (8) ensure that the demands of all plants are met; constraints (9) ensure the production capacity at each plant in every period is not violated. In constraints (10), if item i is produced at plant j in period t, then y_{ijt} assumes value 1, that is, a corresponding setup cost must be added to the total cost. Constraints (11) and (12) define the domain of the decision variables.

It is noteworthy that the number of variables of this formulation asymptotically grows faster with the size than the variables of the model of [14]. Consequently, there is a major influence on the performance of exact solution methods in solving the problems regarding different sizes of instances.

As some applications involving MPLSP have a large number of items, heuristics are solution methods that play an important role in this problem, as discussed in the next section.

Solution Methods

In the literature, one can find a few heuristic methods specially designed for solving both MPULSP and MPCLSP. They are a simple heuristic based on lot transfers [14], a Lagrangian heuristic [15], a greedy randomized adaptive search procedure (GRASP) with path relinking [11], and Lagrangian heuristics [4].

The Lagrangian heuristic proposed in [15] aims at finding a heuristic solution for the relaxed uncapacitated problem and posteriorly transfers lots of production of items as an attempt of achieving a feasible solution for the MPCLSP. The authors take into consideration the mathematical formulation proposed in [14] to develop their method. They evaluate the performance of the Lagrangian heuristic by using artificial instances they generated. As a result of the experiments, the Lagrangian heuristic systematically achieved better results than the first heuristic proposed to solve the MPCLSP, found in [14].

Later, Nascimento et al. [11], besides proposing a novel hybrid metaheuristic, set forth a new set of instances with a more diversified structure and classified them as indicated in the section of experiments of this chapter. In their experiments using the set of instances introduced in [15], they show that the hybrid metaheuristic, GPheur, had a better performance than the Lagrangian heuristic [15]. In order to better investigate the performance of their metaheuristic, they carry out experiments with the set of instances they suggested. For these instances, GPheur could not find optimal solutions for the hard instances, even though the heuristic method achieved better results than CPLEX v. 7.5.

More recently, Carvalho and Nascimento [4] proposed a pair of Lagrangian heuristics that make use of CPLEX v.12.6 to find the optimal solution of the

uncapacitated problem, by considering the model introduced in [16]. One is a Lagrangian heuristic, named by the authors as Lag, and the other is a hybrid version, referred as LaPRe. For finding feasible solutions for the capacitated problem, both feasibility strategies, adapted for tackling the MPCLSP from [18] and the local search phase of the GRASP from [11], were embedded in the strategy. To enhance the quality of the solutions, the path relinking, also proposed in [11], was used to compose the hybrid Lagrangian heuristic. In experiments, the authors show that both LaPRe and Lag significantly outperformed GPheur achieving feasible solutions for the whole set of instances. However, for smaller instances, CPLEX v. 12.6 outperformed the Lagrangian heuristics on most of the classes of instances. On the other hand, when considering large instances, CPLEX could not find good solutions for all instances in a limit of 1800 s established by the authors to run each instance of the experiment, whereas the Lagrangian heuristics presented better lower and upper bounds.

Network Flow-Based Formulation

Because of the importance of the multi-plant lot sizing problems, this chapter gives a new insight to the MPULSP. For this, it presents a network flow-based formulation for the MPULSP, different from the formulations proposed in the literature.

In unlimited capacity lot sizing problems, the production of items is independent. Therefore, the formulation will disregard item indices, without loss of generality.

In a network flow, consider the following parameters:

cs_j : setup cost of an item at plant j;
d_{jt} : demand in plant j in each period t;
h_j : inventory cost at each plant j;
c_j : production cost of an item at plant j;
r_{jk} : transfer cost between plants j and k;
c_j : production cost of an item at plant j;
M : sum of the demands of every plant-period.

Therefore, if a machine is ready to produce an item, the cost of producing at plant j in period t to meet the demand at plant k in period u is:

$$\chi'_{jtku} = c_j + \min\{(u-t)h_j + r_{jk}, \min_{1\leq v\leq p}\{(u-t)h_v + r_{jv} + r_{vk}\}, (u-t)h_k + r_{jk}\}$$

In the most general case, other plants can be considered in the transfer of the produced item. To calculate the costs of the arcs, it is necessary to determine them by solving the minimum cost problem between the pair of vertices of the network. Consider a toy example with two plants and three periods as illustrated in Fig. 1.

In Fig. 1, node 0 is the source of the network. It receives the sum of the demands of all periods in each plant. The nodes represented as yellow rectangles are the

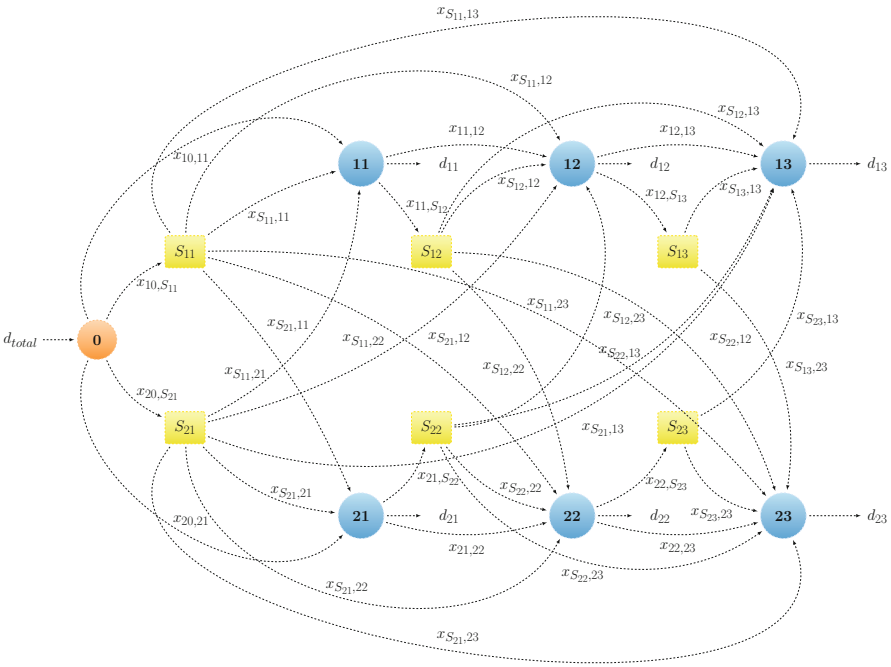


Fig. 1 Example with *two* plants and *three* periods

setup nodes, whereas round blue nodes indicate the demands to be met. The labels of yellow nodes, s_{kt}, indicate the setup at plant k in period t. The labels of round nodes, kt, represent the planning in the plant k at period t. A sink node is associated with each blue node, to represent the addressing of the corresponding demand. Arcs with heads incident to demand nodes are only those with tails in setup nodes or with tails from the immediately previous period at the same plant. These conditions ensure the machine to be prepared for producing that item. Aggregated to such arcs are the aforementioned costs. Moreover, arcs with head in setup nodes have tails in the production or source nodes.

To formally introduce the network flow-based problem, referred here as NYC, consider the following variables:

y_{jt} : binary setup variable that receives 1 if there is setup to produce demands at plant j in period t and 0, otherwise;

$x_{jt-1,jt}$: is an arc that carries demands from plant j not produced up to period $t-1$;

$x_{jt-1,s_{jt}}$: is the amount of demand not met up to period $t-1$ to be produced at plant j in period t;

$x_{s_{jt},ku}$: is the amount of demand produced at plant j in period t to meet the demand of plant k in period u.

The domains of the indices are the following: $j, k \in \{1, \ldots m\}; t \in \{1, \ldots, p\}$; and $u \in \{t, \ldots, p\}$. The general formulation is, thus, the following:

$$\min \sum_{j=1}^{m} \sum_{k=1}^{m} \sum_{t=1}^{p} \sum_{u=t}^{p} \chi'_{jtku} x_{s_{jt},ku} + \sum_{j=1}^{m} \sum_{t=1}^{p} cs_j y_{jt} \tag{13}$$

subject to:

$$\sum_{j=1}^{m} (x_{j0,s_{j1}} + x_{j0,j1}) = \sum_{j=1}^{m} \sum_{t=1}^{p} d_{jt} \tag{14}$$

$$x_{jt-1,s_{jt}} = \sum_{k=1}^{m} \sum_{u=t}^{p} x_{s_{jt},ku} \qquad \forall j, t \tag{15}$$

$$x_{jt,s_{j,t+1}} + x_{jt,jt+1} - x_{jt-1,jt} = 0 \qquad \forall j, t = 1, \ldots, p-1 \tag{16}$$

$$\sum_{j=1}^{m} \sum_{t=1}^{u} x_{s_{jt},ku} = d_{ku} \qquad \forall k, u \tag{17}$$

$$x_{jt-1,s_{jt}} \leq M y_{jt} \qquad \forall j, t \tag{18}$$

$$y_{jt} \in \{0, 1\} \qquad \forall j, t \tag{19}$$

$$x_{s_{jt},ku}, x_{jt-1,s_{jt}}, x_{jt-1,jt} \geq 0 \qquad \forall j, k, t \leq u \tag{20}$$

Constraints (14) define the set of source nodes assigning to them the sum of the demands of the plants. Constraints (15) refer to the flow conservation constraints of setup nodes. Constraints (16) ensure the flow conservation of production nodes. Constraints (17) ensure that the demands of all plants are met. Constraints (18) ensure the setup variables to be 1 if there is production in period t, whereas constraints (19) and (20) define the domains of the variables of the model.

In the next section, computational experiments were carried out to evaluate the performance of CPLEX v.12.6 with the three models discussed in this chapter.

Computational Experiments

For the analysis of the proposed network flow-based model, we performed an experiment comparing its results with those achieved by CPLEX v.12.6 considering the formulations presented in [14] and [16] refined and introduced in [4].

All algorithms were implemented in the C++ language and the experiments carried out in an Ubuntu Server 14.04, Intel Core i7 with 3.4 GHz, 500 MB CPU, and 16 GB DDR RAM.

For the experiment, we employed a set of instances generated as suggested in [11]. It is a set of 8 classes of datasets with 160 instances. The division into classes was due to three main characteristics of the instance: capacity, setup cost, and

setup time. Accordingly, the instances were classified as having tight (T) or normal (N) capacity. Considering $\beta = \sum_{t=1}^{p} \sum_{i=1}^{n} \frac{(d_{ijt} b_{ij} + f_{ij})}{p}$, instances with normal and tight capacities have their parameter C_{jt} set as $1 \times \beta$ and $0.9 \times \beta$, respectively.

Regarding the setup costs, instances were defined with low (L) or high (H). In the former, cs_{ij} was chosen randomly within interval [5.0; 95.0]. In the latter, instances were chosen randomly within interval [50.0; 950.0].

To define each f_{ij}, it was picked randomly a value within interval [10.0; 50.0] when considering an instance with low setup time (L), and from interval [15.0; 75.0] if it was a high setup time (H) instance.

The remaining parameters, c_{ijt}, h_{ij}, r_{jk}, b_{ij}, and d_{ijt}, were set selecting at random values from the respective intervals: [1.5; 2.5], [0.2; 0.4], [0.2; 0.4], [1.0; 5.0], and [0, 180.0].

Therefore, a class of instances of the type NHL has instances with normal capacity, high setup cost, and low setup time. In the first set of instances, generated in [11], each instance corresponds to a problem with 12 periods, 2 plants, and 100, 200, 300, or 400 items. For each combination, the set has 5 problems, totaling 20 instances in each class.

Setup of the Experiments

To describe the results of the experiments and computationally compare the models, the performance profiles of [5] will be employed. This profile consists in a plot that displays the relation between the metric to be evaluated, in this case, the time to solve every instance, and the performance of each algorithm in solving the instances, being the reference the best solution times found among all algorithms. For this, consider the ratio in Eq. (21).

$$\rho_{a,\tau} = \frac{1}{|\mathscr{S}|} \{s \in \mathscr{S} : r_{sa} \leq \tau\} \tag{21}$$

The factor τ in Eq. (21) means specifically the factor that multiplied by the best solution found considering all algorithms results in a lower bound for the solution found by algorithm a for instance s. Therefore, r_{sa} can be calculated as indicated in Eq. (22).

$$r_{sa} = \frac{t_{sa}}{\min\{t_{sa'} : a' \in \mathscr{A}\}} \tag{22}$$

In Eq. (22), t_{sa} corresponds to the metric used for evaluating the solutions that must be minimized. Therefore, r_{sa} is the performance of algorithm a to solve instance s in comparison to every algorithm. Consequently, $\rho_{a,\tau}$ is the percentage of solutions of the set of instances solved by algorithm a that respects the bound τ.

Results of the Experiments

In this experiment, the network flow-based formulation, named NYC, the model presented in [14], SS02, and the formulation adapted in [4], named CN16, were compared. As mentioned before, the time to achieve the optimal solution is the object of assessment. For this, the first performance profiles displayed in Fig. 2 assesses only the small-sized instances (those with a number of items lesser than or equal to 50). In this case, it was possible to consider the time elapsed considering SS02, since it was comparable with NYC.

According to the results of the experiment, CN16 presents a clear advantage in performance in comparison to the other formulations. NYC achieves better results than SS02 even though their performances were very close. However, the difference between the performance profiles of NYC and SS02 becomes more evident with the increase in the number of items.

Figure 3 plots the performance profiles considering the larger instances (with a number of items larger than or equal to 100). This evaluation only takes into consideration the results achieved using models CN16 and NYC since to run instances modeled as SS02 was time-consuming.

Although the performance of NYC was poorer than CN16, it is noteworthy that their difference in performance fell in relation to the smaller instances. The value of

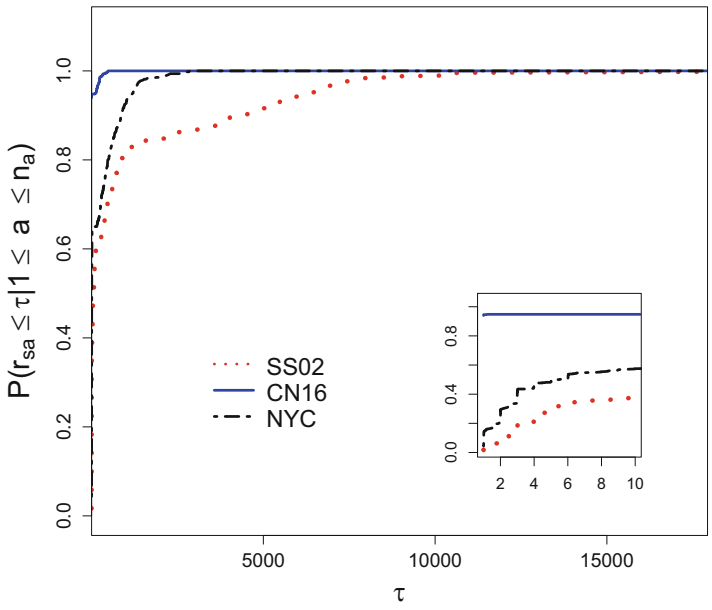


Fig. 2 This figure shows the performance profiles considering the two existing models in the literature, SS02 and CN16, and the formulation proposed in this chapter, NYC

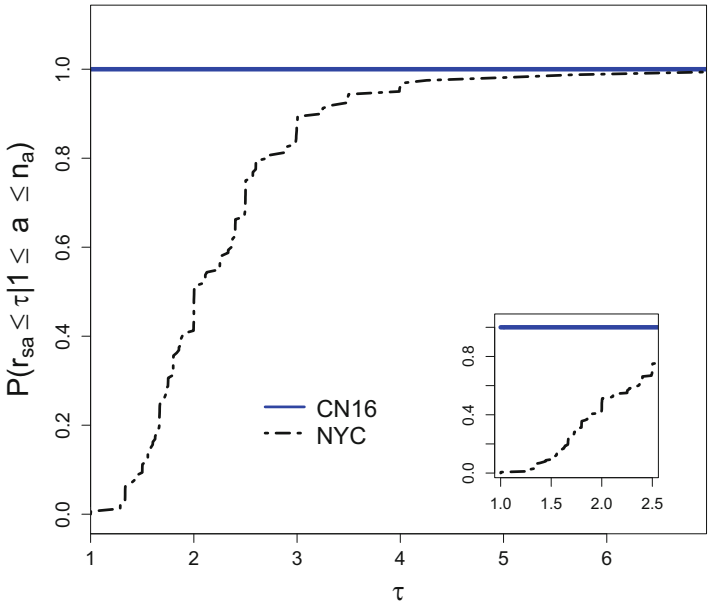


Fig. 3 This figure shows the performance profiles considering the existing model in the literature CN16 and the formulation proposed in this chapter, NYC

τ for NYC to have at least half of the instances with results comparable to CN16 is 6. On the other hand, for larger instances it was approximately 2.

Conclusions

Lot sizing plays a fundamental role in production planning. Considering the planning horizon phased into periods, this problem consists in making decisions with regard to plant, period, item, and amount of items to produce to meet all demands without delay. In multi-plant capacitated lot sizing problem, it is possible to schedule items on multiple machines/plants enabled by transferring items to plants where they are demanded. This is the subject of this study about which is we presented a brief literature review, primarily focusing on existing solution methods and approaches to its solution.

A few efficient solution methods were specially designed to tackle the multi-plant capacitated lot sizing problem (MPCLSP). The hard nature of this problem, for which not even the uncapacitated variant has a polynomial algorithm to find optimal solutions, may be the reason behind the lack of interest in developing frameworks. Nevertheless, two Lagrangian heuristics recently proposed achieved good results mainly on large-scale instances and instances with high setup costs. For smaller instances and those with low setup costs, CPLEX significantly outperformed them.

Despite being NP-hard, few formulations were specially designed for the multi-plant uncapacitated lot sizing problem (MPULSP). The relevance in studying the MPULSP, besides approaching production lines that do not have capacity constraints, is to enable a study of strategies that relax capacity constraints to browse the search space for feasible solutions for the MPCLSP. This chapter introduces a network flow-based formulation for the MPULSP and evaluates a recently proposed mathematical formulation for the MPCLSP by relaxing the capacity constraints to contrast their performance in solving instances according to solver CPLEX. Moreover, the first proposed formulation for the MPULSP was investigated in the computational experiments. Experiments with instances found in the literature were carried out by evaluating the time to solution, assessed with Dolan-Moré performance profiles.

Cross-References

▸ GRASP
▸ Matheuristics

References

1. Bitran G, Yanasse H (1982) Computational complexity of the capacitated lot size problem. Manag Sci 28:1174–1186
2. Brahimi N, Dauzere-Peres S, Najid NM, Nordli A (2006) Single item lot sizing problems. Eur J Oper Res 168:1–16
3. Buschkühl L, Sahling F, Helber S, Tempelmeier H (2010) Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. OR Spectr 32(2):231–261
4. Carvalho DM, Nascimento MCV (2016) Lagrangian heuristics for the capacitated multi-plant lot sizing problem with multiple periods and items. Comput Oper Res 71:137–148
5. Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Prog Ser A 91:201–213
6. Drexl A, Kimms A (1997) Lot sizing and scheduling – survey and extensions. Eur J Oper Res 99:221–235
7. Eppen GD, Martin RK (1987) Solving multi-item capacitated lot-sizing problems using variable redefinition. Oper Res 35(6):832–848
8. Jans R, Degraeve Z (2008) Modeling industrial lot sizing problems: a review. Int J Prod Res 46:1619–1943
9. Karimi B, Ghomi SMTF, Wilson JM (2003) The capacitated lot sizing problem: a review of models and algorithms. OMEGA 31:365–378
10. Maes J, McClain JO, Wassenhove LNV (1991) Multilevel capacitated lotsizing complexity and lp-based heuristics. Eur J Oper Res 53:131–148
11. Nascimento MCV, Resende MCG, Toledo FMB (2010) GRASP with path-relinking for the multi-plant capacitated lot sizing problem. Eur J Oper Res 200:747–754
12. Quadt D, Kuhn H (2008) Capacitated lot-sizing with extensions: a review. 4OR 6(1):61–83.
13. Sambasivan M (1994) Uncapacitated and capacitated lot sizing for multi-plant, multi-item, multi-period problems with inter-plant transfer. PhD thesis, University of Alabama, Tuscaloosa
14. Sambasivan M, Schimidt CP (2002) A heuristic procedure for solving multi-plant, multi-item, multi-period capacited lot-sizing problems. Asia Pac J Oper Res 19:87–105

15. Sambasivan M, Yahya S (2005) A Lagrangean-based heuristic for multi-plant, multi-item, multi-period capacitated lot-sizing problems with inter-plant transfers. Comput Oper Res 32:537–555
16. Silva DH (2013) Métodos híbridos para o problem de dimensionamento de lotes com múltiplas plantas. Master's thesis, Universidade de São Paulo – São Carlos
17. Silva DH, Toledo FMB (2012) Dimensionamento de lotes com múltiplas plantas: comparação entre dois modelo. Simpósio Brasileiro de Pesquisa Operacional, pp 1638–1646
18. Toledo FMB, Armentano VA (2006) A Lagrangean-based heuristic for the capacitated lot-sizing problem in parallel machines. Eur J Oper Res 175:1070–1083
19. Wagner HM, Whitin TM (1958) Dynamic version of the economic lot size model. Manag Sci 5:89–96

Trees and Forests

46

Andréa Cynthia Santos, Christophe Duhamel, and Rafael Andrade

Contents

Introduction 1308
Basic Features for Trees and Forests 1311
Data Structures for Trees 1314
Encodings for Trees 1315
A Lagrangian Heuristic for DCMST 1317
Problem Formulation 1317
A Subgradient Procedure for DCMST Problem 1319
A Greedy Heuristic for DCST 1321
Improving DCSTs with a Local Search 1322
A Lagrangian Heuristic for DCMST 1322
Evolutionary Heuristic for a Generalization of BDMST 1324
Problem Definition 1325
An NSGA-II for Bi-MDCST 1327
Conclusion 1329
Cross-References 1330
References 1330

A. C. Santos (✉)
ICD-LOSI, UMR CNRS 6281, Université de Technologie de Troyes, Troyes Cedex, France
e-mail: andrea.duhamel@utt.fr

C. Duhamel
LIMOS-UBP, UMR CNRS 6158, Université Blaise Pascal, Aubière Cedex, Clermont-Ferrand, France
e-mail: christophe.duhamel@isima.fr

R. Andrade
Departamento de Estatística e Matemática Aplicada, Centro de Ciências, Universidade Federal do Ceará, Fortaleza, Brazil
e-mail: rca@lia.ufc.br

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_49

Abstract

Trees and forests have been a fascinating research topic in Operations Research (OR)/Management Science (MS) throughout the years because they are involved in numerous difficult problems, have interesting theoretical properties, and cover a large number of practical applications. A tree is a finite undirected connected simple graph with no cycles, while a set of independent trees is called a forest. A spanning tree is a tree covering all nodes of a graph. In this chapter, key components for solving difficult tree and forest problems, as well as insights to develop efficient heuristics relying on such structures, are surveyed. They are usually combined to obtain very efficient metaheuristic algorithms, hybrid methods, and matheuristics. Some emerging topics and trends in trees and forests are pointed out. This is followed by two case studies: a Lagrangian-based heuristic for the minimum degree-constrained spanning tree problem and an evolutionary algorithm for a generalization of the bounded-diameter minimum spanning tree problem. Both problems find applications in network design, telecommunication, and transportation fields, among others.

Keywords

bi-objective heuristic · DBMST · DCMST · forest · heuristics · Langrangian heuristic · tree

Introduction

Trees and forests have been a fascinating topic in Operations Research (OR)/Management Science (MS) throughout the years because they are the core of difficult problems, have interesting theoretical properties, and cover a large number of practical applications. Their interest remains intact due to the technical and scientific challenges and the rich diversity of applications using such structures. For instance, several applications have been recently addressed: people tracking is modeled as a minimum cost arborescence problem by [32]; cluster-based topologies with connecting requirements are defined as a minimum spanning tree (MST) to improve wireless sensor network lifetime in [51]; peer-to-peer distributed interactions are studied by [39] as a spanning tree, where end-to-end delays are minimized; reliable telecommunication networks have been investigated by [54], in which redundancy is added to spanning trees by introducing k-cliques ($k \geq 2$); and difficult MST problems have been investigated including uncertain data [35], new variants for optical multicast network design [55], or even multiobjective MST versions [52, 56].

Let $G = (V, E)$ be a finite, undirected, connected, and simple graph with a set V of vertices and a set E of edges, where $n = |V|$ and $m = |E|$. A tree $\mathcal{T} = (V', E')$ is a connected subgraph of G with no cycles, such that $V' \subseteq V$ and $E' \subset E$. Whenever $V' = V$, the corresponding tree defines a spanning tree of G. An arborescence is a special directed tree with a root node r. Moreover, a

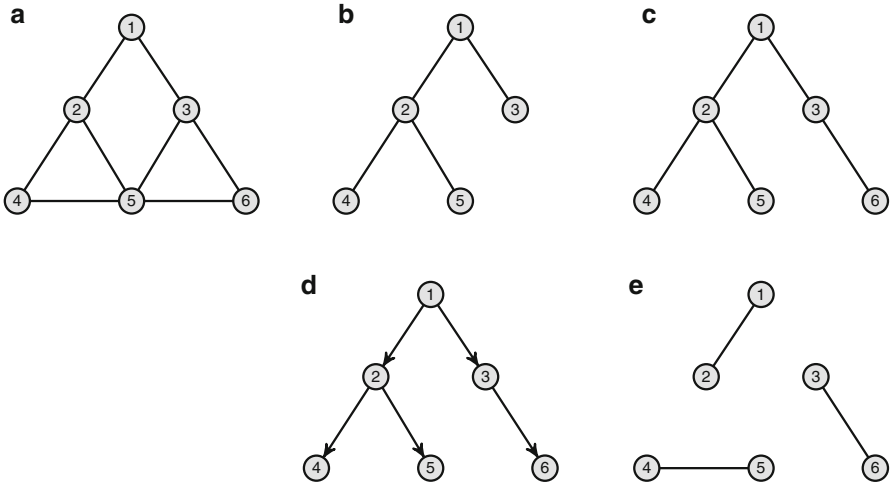


Fig. 1 Examples of tree, spanning tree, arborescence and forest. (**a**) Graph G'. (**b**) Tree. (**c**) Spanning tree. (**d**) Arborescence. (**e**) Forest

forest is a set of disjoint trees $\mathcal{F} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_l\}$. An example of graph G' is given in Fig. 1a, followed by examples of a tree, a spanning tree, an arborescence, and a forest of G', respectively, in Fig. 1b–d where $r = 1$, and (e). In this chapter, we assume familiarity of the reader with basic graph definitions such as connected graphs, connected components, subgraphs, paths, cycles, edges incident to nodes, node degree, etc.

The basic minimum spanning tree (MST) problem is defined on a weighted graph G, where a cost $c_{ij} \geq 0$ is associated with every edge $[i, j] \in E$. It aims at finding a spanning tree $\mathcal{T}$ of G such that its total cost is minimized. The number of possible MSTs for a graph is very high with up to n^{n-2} for complete graphs [12]. However, polynomial-time algorithms such as Prim's, Kruskal's, Boruvka's, and cycle elimination algorithm [14] are available to compute an MST. The general idea of Prim's algorithm is to extend an MST from an initial node $i \in V$. At each step it includes the cheapest-weight edge $[i, j] \in E$ such that one of its extremities belongs to the tree, while the other one does not. Thus, the connected component is extended until all nodes $i \in V$ belong to the solution. Kruskal's algorithm performs forest merging as follows. Initially, each vertex is a tree. Then, at each iteration, the cheapest edge connecting two forests is selected, and the two forests are merged. Boruvka's algorithm is quite similar to Kruskal's algorithm, except that, at each iteration, the cheapest edge connecting each forest to another one is selected. Then the merges are done accordingly. Both Kruskal's and Boruvka's algorithms can be developed with asymptotic complexity of $O(m \log n)$, as well as Prim's algorithm using binary heaps. Moreover, Prim's algorithm complexity can be improved to $O(m + n \log n)$ by using Fibonacci heaps; see [14] for details. The cycle elimination

algorithm scans the graph using a depth-first search (DFS) strategy. The edge $[i, j]$ is added to the tree whenever a vertex $j \in V$ not yet visited is found from the current node i of the DFS. However, if j has already been visited, it means a cycle has been found. In this case, the highest-weight edge from the found cycle is removed from the incumbent partial solution. DFS complexity is $O(m + n)$. Thus, considering a complete graph, the cycle elimination algorithm runs in $O(mn)$ since there are $O(m - n - 1)$ possible back edges and a cycle scan is done in $O(n)$. Several NP-hard problems rely on an MST with additional constraints on nodes, on edges, or even on the MST general structure. Thus, algorithms that compute an MST are usually adapted for such NP-hard extensions in order to define basic heuristics and obtain initial feasible solutions.

A large number of difficult problems rely on trees and forests. Studies [10, 22, 33, 41] provide a collection of such problems, along with their complexity analysis. An example where constraints on the tree structure make the problem difficult is the Steiner Tree problem [3, 24, 34, 38]. Given a subset $N \subset V$ of Steiner nodes (also referred as terminal nodes), the Steiner Tree problem consists in determining a particular minimum tree covering N. Also, setting constraints on the nodes or on the edges (paths) of an MST can transform the problem into an NP-hard problem. For instance, the degree-constrained minimum spanning tree (DCMST) problem [4, 13, 15] consists in finding an MST such that each vertex has a degree not larger than a maximum given value $k \in \mathbb{N}^*$. The bounded-diameter minimum spanning tree (BDMST) problem [26,40,52] is an example of a difficult problem where constraints are imposed on the MST diameter, i.e., the diameter of a tree is the number of edges in the longest path among any pair of vertices. Thus, the BDMST aims at finding an MST where the unique path between any pair of nodes does not exceed a given number of edges. Basic problems can also become difficult due to the nature of the data. For instance, the Minimum Arborescence problem [20] is defined in a digraph with real cost values associated with each arc. The objective is to find a minimum cost arborescence. Thus, for problems involving trees and forests, adding constraints or even changing few parameters or data can turn a problem in P into one that is NP-complete.

Key components for solving difficult tree and forest problems, as well as insights to develop efficient heuristics relying on such structures, are surveyed here. They are usually combined to obtain very efficient metaheuristics, hybrid methods, and matheuristics. Section "Basic Features for Trees and Forests" is dedicated to a number of basic heuristics, local searches, etc. Then, two applications are described as examples, respectively, in sections "A Lagrangian Heuristic for DCMST and Evolutionary Heuristic for a Generalization of BDMST". Two different strategies to handle the difficult constraints are presented in these sections. The strategy used for DCMST consists of removing the difficult constraints and adding them into the objective function, following a Lagrangian relaxation. Another strategy is considered for the BDMST, where the difficult constraints are addressed as a new criterion and a bi-objective genetic algorithm is used.

Basic Features for Trees and Forests

Several heuristics, local search algorithms, operators, different encodings, and perturbations are available in the literature to build trees and forests. They are very often combined to produce sophisticated methods. The most common heuristic structures are classified as shown in Fig. 2. The idea of this classification is not to cover all available heuristics for trees and forests, but to introduce the most used strategies having a strong potential to derive other heuristics for various difficult problems. There are two major classes of heuristics: constructive and improvement heuristics. The former adds edges or nodes at each iteration to obtain a feasible solution. The latter starts with an initial solution (e.g., a spanning tree), not necessarily feasible, and try to improve it in order to obtain better feasible solutions. Constructive and improvement heuristics are presented below, in such a way they can be adapted and applied to different difficult problems on trees and forests, where nodes, edges, or structural additional constraints are considered.

Constructive heuristics generate a feasible solution iteratively, using mainly two strategies: tree expansion and tree fusion. In the tree expansion strategy, one node is added at a time by means of an edge $e = [i, j] \in E$, where one extremity of e belongs to the solution under construction and the other extremity does not. Thus the partial incumbent solution is always a connected component. Tree expansion heuristics are usually implemented using Prim's algorithm. Examples of tree expansion heuristics are the One Time Tree (OTT) [1] and randomized greedy heuristic (RGH) [47] developed for the BDMST. Fusion heuristics start from initial disconnected forests and try to connect them by adding an edge $e = [i, j] \in E$ at a time. Thus a partial solution contains several connected components and Kruskal's algorithm can be adapted and applied [37] (see section "A Greedy Heuristic for DCST"). The way new edges/vertices are selected for inclusion in partial solutions (e.g., constructive heuristics) or for removal from a solution (e.g., improvement heuristics), after evaluating the objective function, has a strong impact on the final

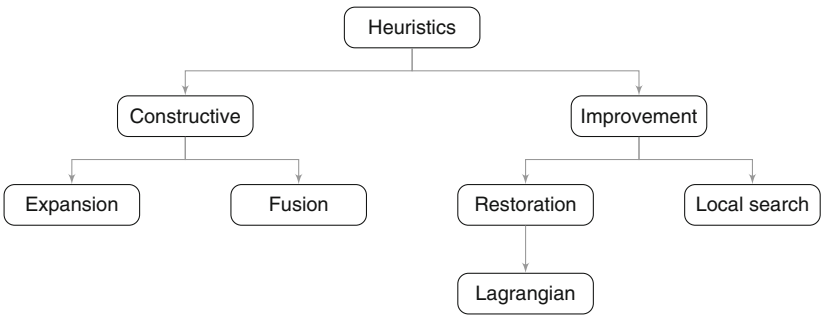


Fig. 2 A classification of heuristics for trees

solution quality. The most common ways to select edges/vertices are greedy, semi-greedy, and random. They can also be handled considering a single criterion or multi-criteria. Without loss of generality and considering a minimization function, a greedy criterion selects the edge/vertex insertion that leads to the smallest increase in the objective function. One way to handle a semi-greedy criterion is to build a list of good candidates to enter the solution, not necessarily the best ones, and make a random choice from this list. Another idea frequently found in the literature is to consider a normalized ratio between the cost and the impact on difficult constraints, e.g., a ratio between the edge cost and the additional increase in the diameter for the BDMST. Various criteria can also be addressed using a priority order instead of a ratio. For instance, one can use a second optimization criterion to select one edge, e.g., the impact on difficult constraints, whenever several edges with the smallest costs occur.

Improvement heuristics start from a solution, not necessarily feasible, and try to obtain a feasible solution by exchanging edges in the tree (resp. in a forest) with edges outside the solution. They can be roughly classified as restoration and local search heuristics. The restoration heuristics try to transform solutions in feasible ones by minimizing violations. A classical example of restoration heuristics is Lagrangian heuristics which have appeared with the pioneering works of [29, 30]. They have been broadly applied to a number of difficult problems relying on trees and forests [5, 16, 42]. The general idea is to relax difficult constraints and transfer them to the objective function, keeping a simple problem that is refined until a good solution to the overall problem is obtained. Consider a generic NP-hard optimization problem given by $Z = \{\min cx : Ax \geq b, Bx \geq d, x \in \{0,1\}^t\}$, where A and B are matrix with rational coefficients having dimensions $r \times t$ and $s \times t$, respectively; b, c, and d are rational vectors, and $x \in \{0,1\}^t$ are integer variables. Let $\{\min cx : Bx \geq d, x \in \{0,1\}^t\}$ be an easy problem that can be solved in polynomial time, and $Ax \geq b$ be the difficult constraints. Thus, Lagrangian multipliers $\lambda \in \mathbb{R}^r_+$ are associated with the difficult constraints and added into the objective function. The Lagrangian problem (LP) is given by $LP = \{\min cx + \lambda(b - Ax) : Bx \geq d, x \in \{0,1\}^t\}$ which is a lower bound on Z. The best lower bound is given by $LP(\lambda^*) = \max\{LP(\lambda), \lambda \geq 0\}$, called dual Lagrangian problem. The dual Lagrangian problem can be solved by using, for example, the subgradient methods of [21] or the volume algorithm of [7] or even by computing the Lagrange multipliers approximately using the multiplier adjustment method [9]. Theoretical results to determine the conditions stating when a Lagrangian relaxation can be better than a linear relaxation are provided by [28]. Obviously, the mathematical formulation and the decomposition will strongly impact the quality of solutions produced using Lagrangian relaxation. This kind of restoration heuristic has an interesting property: at each iteration dual and primal solutions can be built.

Local searches are very sophisticated improvement heuristics which contribute to producing very good local optima (eventually global optima). The most common local search moves, broadly applied to trees, are the *edge drop* move and the *edge insertion* move [18, 40]. Given an initial feasible solution for a difficult problem

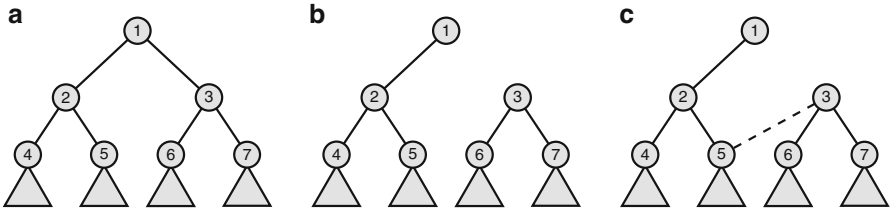


Fig. 3 Example of an *edge drop* move. (**a**) Tree. (**b**) Disconnecting 1 and 3. (**c**) Reconnecting 3 and 5

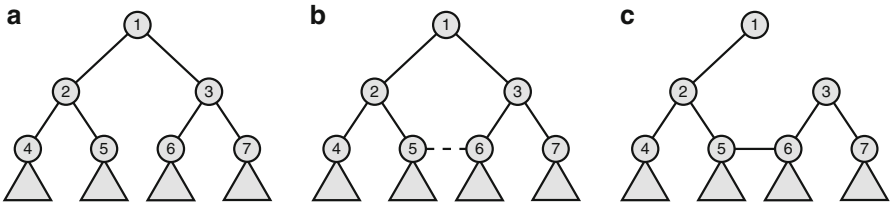


Fig. 4 Example of an *edge insertion* move. (**a**) Tree. (**b**) Connecting 5 and 6. (**c**) Disconnecting 1 and 3

relying on a tree, the *edge drop* move consists of removing an edge from the tree. This produces two distinct connected components which are reconnected using an edge not in the tree. The greedy and semi-greedy criteria and multi-criteria can be used to choose the edge to enter the tree. Figure 3 illustrates an example, where Fig. 3a is a feasible initial solution. Then edge $[1, 3]$ is removed as shown in Fig. 3b, generating two separate connected components. Consider $[3, 5]$ as the edge which produces the best improvement possible. Thus it enters the solution following a greedy strategy as shown in Fig. 3c.

On the other hand, in the *edge insertion* move, an edge is first inserted in the tree. By definition of a tree, it necessarily results in a cycle. Then, an edge may be removed from the cycle. Figure 4 depicts an example for this move. A feasible solution is presented in Fig. 4a. In the sequel, edge $[5, 6]$ is introduced and the cycle $\{5, 6, 3, 1, 2, 5\}$ is formed as shown in Fig. 4b. Suppose that removing edge $[1, 3]$ will produce the highest gain in the objective function. Thus this edge is dropped following a greedy strategy as depicted in Fig. 4c.

A number of variations on the *edge drop* and *edge insertion* moves are found in the literature. These moves are also referred as 1-opt since one edge in the incumbent solution is replaced by another one that does not belong to the solution. When applied to trees, two strategies are available to perform such a 1-opt move: either first dropping an edge or first inserting an edge. This may lead to different final solutions. The k-opt generalizes the 1-opt move by replacing k edges from an incumbent solution. This move is formalized in section "A Lagrangian Heuristic for DCMST."

Data Structures for Trees

Choosing the right data structure for representing and manipulating a solution is a critical task since it directly impacts the complexity of basic operations. They include both building and modifying a solution. Two mapping functions are associated with the data structures: how to store the information from a given tree (encoding) and how to obtain a tree out of the information (decoding). In this section, a basic direct representation is presented as well as several data to handle additional constraints on the trees. In section "Encodings for Trees," other representations are presented, along with the way to obtain the associated tree. Several insights are given regarding their use in evolutionary algorithms.

Let $G = (V, E)$ and $s \in V$ be, respectively, a graph defined as before and an arbitrary root vertex [14]. The following data structure can be used for directed and undirected trees, depending on the root node which may or not be artificial. The predecessor j of a vertex i in a tree is the first vertex in the path from i to s. By definition of a tree, j is unique and only depends on the choice of s. A successor j of a vertex i is a vertex whose predecessor is i. One of the simplest ways to store the information of a tree in a fixed-size data structure is to use the predecessors. This direct representation keeps the predecessors in a vertex-indexed vector $pred$. Thus $pred(i)$ is the predecessor of vertex i or, alternatively, the edge between i and $pred(i)$. The root has no predecessors; then $pred(s) = \emptyset$. Such a structure allows an $O(1)$ access to each predecessor and to each edge connecting the subtree rooted at i with the rest of the tree. It also allows a scan of all edges of a tree in $O(n)$.

Additional redundant data may be useful to perform tree manipulation and to allow efficient checking operations. One such data is the list of all the direct successors of a vertex. It can be explicitly defined as a list of vertices for each vertex, as well as be implicitly stored by adding two attributes to each vertex i: its first successor $succ(i)$ and $next(i)$, the next successor for $pred(i)$. For instance, if the direct successors of vertex a are $\{b, c, d\}$, the first successor of a is $b = succ(a)$. The successors of a are as follows: $c = next(b)$, $d = next(c)$ and $next(d) = \emptyset$. This encoding shares similarities with the forward star structure presented in [2] for graphs, and the implicit list can be made bidirectional by adding an attribute $prev(i)$ to each vertex which stores the previous successor in the list of $pred(i)$. This way, $O(1)$ operations such as successor insertion or deletion are ensured. The number of direct successors $nb_succ(i)$ can be kept as well.

Successor and predecessor structures allow managing several operations such as finding a cycle for *edge insertion* moves, finding the cheapest-weight costs for *edge drop* moves, and, with a few additional information, addressing difficult constraints like degree and diameter constraints. For instance, the $depth(i)$ of a vertex i is equal to the number of edges in the unique path from a given root vertex s to i. It can be used for checking hop and diameter constraints.

Scanning a cycle created by adding an edge $[i, j] \in E$ into a spanning tree can be done efficiently using the predecessor structure. First, each node in the path $i \rightarrow s$ is set as visited. The same is done in the path $j \rightarrow s$, stopping as soon as a given vertex

k has already been visited. Thus k belongs to both paths $i \rightarrow s$ and $j \rightarrow s$. Getting the edge with the highest cost can be done by keeping the edge with the highest cost found when scanning the path $i \rightarrow s$ (resp. $j \rightarrow s$) at each visited node. A modified DFS or breadth-first search (BFS) starting at i and stopping as soon as j is found could also be used. However, this would require more sophisticated tree data structures to obtain $O(n)$ complexity as for the predecessor vector. This way, the *edge insertion* move can be performed in $O(n)$.

Another useful data structure can be defined to quickly check if a vertex belongs to a subtree. Let $first(i)$ and $last(i)$ be two indexes associated with each vertex i. A DFS is applied on the tree, starting from s. Then $first(i)$ stores the DFS vertex counter the first time i is visited and $last(i)$ gets the DFS vertex counter when the DFS leaves the subtree of i. Thus, vertex j belongs to the subtree of vertex i if and only if $first(j) \in [first(i), last(i)]$. Besides, the number of vertices in the subtree of i, i included, is $size(i) = last(i) - first(i) + 1$. This information can be used to check for special cases in *edge insertion* moves, if one extremity belongs to the subtree of the other one. The same idea can be used to check if a tree is feasible in the capacitated MST [18, 22]. A demand D_i is associated with each vertex i and the sum of the demands in any subtree must not exceed a capacity Q. A demand counter is used instead of a vertex counter in the DFS.

The *edge drop* move requires reconnecting two trees disconnected after an edge $[i, j]$ has been removed. Without loss of generality, suppose i is the predecessor of j, i.e., $i = pred(j)$. First, all the vertices are set as unvisited. Then, all vertices in the subtree of j (included) are set as visited by performing a DFS or a BFS, using the list of successors presented before. Getting the lowest cost edge reconnecting the two trees can be done in $O(m)$ by finding the smallest edge in E with one extremity visited and the other one unvisited. This last operation is the most expensive and the *edge drop* move is in $O(m)$.

As mentioned before, the depth of a vertex from the root in a tree can be used to check hop and diameter constraints. For the root vertex, $depth(s) = 0$. A simple DFS or BFS starting from s is sufficient to compute the depth of each vertex in $O(n)$. They may require an artificial vertex 0 playing the role of center, $depth(0) = 0$. The artificial vertex connects the central vertex whenever D is even and one of the extremities for a central edge if D is odd. Let $L = D/2$ resp. $L = (D-1)/2$ be the maximum depth allowed for a node, respectively, when D is even resp. odd. Thus, whenever an artificial vertex is used, a diameter limit D on the tree corresponds to a depth limit $L + 1$ for each vertex from vertex 0. Checking the degree is simpler; one may just add the number of successors and predecessors for each vertex.

Encodings for Trees

The tree representation given in the section "Data Structures for Trees" can be used for most constructive heuristics and for moves in local searches and metaheuristic algorithms. However, evolutionary algorithms (EAs) usually require additional properties in the representation, and several other tree encodings have

been proposed. In EAs, operations are mostly done on encodings (the chromosomes) rather than on the solutions. These usually consist of crossovers and mutations in order to make the set of encodings (the population) evolve. Selection is done to obtain the best elements. It requires the evaluation of each element through the construction of the associated solution, named decoding.

Several properties of the encoding/decoding, i.e., the mapping between encodings and solutions, are needed to fully benefit from the design of EAs: (i) the time and space complexities for those operations should be as low as possible, (ii) any encoding should correspond to a feasible solution, (iii) the decoding should be unbiased, (iv) there should be at least one encoding leading to each optimal solution, (v) any offspring obtained from crossover and mutation should be valid, and (vi) the offsprings should share as much similarities as possible with their parents.

According to those criteria, the predecessor structure presented before offers a $O(n)$ complexity for encoding/decoding. However, a random vector of predecessors is very unlikely to be feasible. Classical crossovers and mutations also generate infeasible offsprings and a repair operator is required. On the other hand, there always exists one encoding for each optimal solution and the feasible offsprings inherit a lot of similarities from their parents.

Another classical representation is the Boolean edge-indexed vector. The status of each edge (used/unused) is stored in the solution. Thus it follows the binary encoding paradigm suggested in the early versions of genetic algorithms (GAs). However, a random vector is highly unlikely to be feasible and crossovers and mutations seldom produce feasible offsprings. Thus the repair operator is mandatory, unless defining dedicated crossovers and mutations.

Prüfer sequences have also been used to encode trees. They were introduced in [45] to prove the Cayley's theorem [12] and there is a bijection between the set of spanning trees and the set of Prüfer sequences. They are basically repetition vectors of size $n-2$ in which each vertex i appears $degree(i)-1$ times. Encoding and decoding can be done in $O(n \log n)$. There is no need for repair operator since every sequence corresponds to a feasible spanning tree. However, while appealing, this representation suffers from several drawbacks [23], especially with respect to property (vi) since a small change in the sequence might lead to a complete different solution.

Random keys [8] have also been used and are quite popular. A random key is a weight in $[0, 1]$. For trees, one random key is associated with each edge, leading to a real-valued vector of size m. The decoding first consists of sorting the edges according to their random key and then applying a Kruskal-based constructive heuristic. Thus, it is done in $O(m \log m)$. It always leads to a feasible spanning tree and does not require a repair operator.

All the encodings suffer from shortcomings with respect to the properties (i)–(vi) mentioned before [46]. Moreover, they may not be able to handle sparse graph or additional constraints on the tree structure. Aside from random keys, a good alternative seems to be edge-set encoding [46]: the edges of the tree are directly stored in a variable-length vector. Thus, this is an explicit variable-size encoding whose encoding/decoding can be done in $O(n)$. It is less biased toward special

trees than random keys. It has been shown to be quite effective for solving NP-hard extensions of MST, provided dedicated crossovers and mutations are used.

A Lagrangian Heuristic for DCMST

The use of Lagrangian relaxation in heuristic procedures has been proved to be a quite powerful tool for solving certain problems, especially DCMST. A Lagrangian heuristic (LH) involves a small number of ingredients that can be implemented efficiently: (i) determining Lagrange multipliers and evaluating Lagrange subproblems, (ii) obtaining feasible solutions from the Lagrange multipliers, and (iii) improving feasible solutions by a local search. The main advantages of an LH are that it is simple to implement, solutions of good quality can be obtained in a reasonable computational time, and lower bounds (LB) and upper bounds (UB) are available at each iteration. In addition, an LH can be used to obtain starting incumbent solutions in more complex exact algorithms.

Let $G = (V, E)$ be a graph defined as before. Costs $c_e \in \mathbb{R}^+$ are associated with each edge $e \in E$, and a maximum degree $d_v \in \mathbb{N}^*$ is associated with each node $v \in V$. The DCMST problem consists in finding a minimal cost spanning tree $\mathcal{T}$ of G such that the degree constraints are ensured for each node.

Figure 5a illustrates an example of a graph G, in which edge costs are reported in the middle of each edge and the maximum node degree constraints for spanning trees of G are given in brackets near each node. An MST of G, with cost equal to 5, is given in Fig. 5b. One may note that this MST violates the degree constraint in node 1. A feasible solution for the degree-constrained spanning tree (DCST) for G is provided in Fig. 5c, with cost equal to 9.

Problem Formulation

In order to present a mathematical model for DCMST problem, let $E(S) \subseteq E$ be the set of edges with both extremities in $S \subseteq V$. Let $\delta(v) \subseteq E$ be the set of edges adjacent to $v \in V$. Binary variables x_e, for all $e \in E$, represent the characteristic vector for a spanning tree of G, where $x_e = 1$ if an edge e belongs to the solution,

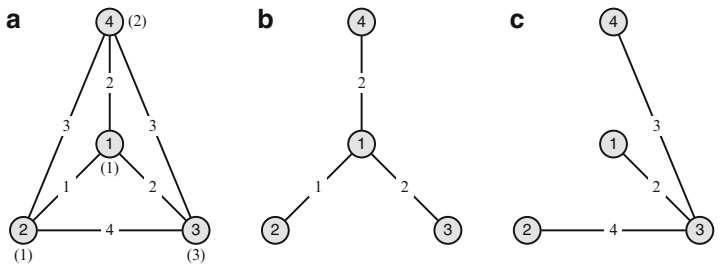


Fig. 5 Example of a tree with degree constraints. (**a**) A graph G. (**b**) MST of G. (**c**) A DCST of G

and $x_e = 0$ otherwise. The mathematical formulation is given from (1) to (4).

$$(P) \quad \min_{x \in \{0,1\}^m} \sum_{e \in E} c_e x_e \tag{1}$$

$$s.t. \sum_{e \in E} x_e = n - 1, \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad S \subset V, \tag{3}$$

$$\sum_{e \in \delta(v)} x_e \leq d_v, \quad \forall\, v \in V. \tag{4}$$

An MST is defined from (1) to (3) and constraints (4) control the degree of each vertex. These last constraints are the difficult ones. Thus they are relaxed in the LH presented next. Note that when $d_v = 2$, for all $v \in V$, the problem is reduced to finding a Hamiltonian path of minimum cost in G. Thus the DCMST is NP-hard [22].

The role of Lagrange penalties or multipliers is to take into account violations of some relaxed constraints. Given a tree $\mathcal{T}$ and for every node $v \in V$, the amount of node violation is the difference $\sum_{e \in \delta(v)} x_e - d_v$ between its maximum degree d_v and its degree in $\mathcal{T}$. In a DCST only nonpositive node violations are allowed. Positive node violation is penalized by associating a nonnegative penalty λ_i with each violated node i. In this case, every edge incident to i has its cost increased by λ_i. When both end nodes i and j of an edge $[i, j] \in E$ have positive penalties λ_i and λ_j, respectively, the resulting Lagrange cost becomes $c_{ij} + \lambda_i + \lambda_j$.

Consider the MST in Fig. 5b. It violates the degree constraint of node 1 (three edges are adjacent to this node and the maximum allowed is one). Figure 6 illustrates an example of MSTs computed with edge costs modified by Lagrange multipliers, where spanning trees of minimum cost are defined by solid lines in each graph. If a penalty $\lambda_1 = 10$ is set and $\lambda_i = 0$ for $i \in V \setminus \{1\}$ and an MST is computed for the graph with modified edge costs, the spanning tree in Fig. 6a is obtained. The degree constraint of node 2 is violated. Alternatively, if $\lambda_1 = 10$, $\lambda_2 = 2$, and $\lambda_i = 0$ for $i \in V \setminus \{1, 2\}$, an MST on the modified graph is in Fig. 6b. It is also feasible for the original graph G in Fig. 5a.

Thus, the idea is to determine the "best" set of Lagrange multipliers leading to an MST on the graph with modified edge costs whose Lagrangian solution value is equal to the value of an optimal DCMST. To do so, the Lagrangian problem (P_λ) is defined by associating Lagrange multipliers $\lambda \in \mathbb{R}^n_+$ with constraints (4) and bringing them to the objective function (1).

$$(P_\lambda) \quad \min_{x \in \{0,1\}^m} \sum_{[i,j] \in E} (c_{ij} + \lambda_i + \lambda_j) x_{ij} - \sum_{i \in V} \lambda_i d_i \tag{5}$$

$$s.t. \quad (2)\text{–}(3).$$

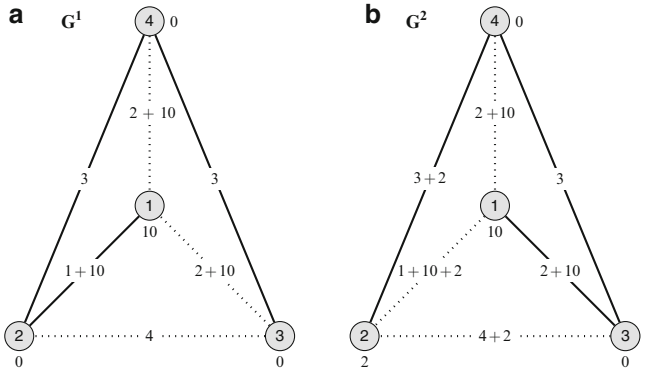


Fig. 6 Example of graphs with edge cost modified by Lagrange multipliers. (**a**) MST with λ_1=10. (**b**) MST with $\lambda_1 = 10, \lambda_2 = 2$

For any $\lambda \geq 0$, the solution value of (P_λ) is a lower bound on the optimal solution of (P). Consequently, the solution of the problem (D) is the best lower bound one can get for the solution of (P). Problem (D) is the Lagrangian dual problem of (P).

$$(D) \quad \max_{\lambda \geq 0} \min_{x \in \{0,1\}^m} \sum_{[i,j] \in E} (c_{ij} + \lambda_i + \lambda_j) x_{ij} - \sum_{i \in V} \lambda_i d_i \tag{6}$$

$$s.t. \quad (2)\text{–}(3).$$

The MST on the graph G^2 in Fig. 6b has a cost $(3 + 5 + 12) - (10 \times 1 + 2 \times 1) = 8$. This is a lower bound on the optimal DCMST value of the graph in Fig. 5a. It is optimal since that solution is feasible for G and its original cost in G is 8.

A Subgradient Procedure for DCMST Problem

The classical subgradient (SG) method of Held et al. [31] is used to compute Lagrange multipliers λ iteratively for problem (P_λ). At each iteration k of the method, the idea is to find a direction s^k and a step size t_k to move from λ^k to a new set of multipliers λ^{k+1}:

$$\lambda^{k+1} = \max\{0, \lambda^k + t_k s^k\} \tag{7}$$

s^k is the subgradient of (P_{λ^k}) with respect to the Lagrangian solution x^k. Its coordinates are given by

$$s_i^k = \sum_{e \in \delta(i)} x_e^k - d_i, \quad \forall\, i \in V \tag{8}$$

Determining the step size t_k requires an incumbent UB on the optimal solution value of (P), the Lagrangian value LB^k referred to x^k, and the norm of the direction s^k. It is defined as

$$t_k = \frac{\alpha(UB - LB^k)}{||s^k||^2} \tag{9}$$

where α is a scaling factor. Initially $\alpha = 2$ and it is reduced (usually $\alpha \leftarrow \alpha/2$) after some iterations with no improvement of the best LB. Thus, convergence of the SG algorithm is ensured. The number of iterations can be limited to a maximum value determined according to the characteristic of the problem instances being solved. If the Lagrangian solution is feasible for (P) and its Lagrangian value is equal to UB, it is optimal and the algorithm stops. Further details on the SG algorithm for DCMST problem can be found in [5].

As an example, the SG algorithm is applied to the graph G in Fig. 5a. Initially $\lambda^0 = 0$ and the resulting graph $G(\lambda^0)$ has the same edge costs as in G. The Lagrangian solution $\mathcal{T}^0$ for $G(\lambda^0)$ is the MST of G (Fig. 5b), and its Lagrangian cost is $\text{LB}^0 = 5$. The degree of nodes 1, 2, 3, and 4 in $\mathcal{T}^0$ are 3, 1, 1, and 1, respectively. The initial subgradient direction is $(s^0)^t = (2, 0, -2, -1)$. Figure 8 presents graphs $G(\lambda^k)$ of each SG iteration k. The Lagrange edge costs are displayed near each edge and Lagrange multipliers near each node. Moreover, Lagrangian solutions $\mathcal{T}^k$ of $G(\lambda^k)$ are defined by solid lines. Lagrange multipliers are computed according to Fig. 5 as follows: suppose the incumbent $\text{UB} = 9$ is given by the DCST in Fig. 5c, as computed by heuristic in section "A Greedy Heuristic for DCST". The new set of Lagrange multipliers is $(\lambda^1)^t = (16/9, 0, 0, 0)$. The corresponding graph $G(\lambda^1)$ leads to the Lagrangian solution $\mathcal{T}^1$ in Fig. 7, in solid lines. The new direction s^1 is carried forward in Fig. 8, column $\mathcal{T}^1$. The SG algorithm iterates until iteration 4, where the Lagrangian solution $\mathcal{T}^4$ is a feasible DCST for G. The original cost of $\mathcal{T}^4$ in G is 8 and its Lagrangian cost is $\text{LB}^4 = 596/75$. As all edges of G have integer costs, the optimal solution value must be integer. Therefore, the optimal solution value UB^* is such that $\lceil 596/75 \rceil \leq \text{UB}^* \leq 8$. Thus $\mathcal{T}^4$ is optimal and the algorithm stops.

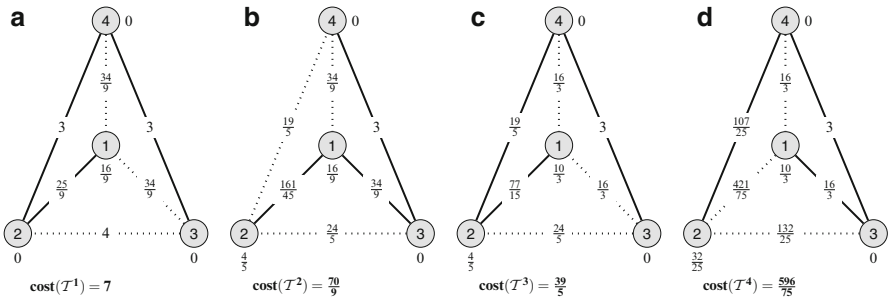


Fig. 7 Example of iterations for the SG method. (**a**) $\mathbf{G(\lambda^1)}$. (**b**) $\mathbf{G(\lambda^2)}$. (**c**) $\mathbf{G(\lambda^3)}$. (**d**) $\mathbf{G(\lambda^4)}$

$$\mathcal{T}^1$$

$$\lambda^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-5)}{9} \begin{pmatrix} 2 \\ 0 \\ -2 \\ -1 \end{pmatrix} \overset{\lambda \geq 0}{=} \begin{pmatrix} \frac{16}{9} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$s^1 = \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix}$$

$$\mathcal{T}^2$$

$$\lambda^2 = \begin{pmatrix} \frac{16}{9} \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-7)}{5} \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix} \overset{\lambda \geq 0}{=} \begin{pmatrix} \frac{16}{9} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix}$$

$$s^2 = \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \end{pmatrix}$$

$$\mathcal{T}^3$$

$$\lambda^3 = \begin{pmatrix} \frac{16}{9} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-\frac{70}{9})}{3} \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \end{pmatrix} \overset{\lambda \geq 0}{=} \begin{pmatrix} \frac{10}{3} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix}$$

$$s^3 = \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix}$$

$$\mathcal{T}^4$$

$$\lambda^4 = \begin{pmatrix} \frac{10}{3} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-\frac{39}{5})}{5} \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix} \overset{\lambda \geq 0}{=} \begin{pmatrix} \frac{10}{3} \\ \frac{32}{25} \\ 0 \\ 0 \end{pmatrix}$$

$$s^4 = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

Fig. 8 Updating Lagrange multipliers during the SG iterations shown in Fig. 7

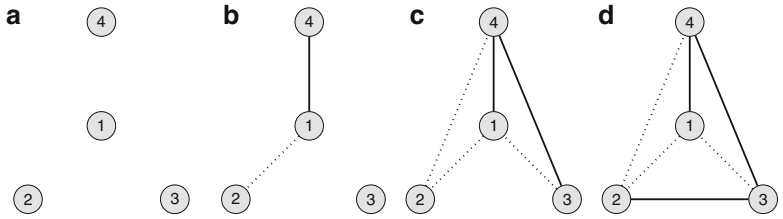


Fig. 9 Example of Kruskal-based heuristic for DCST. (**a**) Initial. (**b**) First edge. (**c**) Second edge. (**d**) Third edge

A Greedy Heuristic for DCST

Kruskal's algorithm can be adapted [5] to compute DCSTs on a graph $G = (V, E)$. It consists of (i) ordering the edges in E by nondecreasing edge costs; (ii) creating a forest with $|V|$ trivial trees $C_i = \{i\}$, for all $i \in V$; and (iii) connecting all disjoint trees (forests) by including one edge at a time, using the ordered list until a spanning tree is obtained. A vertex is saturated if its degree matches the maximum degree allowed. A tree is saturated if all its vertices are saturated. Then, an edge $[u, v] \in E$ is inserted in the solution if both u and v are not saturated and if they do not belong to the same tree. Moreover, merging the two trees using $[u, v]$ must not lead to a saturated tree, except if the resulting tree spans V.

Consider the graph G in Fig. 5a and assume that the ordered list of edges is $\{[1,2],[1,4],[1,3],[2,4],[3,4],[2,3]\}$. Initially, each vertex is a trivial tree. The first edge to be considered is $[1,2]$. It is discarded as its inclusion would result in a saturated tree. In the sequel, edge $[1,4]$ can be added since it does not saturate vertices 1 and 4 nor create a cycle. Node 1 is saturated; thus, the third edge $[1,3]$ is rejected. The fourth edge $[2,4]$ is also not accepted because it would result in a saturated tree. The fifth edge $[3,4]$ is included, *idem* for the sixth edge $[2,3]$. Then the algorithm stops with a feasible DCST. Note that this heuristic stops with a feasible solution for complete graphs only. Figure 9 presents this example with the ordered list of edges defined above. Solid lines correspond to edges included, while dotted lines correspond to discarded edges.

This heuristic can also be applied to graphs with modified Lagrange edge costs during the SG and compute its cost on the graph G. Nevertheless, doing this every SG iteration is time consuming. Finally, this heuristic can be used only when a Lagrangian solution improves the best incumbent LB on the optimal solution of (P).

Improving DCSTs with a Local Search

Given a graph $G = (V, E)$ and a spanning tree $\mathcal{T}$ of G, let $E(\mathcal{T})$ be the set of edges in $\mathcal{T}$. Moreover, consider a given subset $\{e_1, e_2, \cdots, e_k\} \subset E(\mathcal{T})$ of the edges belonging to $\mathcal{T}$. If there is a subset of edges $\{\hat{e}_1, \hat{e}_2, \cdots, \hat{e}_k\} \subset E \setminus E(\mathcal{T})$ such that $E(\mathcal{T}) \setminus \{e_1, e_2, \cdots, e_k\} \cup \{\hat{e}_1, \hat{e}_2, \cdots, \hat{e}_k\}$ induces a spanning tree $\hat{\mathcal{T}}$ of G, then $\hat{\mathcal{T}}$ is said to belong to a k-neighborhood of $\mathcal{T}$, denoted by $\hat{\mathcal{T}} \in N_k(\mathcal{T})$, and that $\mathcal{T} \in N_k(\hat{\mathcal{T}})$.

Consider $k+1$ connected components $C_1, C_2, \cdots, C_{k+1}$ obtained after removing k edges $\{e_1, e_2, \cdots, e_k\} \subset E(\mathcal{T})$ from a given spanning tree $\mathcal{T}$. If $\{\bar{e}_1, \bar{e}_2, \cdots, \bar{e}_k\} \subset E \setminus E(\mathcal{T})$ is a minimum cost subset of k edges to reconnect the $k+1$ components $C_1, C_2, \cdots, C_{k+1}$ forming a new spanning tree, then $\mathcal{T} \cup \{\bar{e}_1, \bar{e}_2, \cdots, \bar{e}_k\} \setminus \{e_1, e_2, \cdots, e_k\}$ is called a *k-opt edge exchange*. Such a move corresponds to applying k times an *edge drop* or an *edge insertion* move, as seen in section "Basic Features for Trees and Forests".

Figure 10 illustrates an example of a 2-*opt edge exchange*. Tree $\mathcal{T}^1$ in Fig. 10b is obtained from $\mathcal{T}$ in Fig. 10a by removing the edges $[1,4],[2,3]$ and adding the edges $[1,3],[2,4]$ while keeping the feasibility of the resulting tree.

Determining the best *k-opt edge exchange* move among all $\binom{|E(\mathcal{T})|}{k}$ combinations of k edges of $E(\mathcal{T})$ is very time consuming. A reasonable trade-off between solution quality and processing time is to limit k to 3. A greedy local search then iteratively performs *k-opt edge exchange* moves, until reaching a local optimum.

A Lagrangian Heuristic for DCMST

Using the components presented before, Algorithm 1 is a Lagrangian heuristic for DCMST problem. Given a graph $G = (V, E)$, a feasible starting DCST

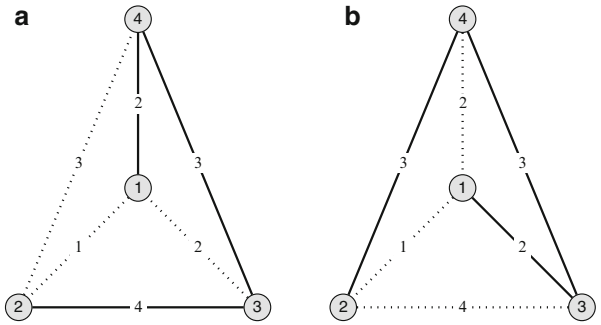


Fig. 10 Example of a 2-*opt edge exchange* move. (**a**) Tree $\mathcal{T}$. (**b**) Tree $\mathcal{T}^1$

$\mathcal{T}^*$ is computed by the heuristic in section "A Greedy Heuristic for DCST". A limit $MaxIter$ is set to the number of SG iterations. Basically, the subgradient method (section "A Subgradient Procedure for DCMST Problem") is used to update Lagrange multipliers until the stopping criterion (number of iterations or optimal solution) is met. At each iteration k, the classical Kruskal's algorithm [37] computes a Lagrangian solution $\mathcal{T}^k$ on the graph $G(\lambda^k)$: $\mathcal{T}^k \backslash MST(G(\lambda^k))$. The Lagrangian solution cost of $\mathcal{T}^k$ ($LagrCost(\mathcal{T}^k)$) gives a lower bound LB^k on the optimal solution value. If it improves LB, the new value LB^k is updated. Eventually, $\mathcal{T}^k$ is a DCST. In this case, a local search procedure (section "Improving DCSTs with a Local Search") is called to improve $\mathcal{T}^k$ if the cost $Cost(\mathcal{T}^k)$ of $\mathcal{T}^k$ in G improves UB: $\mathcal{T}^* \backslash LocalSearch(\mathcal{T}^k)$.

Algorithm 1: Lagrangian heuristic [5]

Data: a graph $G = (V, E)$, a DCST $\mathcal{T}^*$, a maximum number of iterations $MaxIter$;
Result: a DCST of G;
initialization: $k \leftarrow 0$; $\lambda^0 \leftarrow \mathbf{0}$; $T^0 \leftarrow MST(G(\lambda^0))$; LB$\leftarrow LagrCost(\mathcal{T}^0)$; UB$\leftarrow Cost(\mathcal{T}^*)$; **while** *(LB<UB* **and** $k < MaxIter$*)* **do**
  compute λ^{k+1} by using Equation (7);
  set $\mathcal{T}^k \leftarrow MST(G(\lambda^k))$ and $\text{LB}^k \leftarrow LagrCost(\mathcal{T}^k)$;
  if *(*$\text{LB}^k > LB$*)* **then**
    LB$\leftarrow \text{LB}^k$;
  end
  if *(*$\mathcal{T}^k$ *is a DCST* **and** $Cost(\mathcal{T}^k) < UB$*)* **then**
    $\mathcal{T}^* \leftarrow LocalSearch(\mathcal{T}^k)$;
    UB$\leftarrow Cost(\mathcal{T}^*)$;
  end
  $k \leftarrow k + 1$;
end
return $\mathcal{T}^*$;

As mentioned before, halving α after some SG iterations without any improvement on LB guarantees the convergence of the subgradient procedure. Also, one can work with a reduced instance $G' = (V, E')$ of the problem, by considering only a subset $E' \subset E$ of ordered edges required to obtain a feasible DCST for the problem, by using the modified Kruskal's algorithm from section "A Greedy Heuristic for DCST." Moreover, more sophisticated local searches and even metaheuristics can be used instead of the local search presented in section "Improving DCSTs with a Local Search". For instance, variable neighborhood search (VNS), dynamic variable neighborhood descent (VND), and absorption function [57] have been used in [4]. As the k-opt local search procedure is based on a greedy criterion, the Lagrangian heuristic may not escape local optima. To overcome this issue, the authors in [4] suggest the integration of a VNS and a dynamic VND, as well as absorption functions of [57].

The LH presented here has been applied to a number of benchmark instances from the literature: Euclidean And-inst, Hamiltonian Ham-inst, Shrd [36], M-graph and R-graph instances, and the new Dr-inst and De-inst instances [15]. In addition, the solutions obtained by LH have been used as cutoff to speed up exact methods [4, 15] for DCMST problems.

Results obtained by LH have provided new average gaps of up to 0.188 % and 6.230 % for And-inst and Ham-inst instances, respectively. Moreover, it has solved to optimality the Shrd, M-graph, and R-graph sets. After introducing the VNS, optimality was reached for 23 out of 25 And-inst instances, 4 out of 15 Ham-inst instances, and 8 out of 18 De-inst instances. Furthermore, optimal solutions were obtained for all Dr-inst instances. Thus, LH has been shown to be a powerful approach for DCMST problems.

Evolutionary Heuristic for a Generalization of BDMST

In this section, another way to handle difficult constraints is described for a generalization of the BDMST. The difficult constraints of the problem are defined on the paths of the tree rather than on the nodes as for DCMST. BDMST is defined in a connected and undirected graph $G = (V, E)$, where costs $c_{ij} \geq 0$ are associated with each edge $[i, j] \in E$. Consider $\mathcal{T}$ as a spanning tree of G. By MST definition, there is a unique path $\mathcal{P}_{ij}$ in $\mathcal{T}$ between any pair of nodes $i, j \in V$. Thus, given ρ_{ij} the number of edges in the path $\mathcal{P}_{ij}$, the diameter D of $\mathcal{T}$ is defined as $D = max\{\rho_{ij} : \forall i, j \in V, i \neq j\}$. The BDMST consists in finding an MST such that its diameter does not exceed a given $k \in \mathbb{N}^*$, i.e., $D \leq k$. This problem belongs to the NP-hard class whenever $3 < k < n - 1$ [22].

An example is shown in Fig. 11 for $k = 2$, considering the graph given in Fig. 11a. The MST is presented in Fig. 11b. It violates the diameter since $D = 3$. The optimal BDMST solution is provided in Fig. 11c, where the longest path has two edges ($D = 2$).

Several works are found in the literature for BDMST such as mathematical formulations [25,27,53], exact methods [26,43,44], and heuristics [40,47,49]. Yet,

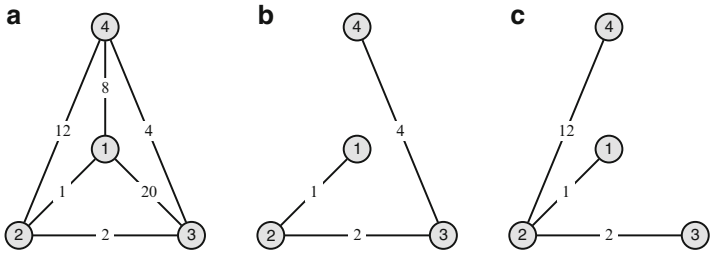


Fig. 11 Example of a tree with diameter constraints. (**a**) A graph G. (**b**) MST of G (**c**) A BDMST of G

optimality has not been proved for an important benchmark of instances proposed by [47], containing complete graphs whose size varies from 50 to 1000 vertices. Thus, an original strategy is to investigate the search space of this problem by using bi-objective approaches to compute the Pareto front. The idea is to drop the difficult constraints, i.e., diameter, and to consider them as a new criterion. Then a bi-objective optimization problem is solved. Thus the total tree cost and the diameter are minimized simultaneously. The problem remains difficult since the goal is to determine a Pareto front. However, the multiobjective strategy is interesting since every spanning tree is feasible and efficient algorithms are available to compute them. Moreover, in recent years, several advances have been done in solving multiobjective (bi-objective) problems using heuristics and metaheuristics. In addition, bi-objective heuristics do not depend on a good mathematical formulation. Another key point is that it is possible to bound the values for each criteria for problems relying on trees such as BDMST.

Bi-objective optimization for MSTs has already been investigated considering two cost objective functions. For example, a branch-and-bound was proposed in [56] and a two-phase enumeration was introduced in [48, 58]. Bi-objective metaheuristic algorithms are also used for two cost objective functions as in [6, 59]. A multiobjective evolutionary algorithm (MOEA) for the network design problem is presented in [59] to minimize the infrastructure cost and the maintenance cost, while a multiobjective greedy randomized adaptive search procedure (GRASP) is applied in [6] to find MST with two costs. Using this idea to handle difficult constraints seems to be more recent, in particular for the BDMST. Works [17, 50, 52] are dedicated to multiobjective strategies for the BDMST, referred as the bi-objective minimum diameter-cost spanning tree (bi-MDCST) problem. One of the most efficient multiobjective heuristics developed for bi-MDCST is the nondominated sorting genetic algorithm (NSGA-II), presented in the sequel.

Problem Definition

Without loss of generality, applying bi-objective heuristics for a problem $\mathcal{P}$ like the bi-MDCST implies the minimization of two objective functions $\{\min\ f_1(x),$

min $f_2(x) \mid x \in \mathcal{X}\}$. $\mathcal{X}$ is the feasible solution space of $\mathcal{P}$ and $f(x)$ is the vector of objective functions. For the bi-MDCST, $\{f_1(x), f_2(x)\}$ denote the cost and the diameter, respectively. Then a set of compromise solutions with respect to the two objective functions has to be obtained instead of a single solution as for the classical BDMST.

The concept of dominance is used to define the set of compromise solutions, usually called Pareto front. Considering two bi-MDCST solutions x and y, x *dominates* y if and only if the following conditions hold:

$$\begin{cases} f_k(x) < f_k(y) & \exists k \in \{1,2\} \quad \text{and} \\ f_k(x) \leq f_k(y) & \forall k \in \{1,2\} \end{cases} \tag{10}$$

A Pareto solution x^* is optimal whenever it is not dominated by any solution in $\mathcal{X}$. The Pareto-optimal front is composed of the Pareto-optimal solutions (nondominated). The bi-MDCST seeks a set of Pareto-optimal spanning trees $\mathcal{T}$ of G where f_1 and f_2 are simultaneously minimized. Figure 12 presents three solutions for bi-MDCST, considering the graph of Fig. 11a. The cost and the diameter for $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are, respectively, $\{29, 2\}$, $\{14, 3\}$, and $\{24, 2\}$. Solution $\mathcal{T}_1$ is dominated by $\mathcal{T}_3$ due to the cost. However, solutions $\mathcal{T}_2$ and $\mathcal{T}_3$ are not dominated by each other since $\mathcal{T}_2$ has a smaller cost than $\mathcal{T}_3$, while $\mathcal{T}_3$ has a smaller diameter than $\mathcal{T}_2$. One may note that solutions $\mathcal{T}_2$ and $\mathcal{T}_3$ are not Pareto-optimal because they are, respectively, dominated by the MST depicted in Fig. 11b of value {7,3} and the solution shown in Fig. 11c of values {15,2}.

For bi-MDCST, an obvious LB on the cost can be obtained by computing an MST on the graph. The diameter of this solution also provides an UB on the diameter. In addition, for complete graphs, the star is also a trivial LB on the diameter. The minimal cost of a star in the graph can provide an UB for the cost. Spanning trees of diameters $D \in \{2, 3\}$ can be computed in polynomial time as shown in [52]. Those LB on the diameter may not necessarily exist on general graphs. However, checking if a graph contains at least one such spanning tree can be done in polynomial time.

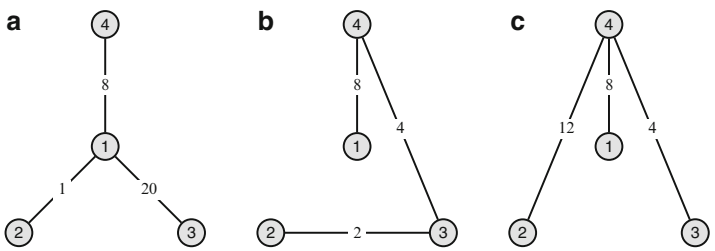


Fig. 12 Example of solutions for bi-MDCST. (**a**) Solution $\mathcal{T}_1$. (**b**) Solution $\mathcal{T}_2$ (**c**) Solution $\mathcal{T}_3$

An NSGA-II for Bi-MDCST

The NSGA-II has been proposed by [19] and is one of the metaheuristic algorithms available to compute Pareto fronts for multiobjective optimization problems. This method has been applied to a number of multiobjective optimization problems for the past ten years. Moreover, it uses simple and efficient operators to manage the Pareto front convergence. Such operators are added to a classical genetic algorithm (GA), which is very popular in the scientific literature. The two special multiobjective operators are the *ranking* and the *crowding distance*. These two operators are computed for each solution and are responsible to manage the Pareto front convergence. Both the *ranking* and the *crowding distance* are associated with each solution. Given a target solution $\mathcal{T}$, the *ranking* contains the number of solutions that dominate $\mathcal{T}$. Thus, the lower the ranking, the better the solution. Fronts are then defined by solutions of same ranking. Given a front of M solutions, the *crowding distance* w is computed following Equation (11), where $sc(j)$ and $pr(j)$ are, respectively, the value which precedes j in w and the value which follows j in w. For any (nondegenerate) front with more than one solution, $z_m^{\max} \neq z_m^{\min}$. The bi-MDCST involves two objectives $m = 1, 2$; thus, $z_m^{\max}$ and $z_m^{\min}$ are the maximum and the minimum objective function values, which need to be properly normalized. The crowding distance is a kind of Manhattan distance between two solutions from the same front. Then, the higher the crowding distance is, the more distant the two solutions are. As a consequence, solutions with larger crowding distance are preferred since they belong to less covered areas.

$$w_j = \sum_{m=1}^{M} \left(\frac{z_m^{sc(j)} - z_m^{pr(j)}}{z_m^{\max} - z_m^{\min}} \right) \tag{11}$$

The algorithm performs the following steps: (1) generate the initial population (set of solutions), (2) compute and order the population using the ranking and crowding distance, (3) select the first half elements of the population, and (4) generate the remaining elements using genetic operators and go to step (2). These steps are repeated until stopping criteria are met and are detailed below. In addition, a local search can be applied to each new solution in the population, i.e., before moving to step 2. In the vocabulary of GA, the use of a local search within a GA is referred as a memetic algorithm (MA). The idea of generating all $2n$ solutions and ordering them before cutting n solutions avoids cutting good solutions.

Given a graph $G = (V, E)$, a solution for bi-MDSCT (also referred as chromosome in the GA vocabulary) can be encoded using a vector of size n containing the direct predecessor of a vertex in the tree. A population can be generated using any heuristics (greedy, randomized, etc.) or even randomly generated spanning trees (step 1). The work [53] suggests an initial population of size $2n$ which contains (i) two specific solutions, an MST of G and a spanning tree of diameter $D = 2$ or $D = 3$, if it applies; (ii) half of the population obtained randomly; and (iii) the

remaining solutions computed using a randomized version of Prim's algorithm (i.e., the vertex entering the solution is randomly selected from a list of good candidates).

As mentioned above, the *ranking* and the *crowding distance* are computed for each solution in the initial population (step 2). Then, these operators will be responsible for the selection of solutions, following a multiobjective evolutionary approach rather than directly considering their cost and diameter. The comparison between two solutions $\mathcal{T}_1$ and $\mathcal{T}_2$ with respect to their *ranking* r_1 and r_2 and their *crowding distance* w_1 and w_2 is as follows:

$$(r_1 < r_2) \text{ or } ((r_1 = r_2) \text{ and } (w_1 > w_2)) \tag{12}$$

$\mathcal{T}_1$ is said to be better than $\mathcal{T}_2$ whenever condition (12) holds. In the following (step 2), the population is ordered. Now, the algorithm proceeds to the selection (step 3) by keeping the n best solutions for the next iteration. The n new solutions to enter the population can be generated by the crossover proposed by [11] (step 3). One advantage of this crossover is that feasibility of new solution is ensured, even for sparse graphs. The way to select the parents in the crossover can follow a classical elitist strategy, i.e., one parent from the elite set and the other one randomly selected from the remaining population. The crossover proposed by [11] works as follows: given two solutions (parents) $\mathcal{T}_1$ and $\mathcal{T}_2$, a support graph G' containing all edges from $\mathcal{T}_1$ and $\mathcal{T}_2$ is built. Then, an MST is computed on G' to obtain the new solution. Figure 13 illustrates this crossover, considering the graph G shown in Fig. 11a. The two parents $\mathcal{T}_1$ and $\mathcal{T}_2$ are, respectively, given in Fig. 13a and 13b. The resulting support graph G' is presented in Fig. 13c and the new individual is provided in Fig 13d, for which $Cost = 11$ and $D = 3$.

The local search iteratively performs a 2-opt move until no improvement can be done on the current solution. The 2-opt move uses the *edge drop* move previously presented. A specific feature of the 2-opt move for bi-MDCST suggested by [52] is that a move is accepted if and only if the diameter does not change and the cost is reduced. This means the local search will reach a local optima having the same diameter as the current solution.

The main insights obtained in [17, 52] by applying the NSGA-II for bi-MDSCT are summarized below. Three main sets of instances available in the literature for

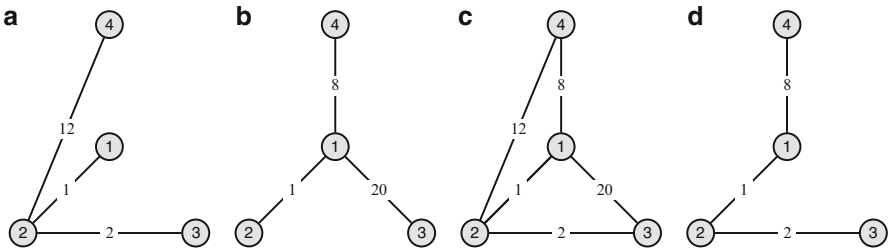


Fig. 13 Example of the crossover operator. (**a**) Solution $\mathcal{T}_1$. (**b**) Solution $\mathcal{T}_2$. (**c**) Graph G'. (**d**) MST of G'

the BDMST were used in the experiments. They were proposed by [25, 40, 47], respectively. In addition, two test sets were proposed by [52] containing 9 and 11 instances, named Hamiltonian cycle and Hamiltonian path, respectively. Results produced by NSGA-II were compared with available optimal values for the BDMST from [26, 53]. The NSGA-II for bi-MDCST manages to find some optimal diameter values of the BDMST and it is very close in the other cases. The computational time does not exceed the computational time spent by dedicated and sophisticated methods for the BDMST. The Pareto-optimal fronts for the test sets used in [40, 52] are published at the site http://di.uern.br/dario/bi-mdcst-problem/.

Some interesting characteristics have been observed: (i) a significant difference in tree cost occurs in the Pareto-optimal front between solutions with $D = 2$ and $D = 3$; (ii) since the instances from [40] have edges with similar cost, there are several MSTs with different diameters; (iii) some diameters are not interesting because the best corresponding solutions are dominated; and (iv) for the majority of instances from the sets mentioned above, up to 15 target diameters exist. The multiobjective approach has shown to be effective. It works consistently well and can provide useful information about the search space and the Pareto front for bi-MDCST problems.

Conclusion

Trees and forests are very rich topics, both in terms of theoretical and practical issues. Integrating good ingredients in sophisticated heuristics is the first step to produce good, competitive heuristics and even find new better results for NP-hard problems relying on these structures. In this context, some basic components for trees and forests are summarized in this chapter and two ways of addressing difficult constraints (degree and diameter) are presented, with the hope it will inspire new and fruitful research on these topics.

In terms of constructive heuristics, two basic ways have been addressed in the literature: tree expansion and tree fusion. Concerning the improvement heuristics, two main classes appear: restoration and local search. Some of these strategies use tree properties such as the local searches based on k-opt moves. In fact, by definition of a tree, removing an edge from a tree results in two connected components and inserting an edge in a tree obviously creates a cycle. Thus even a 1-opt can be considered two ways, either by first removing an edge or by first adding an edge. Even if they basically are 1-opt moves, these may impact the way the search space is explored.

Some key points are also provided in order to represent a tree in terms of data structures. Encoding and decoding a tree can be done directly by means of predecessor vectors indexed on the vertex or edge sets. It can also be done indirectly by using random keys. Additional data structures are also required to handle difficult constraints such as degrees and diameter. The choice of data structures will mainly impact the algorithm complexity. As a consequence, careful choices can save running time.

Two different strategies to address difficult constraints have been presented. The first, more classical, relies on a Lagrangian relaxation in which the constraints are relaxed and integrated into the objective function using the Lagrangian penalties. The second way consists of dropping the constraints, considering them as a new criterion. The resulting bi-objective problem is then solved by multiobjective approaches. The former presents the advantages of being simple to implement; it obtains solutions of good quality in a reasonable computational time, and lower and upper bounds are available at each iteration. A drawback is that it strongly depends on the mathematical formulation and the chosen decomposition. The latter is far less investigated in the literature, in particular for addressing difficult constraints. It has the following advantages: it provides additional information about the search space, and the computational time to obtain the Pareto front is very close to the time spent applying sophisticated dedicated methods. Moreover, limits in the search space can be computed and dominance rules can speed up the inspection of the solutions. A drawback is that the problem remains difficult since it may involve solving several NP-hard problems. However, facing NP-hard problems from a multiobjective perspective remains an interesting direction for further research.

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Multi-objective Optimization
- ▸ Variable Neighborhood Descent
- ▸ Variable Neighborhood Search

References

1. Achuthan NR, Caccetta L, Caccetta PA, Geelen JF (1994) Computational methods for the diameter restricted minimum weight spanning tree problem. Australas J Comb 10:51–71
2. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows – theory, algorithms and applications. Prentice Hall, Upper Saddle River
3. Álvarez-Miranda E, Ljubić I, Toth P (2013) Exact approaches for solving robust prize-collecting Steiner tree problems. Eur J Ope Res 229(3):599–612
4. Andrade R, Freitas AT (2013) Disjunctive combinatorial branch in a subgradient tree algorithm for the DCMST problem with VNS-Lagrangian bounds. Electron Notes Discrete Math 41(0):5–12
5. Andrade R, Lucena A, Maculan N (2006) Using Lagrangian dual information to generate degree constrained spanning trees. Discrete Appl Math 154(5):703–717
6. Arroyo JEC, Vieira PS, Vianna DS (2008) A GRASP algorithm for the multi-criteria minimum spanning tree problem. Ann Oper Res 159:125–133
7. Barahona F, Anbil R (2000) The volume algorithm: producing primal solutions with the subgradient method. Math Program 87:385–399
8. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6(2):154–160
9. Beasley JE (1993) Lagrangean relaxation. In: Reeves C (ed) Modern heuristic techniques for combinatorial problems. Wiley, New York, pp 243–303

10. Camerini PM, Galbiati G, Maffioli F (1980) Complexity of spanning tree problems: part I. Eur J Oper Res 5(5):346–352
11. Carrano EG, Fonseca CM, Takahashi RHC, Pimenta LCA, Neto OM (2007) A preliminary comparison of tree encoding schemes for evolutionary algorithms. In: IEEE international conference on systems, man and cybernetics, ISIC, Montreal, pp 1969–1974
12. Cayley A (1889) A theorem on trees. Q J Pure Appl Math 23:376–378
13. Cerrone C, Cerulli R, Raiconi A (2014) Relations, models and a memetic approach for three degree-dependent spanning tree problems. Eur J Oper Res 232(3):442–453
14. Cormen TH, Leiserson CE, Rivest R, Stein C (2009) Introduction to algorithms, 3rd edn. The MIT Press, Cambridge
15. da Cunha AS, Lucena A (2007) Lower and upper bounds for the degree-constrained minimum spanning tree problem. Networks 50(1):55–66
16. da Cunha AS, Lucena A, Maculan N, Resende MGC (2009) A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. Discrete Appl Math 157(6):1198–1217
17. de Sousa EG, Santos AC, Aloise DJ (2015) An exact method for solving the bi-objective minimum diameter-cost spanning tree problem. RAIRO Oper Rech 49:143–160
18. de Souza MC, Duhamel C, Ribeiro CC (2003) A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In: Resende M, de Sousa J (eds) Metaheuristics: computer decision-making. Kluwer, Boston, pp 627–658
19. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
20. Duhamel C, Gouveia L, Moura P, Souza M (2008) Models and heuristics for a minimum arborescence problem. Networks 51(1):34–47
21. Fisher ML (1981) The Lagrangian relaxation method for solving integer programming problems. Manag Sci 27:1–18
22. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-Completeness. W.H. Freeman, New York
23. Gottlieb J, Julstrom BA, Raidl GR, Rothlauf F (2001) Prüfer numbers: a poor representation of spanning trees for evolutionary search. In: Spector L, Goodman ED et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 343–350
24. Gouveia L, Leitner M, Ljubić I (2012) On the hop constrained Steiner tree problem with multiple root nodes. In: Ridha Mahjoub A, Markakis V, Milis I, Paschos VangelisTh (eds) Combinatorial optimization. Volume 7422 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 201–212
25. Gouveia L, Magnanti TL (2003) Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. Networks 41:159–173
26. Gouveia L, Simonetti L, Uchoa E (2011) Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. Math Program 128:123–148
27. Gruber M, Raidl GR (2005) A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem. In: Hansen P, Mladenovic N, Pérez JAM, Batista BM, Moreno-Vega JM (eds) The 2nd international network optimization conference, Lisbon, vol 1, pp 178–185
28. Guignard M (2003) Lagrangean relaxation. In: Cerdá MAL, Jurado IG (eds) Trabajos de Investigación Operativa, vol 11, chapter2. Sociedad de Estadística e Investigatión Operativa, Madrid, pp 151–228
29. Held M, Karp RM (1970) The travelling-salesman problem and minimum spanning trees. Oper Res 18:1138–1162
30. Held M, Karp RM (1971) The travelling-salesman problem and minimum spanning trees: part II. Math Program 1:6–25
31. Held MH, Wolfe P, Crowder HD (1974) Validation of subgradient optimization. Math Program 6:62–88

32. Henschel R, Leal-Taixé L, Rosenhahn B (2014) Efficient multiple people tracking using minimum cost arborescences. In: Jiang X, Hornegger J, Koch R (eds) Pattern recognition lecture notes in computer science. Springer, Cham, pp 265–276
33. Ho J-M, Lee DT, Chang C-H, Wong K (1991) Minimum diameter spanning trees and related problems. SIAM J Comput 20(5):987–997
34. Hwang FK, Richards DS, Winter P (1992) The Steiner tree problem. Annals of discrete mathematics, vol 53. Elsevier, Amsterdam
35. Kasperski A, Zieliński P (2011) On the approximability of robust spanning tree problems. Theor Comput Sci 412(4–5):365–374
36. Krishnamoorthy M, Ernst AT, Sharaiha YM (2001) Comparison of algorithms for the degree constrained minimum spanning tree. J Heuristics 7(6):587–611
37. Kruskal JB (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Am Math Soci 7:48–50
38. Leitner M, Ljubic I, Sinnl M (2015) A computational study of exact approaches for the bi-objective prize-collecting Steiner tree problem. INFORMS J Comput 27(1):118–134
39. Li Y, Yu J, Tao D (2014) Genetic algorithm for spanning tree construction in P2P distributed interactive applications. Neurocomputing 140(0):185–192
40. Lucena A, Ribeiro C, Santos AC (2010) A hybrid heuristic for the diameter constrained minimum spanning tree problem. J Glob Optim 46:363–381
41. Magnanti TL, Wolsey LA (1995) Optimal trees. In: Monma CL, Ball MO, Magnanti TL, Nemhauser GL (eds) Network models. Volume 7 of handbooks in operations research and management science. Elsevier, Amsterda, pp 503–615
42. Martinez LC, da Cunha AS (2012) A parallel Lagrangian relaxation algorithm for the min-degree constrained minimum spanning tree problem. In: Mahjoub AR, Markakis V, Milis I, Paschos V (eds) Combinatorial optimization. Volume 7422 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 237–248
43. Noronha TF, Ribeiro CC, Santos AC (2010) Solving diameter-constrained minimum spanning tree problems by constraint programming. Int Trans Oper Res 17(5):653–665
44. Noronha TF, Santos AC, Ribeiro CC (2008) Constraint programming for the diameter constrained minimum spanning tree problem. Electron Notes Discrete Math 30:93–98
45. Prüfer H (1918) Neuer beweis eines satzes über permutationen. Archiv für Mathematik und Physik 27:141–144
46. Raidl GR, Julstrom BA (2003) Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans Evol Comput 7(3):225–239
47. Raidl GR, Julstrom BA (2003) Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In: Proceedings of the 18th ACM symposium on applied computing, Melbourne, pp 747–752
48. Ramos RM, Alonso S, Sicilia J, Gonzalez C (1998) The problem of the optimal biobjective spanning tree. Eur J Oper Res 111(3):617–628
49. Requejo C, Santos E (2009) Greedy heuristics for the diameter-constrained minimum spanning tree problem. J Math Sci 161:930–943
50. Saha S, Kumar R (2011) Bounded-diameter MST instances with hybridization of multi-objective ea. Int J Comput Appl 18(4):17–25
51. Santos AC, Duhamel C, Belisário LS, Guedes LM (2012) Strategies for designing energy-efficient clusters-based WSN topologies. J Heuristics 18(4):657–675
52. Santos AC, Lima DR, Aloise DJ (2014) Modeling and solving the bi-objective minimum diameter-cost spanning tree problem. J Glob Optim 60(2):195–216
53. Santos AC, Lucena A, Ribeiro CC (2004) Solving diameter constrained minimum spanning tree problem in dense graphs. Lect Notes Comput Sci 3059:458–467
54. Shangin RE, Pardalos PM (2014) Heuristics for minimum spanning k-tree problem. Procedia Comput Sci 31(0):1074–1083
55. Silva RMA, Silva DM, Resende MGC, Mateus GR, Gonçalves JF, Festa P (2014) An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. Optim Lett 8(4):1225–1243

56. Sourd F, Spanjaard O (2008) A multiobjective branch-and-bound framework: application to the biobjective spanning tree problem. INFORMS J Comput 20(3):472–484
57. Souza MC, Martins P (2008) Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem. Eur J Oper Res 191(3):677–690
58. Steiner S, Radzik T (2008) Computing all efficient solutions of the biobjective minimum spanning tree problem. Comput Oper Res 35(1):198–211
59. Zhou G, Gen M (1999) Genetic algorithm approach on multi-criteria minimum spanning tree problem. Eur J Oper Res 114:141–152

World's Best Universities and Personalized Rankings

47

Mario Inostroza-Ponta, Natalie Jane de Vries, and Pablo Moscato

Contents

Introduction 1336
Materials and Methods 1338
Construction of a Dominance Graph and Transitive Reduction 1340
Symbolic Regression Analysis to Predict Levels Using All Variables 1342
Results 1343
University Ranking Results 2007–2009 1343
Transitive Dominance Example Results 1347
Sensitivity Analysis of Mobility over the Period 2010–2012 1347
Dominance Graph Computation of 2007–2012 with Symbolic Regression Analysis 1350
The Case of the Alumni Attribute 1359
The Case of the Performance Per Capita Attribute (Size) 1361
Discussion 1363
A Transparent and User-Centric Ranking Method 1363
Implications for Research Policy Makers 1364
Overview of Existing Multi-objective Ranking Methods 1365
Conclusion 1366
Cross-References 1367
References 1367

Electronic supplementary material: The online version of this chapter (https://doi.org/10.1007/978-3-319-07124-4_60) contains supplementary material, which is available to authorized users.

M. Inostroza-Ponta (✉)
Departamento de Ingeniería Informática, Universidad de Santiago, Santiago, Chile
e-mail: mario.inostroza@usach.cl; mario.inostroza@gmail.com

N. J. de Vries and P. Moscato
School of Electrical Engineering and Computing, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, Australia
e-mail: natalie.devries@newcastle.edu.au; pablo.moscato@newcastle.edu.au

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_60

Abstract

This chapter presents a heuristic for a multi-objective ranking problem using a dataset of international interest as an example of its application, namely, the ranking of the world's top educational institutions. The problem of ranking academic institutions is a subject of keen interest for administrators, consumers, and research policy makers. From a mathematical perspective, the proposed heuristic addresses the need for more transparent models and associated methods related to the problem of identifying sound relative rankings of objects with multiple attributes. The low complexity of the method allows software implementations that scale well for thousands of objects as well as permitting reasonable visualization. It is shown that a simple and multi-objective-aware ranking system can easily be implemented, which naturally leads to intuitive research policies resulting from varying scenarios presented within. The only assumption that this method relies on is the ability to sort the candidate objects according to each given attribute. Thus the attributes could be numerical or ordinal in nature. This helps to avoid the selection of an ad hoc single score based on an arbitrary assignment of attributes' weights as other heuristics do. To illustrate the use of this proposed methodology, results are presented and obtained using the dataset on the ranking of world universities (of the years 2007–2012), by academic performance, published annually by ARWU.

Keywords

Analytics · Digital humanities · Pareto optimality · Ranking · Symbolic regression

Introduction

Ranking multi-attributed objects is ubiquitous in the twenty-first century. This is not an overstatement. Every time people use a web search engine and query it using a particular keyword or phrase, a ranked list of webpages appears on our screen. This ranking is automatic, in most cases deterministic, and is well defined in mathematical terms. Not only the speed but the quality of the ranking may decide the fate of the company that has created that search engine. Every time people read a newspaper, or watch television, the news stories provided have been edited, and the core subject has been ranked and selected. Today, our governments want to decide on spending based on the objective rankings of institutions, their quality, performance, and delivery of service.

Specifically, the ranking of educational institutions has received increasing interest from scholars, policy developers, and strategic decision-makers worldwide. Our society, regardless of nationality, enjoys (and in some cases needs) to rank tennis players, sport teams, health systems, supercomputers, economical performance, scientists' achievements, most and least "liveable" countries and cities, restaurants, waiting lists for surgery and transplantation [2], targets in

structure-based virtual screening of three-dimensional protein libraries [30], genes in microarray experiments [6, 32] or in association studies [58], athletic training education programs [63], pedestrian crash zones [47], stressfulness of joints and joint motions in ergonomics applications [29], movie stars, dental esthetics [8]), and even US presidents.

The academic ranking of world universities follows the general trend of globalization of the economy. Researchers agree that ranking global universities is attracting increasing attention as it is a topic of high interest to many different stakeholders; see, for instance, [7, 34, 39, 40], among others. If you look at universities as service providers and if their potential students are free to choose among a variety of them, it is natural to pay attention to rankings to help them select the best tailored to their interests. This has become of even greater importance in recent years due to the increased mobility of students, researchers, and staff [52]. Furthermore, strategic decision-making within institutions, and policy development at a larger scale, relies on the provision of useful information regarding an institution's quality, excellence, and global ranking position.

Needless to say, ranking global universities is a quest of great impact and social significance [59]; however, the methodologies proposed are naive at best. It could be said that the problem is inherently *ill posed*. First, outcomes of ranking activities should be user centric. A student who has to choose where to go for graduate school studies may be more interested in the quality of their specific disciplines that constitute the core set of knowledge she is looking for and may dismiss an institution that has Nobel Prize winners lecturing using chalk and a blackboard (see [20] for a discussion on how to define quality). Managers and strategic decision-makers need to have access to information about how their institution benchmarks against their "competitors" in order to successfully and optimally allocate their resources and funds. Furthermore, it is important to remember that different stakeholders have different expectations about quality [34] and different expectations of an educational institution in general. In essence, it is important to highlight that ranking systems should be developed allowing the various users of these systems to easily add or remove an attribute of the objects in order to suit their specific needs and interests.

The request of [24] *"...there is still a dearth of peer-reviewed scientific publications on international ranking methods. Raw data and several key details about the methodology still remain unavailable to public scrutiny."* [24] easily resonates with this study. In addition to this, it recognized that the trustworthiness of many ranking systems is sensitive to the conceptual framework (its indicators) as well as the modeling choices such as weighting of indicators [52]. In order to fill this methodological gap, the issue of academic rankings of universities is addressed as an illustrative test case. This chapter thus aims to differ from Ioannidis et al. who have well articulated their case for the importance of a common international consensus on the validity of the measured attributes for each institution. The methodology presented here moves away from the traditional comparison of one-dimensional lists between ranked objects.

In this chapter other metrics that can be computed are included as well as a very simple heuristic that, using raw data, allows users to select/deselect attributes to

better target their interests. This approach can "tailor" a ranking system according to the user's individual preferences. Therefore, this system naturally provides a user-centric experience, while it is still based on quantitative nonsubjective information. This said, while this chapter is focussed on the largest online dataset of universities' performance, its final aim is to exemplify on its use to address the multi-objective ranking problem from a general methodological standpoint. In this method, the issue of sensitivity due to arbitrarily assigning of weights to indicators is addressed and the problems that arise in doing so, rather than using raw data.

The chapter is organized as follows. Section "Materials and Methods" illustrates where the data was obtained from, a brief description of the data, and an outline of the methodology used. Section "Results" has several subsections in order to display the results. Results for the computation of the dominance graph for the period 2007–2009 are displayed in section "University Ranking Results 2007–2009" followed by results describing transitive dominance as well as the implementation of symbolic regression modeling (section "Transitive Dominance Example Results"), mobility of universities in ranking (section "Sensitivity Analysis of Mobility over the Period 2010–2012"), and case examples of various universities, and, finally, a brief discussion on the attribute *Alumni* is included in section "The Case of the Alumni Attribute". Section "Discussion" provides a discussion of the results and includes the limitations thereof and recommendations for possible future areas of research.

Materials and Methods

To illustrate the proposed analysis methods for ranking, and to ensure reproducibility and interpretability of the results, the online publicly available information provided by the Academic Ranking of World Universities (ARWU) is used in this study. This is an annual publication and the data that is used for their ranking is provided by the Institute for Higher Education, Shanghai Jiao Tong University, since 2003.

Table 1 Explanation of the institute for higher education, Shanghai Jiao Tong University ranking system. Note that the attribute "Size" is also referred to as "Per Capita Performance" (PCP) and the attribute "SCI" is also referred to as "PUB" in various articles and resources

Attribute	Weighting	Explanation
Alumni	10%	Number of alumni of the institution winning Nobel Prizes and Fields Medals
Award	20%	Number of staff of the institution winning Nobel Prize and Field Medals categories
HiCi	20%	Number of highly cited researchers in 21 broad subjects
N&S	20%	Number of articles published in Nature or Science
SCI	20%	Articles in Science Citation Index Expanded, Social Science Citation Index, and Arts and Humanities Citation Index
Size	10%	Academic performance with respect to the size of the institution

The ARWU from the Shanghai Jiao Tong University is based on the performance of universities according to six categories (attributes). These attributes are displayed in Table 1 with their respective weighting given by the Shanghai Jiao Tong University and their explanations. The six attributes are *Alumni*, *Award*, *HiCi*, *N&S*, *SCI* (or PUB), and *Size* (or PCP). It is important to note that for institutions specializing in humanities and the social sciences, such as the London School of Economics, in the ARWU ranking systems, the attribute of N&S is not considered, and weighting is distributed to other indicators [34]. This borderline between institutions, however, is somewhat arbitrary. Therefore, it has to be recognized that for some institutions only four attributes (with arbitrary weighting) remain for consideration and that is a natural aspect of its organization on a reduced number of fields. In each category a university will receive a value between 0 and 100. The university with the best performance in a particular category receives the top value of a 100, and the rest of the universities will receive a proportional value against this measure. Further details and explanation of these attributes and the scoring procedure can be found on the Shangai Jiao Tong University ARWU website (http://www.shanghairanking.com) or in Liu et al. [34].

The raw data of this ranking process for the years 2007–2012 is used. From the top 500 universities in each year, only those that are present in all years since 2007 are included. This means that in total, the scores of 444 universities are used in the method. With this information, three datasets for analysis are created; one for the years 2007–2009, another for 2010–2012, and a combined dataset including the attributes over the whole six-year period. The first two are an array U of 444 rows and 18 columns, and each value in the array represents the score that the institution has in each of the six categories in the three years. The combined array has, of course, 36 columns.

The purpose of this chapter is not to argue the accurateness of this information, but rather utilize this data for an application of the method proposed. The reader should recognize that the use of this dataset has been challenged in terms of its reproducibility by Razvan [50] and also recognize the work by Docampo [14] who also alert of some problems in the ARWU's processes. However, as highlighted by many authors, after recognition of these issues and also noting the criticisms revolving about the ARWU ranking, this contribution addresses two of them. In particular, in section "Results", the *Alumni* attribute and the attribute related to *Size* (PCP) deserve two separate experiments to understand their contribution. On this point, it can be argued that, to ensure transparency, an international standard on the universities reporting on performance on specific attributes should be made available and that it is best, for reproducibility, to work with raw data. This said, this information is taken *bona fide*, as correct and complete.

From this set of attributes, other types of ranking systems could create other attributes or "meta-features" by combining or multiplying multiple attributes of the dataset. Typically, aggregates should be avoided whenever possible, and normalizations (e.g., dividing total number of publications by the total number of full-time employees) can easily be derived should this be required or desired by the user. Having said this, these standards may not make the dataset used in this study fully satisfactory (aggregates exist, normalization by full-time employees equivalents are

not made, other problems have been identified in Ioannidis et al. [24]), but this dataset is one of the world's best, regarding its coverage, and serves to illustrate our proposed methodology.

Construction of a Dominance Graph and Transitive Reduction

The method proposed in this chapter uses concepts inspired by transitive dominance to build a directed acyclic graph (DAG) ($G(V, A)$). A university x dominates university y, if and only if, for each attribute i the values of the universities for this attribute satisify $x_i \geq y_i$, $\forall\, i : [1..m] \wedge \exists\, j \in [1..m] \,/\, x_j > y_j$. This means that the performance of university x is at least better or equal in all attributes than university y, with at least one attribute at having a higher value. This definition creates a DAG $G(V, A)$ where each vertex represents a university, and if there is an arc $a_{xy} \in A$, it means that university x dominates university y. Within this chapter it will be shown that this DAG naturally induces a hierarchy of universities, with universities at Level 1 being those that are *not* dominated by any other university in the set. Here it is implied that the notation $x_i \geq y_i$ is general in meaning and is valid for both numerical and ordinal attributes. It is assumed that the attributes' measures are selected in such a way that a higher value on them implies a higher standing. This is a strength of this method as it can deal with datasets of mixed attribute types.

The resulting graph of this procedure would be hard to visualize in two dimensions as there are 444 nodes and it is very dense. Therefore, in order to simplify the DAG, without losing the hierarchy created, a method similar to, but less restrictive than, *transitive reduction* [4] is used. As Bang-Jensen and Gutin [4] describe, a transitive reduction of a directed graph (digraph) D is a spanning subdigraph H of D with no *transitively irrelevant* arcs, and it is called *minimum equivalent subdigraph*. In transitive reduction, "reachability" between nodes is not affected as only superfluous arcs are removed [41]. The minimum equivalent subdigraph of an acyclic digraph can be found in polynomial time and is unique.

However, in this instance, another algorithm that reduces the number of arcs further is used so that it is less restrictive in maintaining the reachability condition. A simplified visualization of this procedure is presented in Fig. 1 and is mathematically outlined here:

1. Identify the set of all universities which are nondominated; this set will be called the set of "Level 1" universities;
2. Add a dummy vertex v_d, which would correspond to a hypothetical nondominated university by all universities in the set (the Level 0 or "dream university"), and draw an arc from v_d to each of the universities in Level 1 (Fig. 1b);
3. The level of a university (represented by a vertex v_i) is then equal to the number of arcs that need to be traversed in the longest directed path from the "dream university" (vertex v_d) to v_i;
4. For each arc $a_{xy} \in A$ if $|Level(x) - Level(y)| > 1$, then $A = A - \{a_{xy}\}$ (see, for instance, the bold arcs in Fig. 1c are deleted from the DAG Fig. 1d).

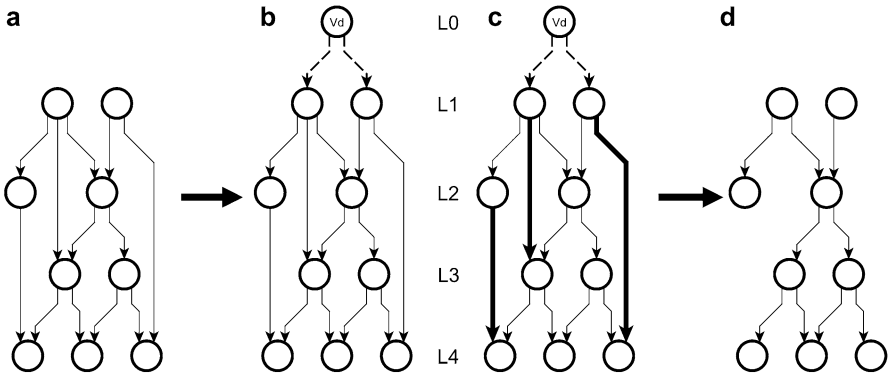


Fig. 1 Construction of the dominance graph. Each vertex represents a university, and there is an arc from each pair of universities (x, y) whenever x dominates y, according to the dominance definition given in the text. This method starts by first identifying the set of universities that are not dominated by any other university in the set. To illustrate on the algorithm, a set of figures are produced. Figure **(a)** shows the initial status. By assigning the nondominated set of universities to a top level, it would be possible to identify levels for all the other universities in the set. Figures **(b)**–**(d)** illustrate on the procedure to compute this and the resulting reduced graph as some arcs are removed. In **(b)** a dummy vertex v_d is added, which corresponds to a hypothetical "dream university" which is nondominated, which would be at "Level 0." From the dummy vertex to each of the nondominated universities, an arc is added (as shown in **(b)**). Then a level of a university is recursively defined as the maximum number of arcs that need to be transversed from v_d to it. Figure **(c)** shows arcs that connect universities for which the difference levels are more than one. These arcs are shown in boldface. Since visualizing such a dense directed graph would be unclear, these are removed as it would not affect the level of any given university. As a result, a reduced DAG is produced as shown in **(d)**, and through visualization it is shown how its results now uncover the underlying layer structure of **(a)**

The resulting graph from this process has the same hierarchical properties as the original one. Based on this final reduced graph, university x *"closely dominates"* university y if $a_{xy} \in A$ after the reduction.

This heuristic ranking methodology in terms of levels comes from a clear algorithmic procedure and is nonparametric, and it does not require an ad hoc definition of weights to be applied to each attribute. It is based on established grounds in multi-objective optimization and clearly reflects all pairwise comparable dominance relationships in the data. Furthermore, this method accepts numerical as well as categorical attributes allowing the inclusion of more variables in ranking activities.

Following the above process, in order to investigate "global" dominance relationships, the number of institutions that each university is transitively dominated by in the reduced graph will be calculated. This will provide an overall picture of where each institution is placed in comparison to all institutions included in the set. As an extra decision-making tool, it will provide institution leaders with the necessary information to compare and benchmark against other similar institutions. Examples of this are provided in section "Results".

Symbolic Regression Analysis to Predict Levels Using All Variables

In order to go one step further and investigate which variables are important in terms of "predicting" which level (rank) a university is going to be a part of, symbolic regression analysis is used in this chapter. By doing this, it clarifies whether the results found are consistent when using variables of each year for six consecutive iterations of the symbolic regression analysis. Symbolic regression is a data-driven method to find a structure in data. Unlike linear regression methods, symbolic regression not only finds the best values for a set of coefficients, it also finds the structure of the model that best fits the target variable [57] (including but not necessarily limited to linear models). In order to do this, a powerful software named *Eureqa* [54] is used which is based on evolutionary computation techniques to search for the best model. Eureqa runs an evolutionary search procedure to find the solution that fits the data best with the lowest possible level of complexity as is consistent with other symbolic regression methods [62].

In Eureqa, the "best models" are those that are observed in a Pareto optimality curve which imply a trade-off between their complexity and their fit. Complexity meaning the number of functions and "building blocks" used by *Eureqa* to fit a model to the target variable. The user selects a fitness function (for instance, absolute error or mean squared error) that guides Eureqa in selecting the best models. Users also have the option to select their preferred "building blocks" (for instance, arithmetic operations like multiplication, subtraction, and addition or the introduction of a constant). In a previous publication, the authors of Ref. [64] have shown the usefulness of symbolic regression and *Eureqa* in finding "functional constructs" from online consumer behavior data. Furthermore, the data-driven nature of symbolic regression is suitable to the context of this study as the aim is to make inferences on the data based on the information present in the data alone without a priori parameters or attribute weights.

In this study in particular, the aim is to find those variables that are of importance when "predicting" which level an institution is in as well as finding out whether the assignment to levels of all universities can be consistently predicted with variables from each of the years. To do this, the "level" is set as the target variable in *Eureqa*, while it tries to fit a model to predict between Levels 1–11. The level ranking dataset used for this procedure is the ranking obtain from running the dominance graph on the whole dataset, i.e., years 2007–2012. Following this, six searches are conducted in *Eureqa* in which every search uses only one year of data. The reason is to find out whether the models are consistent in predicting one outcome of the dominance graph using variables from several years. The variables in the dataset that are most frequently used in the models are highlighted, and these are identified as the key variables in predicting "level membership." Furthermore, the best simple linear function found by *Eureqa* is used for further analysis which is consistent with a previous publication using this software [64]. In this manner it is possible to inspect the variables that are found to predict the target variable, in this case, ranking level in detail.

In this study, the following "building blocks" were selected for the *Eureqa* analysis, the use of a "constant," "integer constant," "input variable," "addition," "subtraction," and "multiplication." For all of the six years, the six searches are run two times, once with the error metric as *R-squared goodness of fit* and once with *squared error Akaike information criterion* [AIC]. The *squared error* with AIC is selected as AIC provides a means for model selection. AIC deals with the *goodness of fit* of the model and the complexity of the model which is in line with the other selection criteria that are used based on the Pareto optimality curve.

Results

The results based on the construction of the directed dominance graphs are presented here. The total dataset for the years of 2007–2012 was analyzed and split into two datasets representing the periods 2007–2009 and 2010–2012. In Figs. 2 and 3 the hierarchical layout of the graph created for the three-year periods of 2007–2009 and 2010–2012, respectively, can be seen.

Resulting from the ranking procedure, as outlined in the methodology, the graph of 2007–2009 has 12 levels with a different number of universities in each level in the first period. In the second period, the graph has 13 levels, with Level 13 only containing one university (University of Jyvaskyla, Finland). Furthermore, colors are used to highlight the correlation with the Shanghai Jiao Tong University study. Red corresponds to those institutions that have been considered at the top 10 according to the Shanghai Jiao Tong University study (in the last year included) with the remaining top 50 in orange, and the remaining top 100 are colored in green. Also included in this section are the results of the calculation of transitive dominance of institutions, the symbolic regression analysis testing, the volatility of ranking over a longer period of time, and the removal of the attribute Alumni as well as the attribute PCP (*Size*) in the ranking method and the effects of the removal of each of these attributes on the outcome. In the representation of results, various institutions are highlighted as examples in order to communicate the findings of this study.

University Ranking Results 2007–2009

When looking at Level 1 in Fig. 2, some information that is clearly revealed by the method becomes apparent. The "Level 1" universities correspond to those which are not dominated by any other university in terms of the attributes considered in the method. In this graph, these are Harvard University and the California Institute of Technology (Caltech). While Harvard closely dominates several others, Caltech does not closely dominate any other institution of Level 2. Level 2 institutions are Stanford, Columbia University, the University of Tokyo, the University of California at Los Angeles, the University of Michigan at Ann Arbor, the University of Pennsylvania, the University of Washington, the University of California at

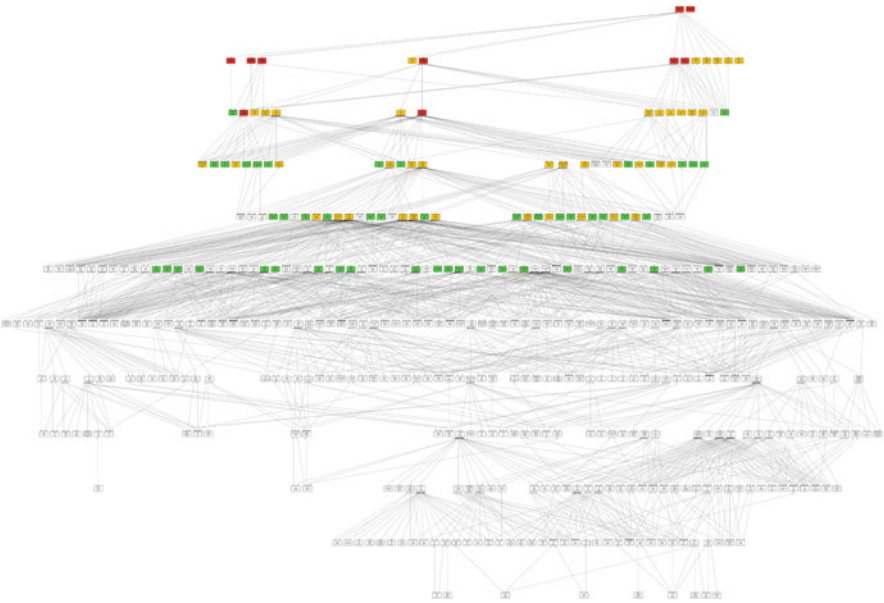


Fig. 2 The ranking of 444 world universities according to our proposed methodology using the same raw data provided by the Institute of Higher Education, Shanghai Jiao Tong University, for the years 2007, 2008, and 2009. Universities are arranged in *levels*, and a university is in a level if there exists another university that *closely dominates* it and is in the immediately higher level (the mathematical definition of this concept is outlined in the Methods Section). This graph provides important information for decision-making. In order to improve the overall ranking level, the university strategic decision-makers should pay attention to the best effective way of reaching the next level by improving in those attributes that would allow it to rank in the next level in the hierarchy. In addition, the university should also closely monitor the set of universities in the inferior level as they are the in direct competition for a position at the same level in the hierarchy. For instance, Yale University and Cornell University are two Level 3 universities that are both closely dominated by University of California at Berkeley. At the same time, these two institutions have a large number of institutions that they closely dominate at Level 4 (NB: A high-resolution version of this image can be found in the Supplementary material)

Berkeley, the University of Toronto, the University of Cambridge, Massachusetts Institute of Technology, and Princeton University.

The reason for Caltech's position in Level 1 can be explained not by Caltech's total productivity but rather its productivity regarding its size. Here it is noted that instead of normalizing raw data on the institution's performance by the number of full-time academics (or their equivalent full-time aggregated data), the Institute of Higher Education, Shanghai Jiao Tong University proposed the addition of an attribute called *"Academic performance with respect to the size of an institution"* or "Performance Per Capita" (PCP). It is important to remark that Caltech is a special case in terms of size. As pointed out by Baty in early 2014 [5], Caltech may be called not just "small," but *"tiny"* with its 300 professorial faculty, about 600 research scholars, and, at the last count, only 1,204 graduate students and just

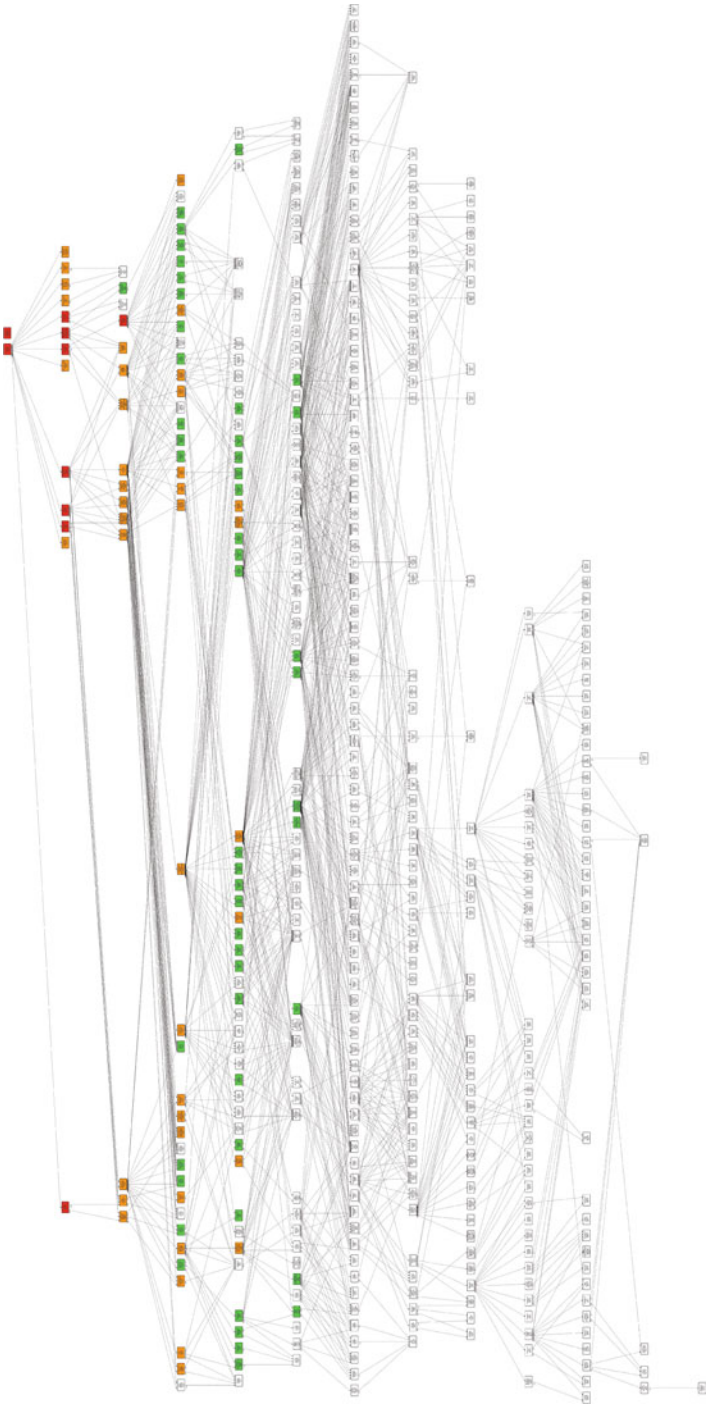


Fig. 3 (Continued)

977 undergraduates. This translates to an almost 3:1 student to faculty ratio. As a result of the introduction of the "Performance Per Capita" (PCP) attribute, which relates to the size of the institution in the ranking method, Caltech (with its "tiny" size) is "singled out" at the top due to its maximum value of this attribute without any close domination of Level 2 institutions. This reflects the consensus that the lack of normalization of the other attributes would bias ranking systems to benefit larger institutions. For instance, Rockefeller University is another small-sized institution that is very productive and is at Level 3, closely dominated by the Massachusetts Institute of Technology at Level 2.

Other Level 3 institutions include the Ecole Normale Superioure of Paris, the University of Chicago, Cornell University, Yale University, the University of Oxford, the University of California of San Diego, Kyoto University, Duke University, John Hopkins University, the University of Minnesota at Twin Cities, the University of Wisconsin at Madison, the University of Sao Paulo, and the University of Pittsburg. The University of Sao Paulo (Brazil) consolidates its position as a world Level 3 institution due to its relatively large SCI performance score.

At Level 4, the international mix of institutions is more clear. Level 4 includes the Hebrew University of Jerusalem (Israel), two German universities (the Technical University Munich and the University of Munich), three UK institutions (the Imperial College of Science, Technology, and Medicine, London School of Hygiene and Tropical Medicine, and the University College London), three from France (Pierre and Marie Curie University Paris, the University of Paris-Sud, and the University of Montpellier), the University of Oslo (Norway), the University of Basel (Switzerland), Moscow State University (Russia), the University of British Columbia (Canada), Osaka University (Japan), and a large group of USA-based universities, Carnegie Mellon University, the University of Colorado at Boulder, the

◄

Fig. 3 The ranking of 444 world universities according to our proposed methodology using the same raw data provided by the Institute of Higher Education, Shanghai Jiao Tong University, for the years 2010, 2011, and 2012. The California Institute of Technology and Harvard University continue to be the single institutions at Level 1. At Level 2, there is almost the same group of American institutions (and The Universities of Tokyo and Cambridge). However, the University of Pennsylvania now is at Level 3, and John Hopkins University and the University of Oxford now climb a level and now are at Level 2. If a single city of interest is to be highlighted, the universities in Hong Kong show a clear upward trend with the City University of Hong Kong climbing (climbing from Level 10 to Level 7), the Chinese University of Hong Kong and the University of Hong Kong (both raised from Level 8 to Level 6), and the Hong Kong Polytechnic University (from Level 10 to Level 8). Other Asian institutions raised three levels (Sun Yat-sen University and Hanyang University, from a Level 11 to Level 8). In Europe the remarkable uphill move of the Italy's Scuola Normale Superiore di Pisa that raised from Level 6 to Level 3 can be noted, as well as Sweden's Stockholm School of Economics (from Level 7 to Level 4) and Austria's Medical University of Innsbruck that from the bottom Level 12 is now at Level 7. However, over the three-year subsequent period, only 10% of the institutions raised two or more levels (and only 4.05% decreased it by a similar gap) indicating that as a metric is less volatile than other metrics to evaluate institutional performance (NB: A high-resolution version of this image can be found in the Supplementary material)

University of California San Francisco, Pennsylvania State University – University Park, the University of Illinois at Urbana-Champaign, the University of Texas Southwestern Medical Center at Dallas, Northwestern University, the University of California Davis, The Ohio State University Columbus, Case Western Reserve University, the University of Florida, Washington University in St. Louis, and the University of Rochester.

Lastly, the Ecole Polytechnique Federale de Lausanne (EPFL) Institute in Switzerland is also situated on Level 4. This institution is highlighted because Ioannidis et al. in [24] have pointed out how the EPFL of Lausanne was a missing institution in the top lists of Shanghai Jiao Tong University ranking system in previous years, yet the methodology in this chapter has the EPFL institution in a relatively high position. The EPFL is located at Level 4 as shown in Fig. 2. Furthermore, as is to be expected, as you move down the levels further, the number of institutions in each level increases, and the variety of origins becomes more apparent which can be examined in Fig. 2. A full presentation of institutions at all remaining levels can be found in Fig. 2.

Transitive Dominance Example Results

As stated, the number of institutions that each university is transitively dominated by and which ones it dominates has been computed, for instance, looking at the three UK-based institutions at Level 4, the Imperial College of Science, Technology, and Medicine, London School of Hygiene and Tropical Medicine, and the University College London. The Imperial College is transitively dominated by only seven universities in the world, while it transitively dominates 337 institutions. This is an impressive number for a Level 4 university, very close to the 346 institutions dominated by Caltech at Level 1. The University College London (UCL) numbers are equally impressive with only five institutions that transitively dominate it and with 377 transitively dominated by UCL. In contrast, the London School of Hygiene and Tropical Medicine is transitively dominated by only 12 institutions (consistent with the overall trend of expected dominance for an institution at this high level), but it does not dominate any other institution.

The same numbers of the three French institutions at Level 4 are Pierre and Marie Curie University Paris (9 and 209), the University of Paris-Sud (9 and 117), and the University of Montpellier 2 (13 and 45). This analysis provides some clarity on the peculiarities of some small-sized but intensively research-oriented institutions.

Sensitivity Analysis of Mobility over the Period 2010–2012

Research policy makers can intuitively evaluate the chances that a university can climb the level structure by way of improving in different attributes. A university can climb up one level only if it improves on those categories in which it is being closely dominated by other institutions (given that those institutions do not move

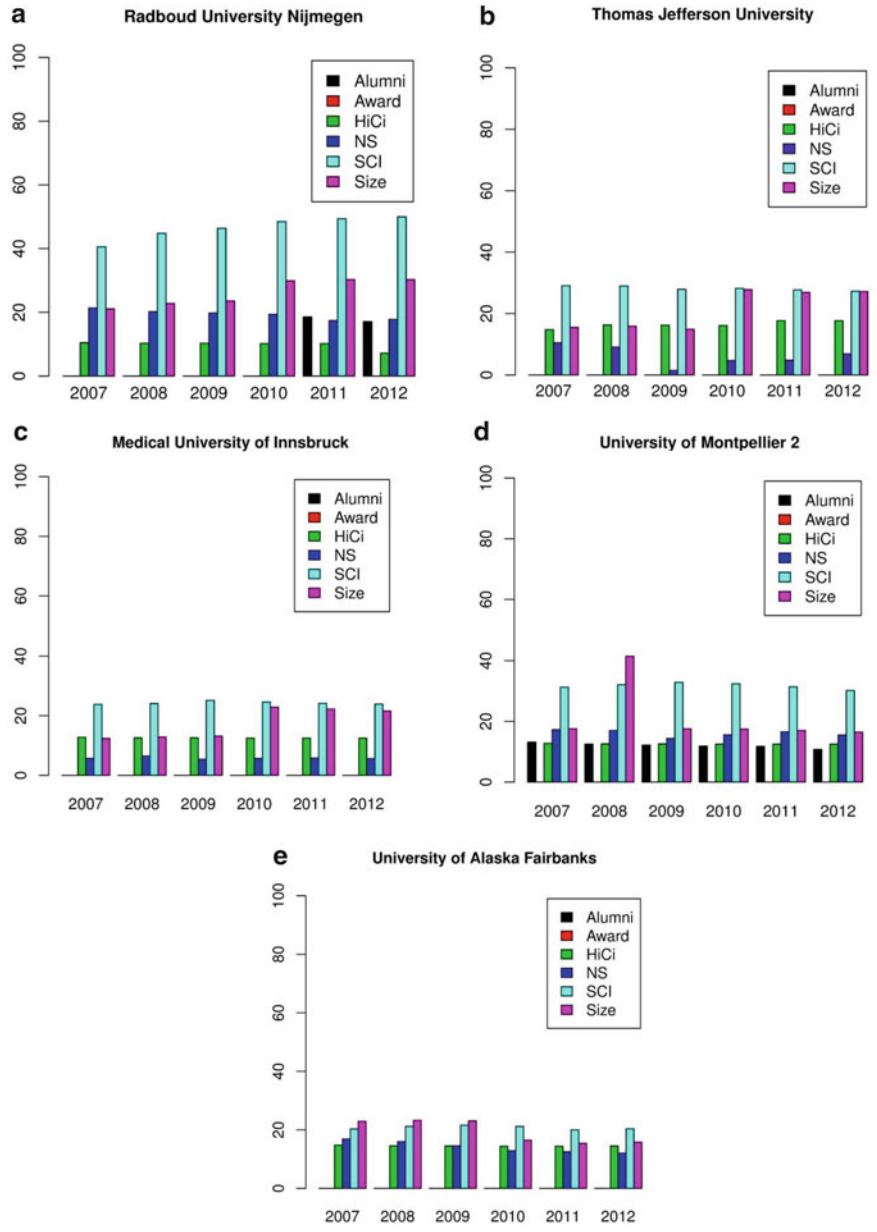


Fig. 4 (Continued)

themselves). The inbound degree and the "weight" of each arc implicitly convey how likely it is for a university to improve its level. It is intuitive that the lower the inbound degree and the lower the weights, the easier it is for a university to climb levels up in the overall ranking.

There is a clear linear correlation trend between changes in levels and changes in the transitive dominance of a given university. In order to inspect this, a comparison is made between the change in level position of the 2010–2012 dominance graph and the 2007–2009 dominance graph. Firstly, several universities are highlighted as examples of level changes, followed by universities' changes in level over these two periods when the attribute PCP is removed from the dominance graph computation. This will show those changes that occur based on only the other attributes that remain.

Figure 4 shows five universities that provide different examples of changes in levels. Firstly, in Fig. 4a, the Netherland's Radboud University Nijmegen is shown. In the Figure it is noticeable that the attribute "Alumni" is nonexistent in the years 2007–2010 and appears for the years 2011–2012. For the case of the Radboud University Nijmegen, this steep increase is due to the fact that in 2010, two researchers affiliated to the University obtained the Nobel Prize in Physics for their research in the properties of graphene [43]. Examining these types of changes further, results show that the Radboud University Nijmegen shows an impressive improvement (from 58 institutions dominating it to only 12). Next in Fig. 4b, the USA-based Thomas Jefferson University attributes are shown which explain its level improvement to Level 6 from a previous Level 10 position and has only 41 universities dominating it from a previous 162 total. The Figure shows that between the period 2007–2009 and 2010–2012, the Thomas Jefferson University increased significantly in the attribute related to size (PCP). The Medical University of Innsbruck that was at the lowest level in 2007–2009 (Level 12) jumped to Level 7 in 2010–2012 also experienced the largest change in transitive dominance (from 222 to 88) and is shown in Fig. 4c. This is a change of approximately 60% in the number of institutions that dominate it. In this figure it shows that the Medical University of Innsbruck has a steep increase in the value for the attribute related to size. As

◀

Fig. 4 Attribute values of universities that have changed levels over time – Figure **(a)** shows Radboud University which has climbed levels due to their increase in Alumni score. In figure **(b)**, Thomas Jefferson University provides an example of a university that has improved its level position due to an increase in Performance Per Capita (PCP), likely due to the institution changing in size. The Medical University of Innsbruck also provides an interesting example in figure **(c)** as it has the largest climb in level (from Level 12 to Level 7). It is possible to see for this university that all the categories are kept mostly the same, with the exception of category PCP, related to size. Figure **(d)** shows the university with the largest change in its level, University of Montpellier 2, which dropped from Level 4 to Level 8 when comparing the two periods that are investigated in this study. The profile of the university shows an unusual value for the PCP category in year 2008. This is probably due to an error in the data collected or published. Finally, in figure **(e)**, the University of Alaska Fairbanks drops from Level 7 to 10 which is explained by the drop in the category N&S, as well as the drop in PCP (Size)

stated, this attribute measures "Per Capita Performance" (PCP) in terms of research. Therefore, it could be that either the University reduced in size at this point in time or that it has significantly increased their research output with the same institutional size or even that some part of its research output was not previously included.

Furthermore, Fig. 4d shows the French University of Montpellier 2 and its peculiar values for the attribute "Size" (PCP). When comparing the two periods of time, the University of Montpellier 2 experiences a large drop in their level position from Level 4 to Level 8. In the figure it is evident that the University of Montpellier 2 had a steep decline in their score for the attribute "Size" (PCP). It may be argued that this could be due to an abnormality in one of the values of "PCP" (for the year 2008) which is conspicuously high. As a consequence, this may have affected the position of this university in the level for years 2010–2012. Finally, Fig. 4e shows the University of Alaska Fairbanks. This university is also an example of a university falling in its level position. The University of Alaska at Fairbanks is now at Level 10 from a previous position at Level 7. In the figure it can be seen that this drop is explained by the decrease in the attribute of N&S as well as a large decline in PCP.

Considering that many universities' drop in level can be attributed to their change in PCP, the results of the two periods without the PCP variable are compared. This will show those universities that still change in level position without considering their Performance Per Capita and may shed light on which other attributes are important to level position changes. Fig. 5 shows four universities which provide examples of this. In Fig. 5a, the UK-based University of Bath is shown. The University of Bath declined in its position over the years due to a decrease in the value for N&S attribute. The USA-based Northeastern University and the Australian University of Tasmania are shown in Fig. 5b, c, respectively, and show the reason for their level improvement. For both of these universities, their level improvement can be attributed to their increase in value for the N&S attribute in the new three-year period. This means that it is likely that each of these institutions shifted resources, and academics allocated their time differently in order to publish more in the journals Nature and Science in the second period (2010–2012) than in the first period (2007–2009). Lastly, Fig. 5d shows the City University of Hong Kong which experienced a drop in their position due to the decline in citations in N&S. Interestingly, although their value for "HiCi" improves in the newest three-year period, the drop in N&S is enough to shift this university's level position.

A complete list of the results of universities' mobility over the two time periods, for those universities that fell or increased more than two levels, is shown in Table 2.

Dominance Graph Computation of 2007–2012 with Symbolic Regression Analysis

After the periods 2007–2009 and 2010–2012 were examined separately, the results of the dominance graph for the whole period are examined. The resulting dominance graph is displayed in Fig. 6.

Examining the results of universities' position in the dominance graph over a total of 6 years allows the further investigation of the implications for institutions.

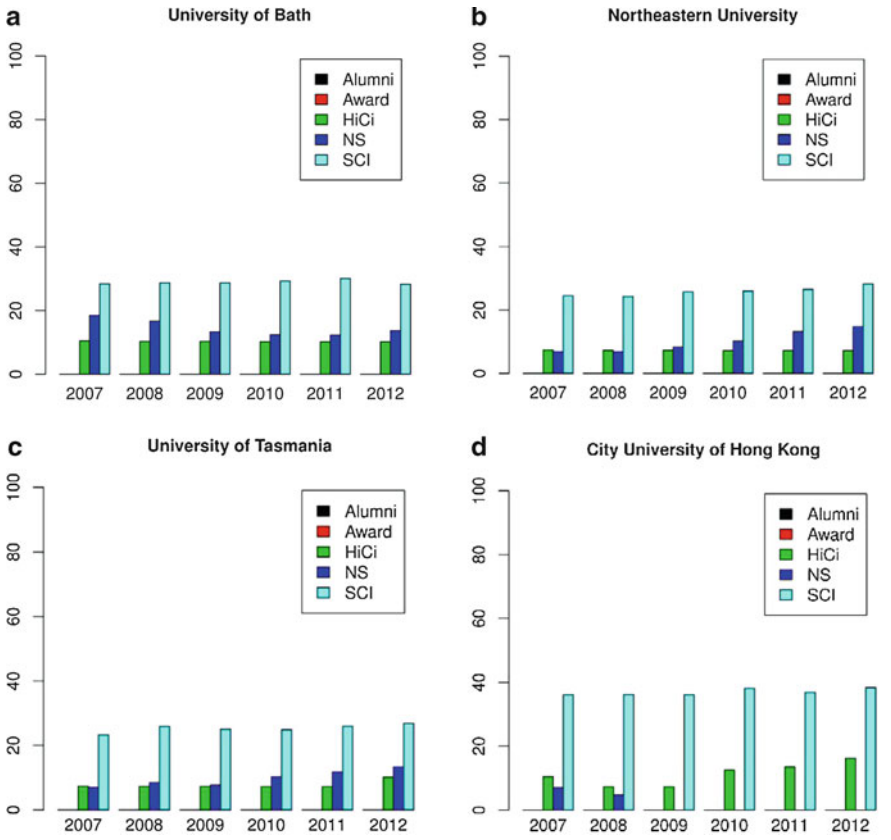


Fig. 5 Examples of universities that still moved levels in between the two periods of time when Performance Per Capita (PCP) was moved – This figure shows four universities that dropped or improved in their level position in the dominance graphs of 2007–2009 and 2010–2012 without PCP included. Figure **(a)** shows the University of Bath, which drops two levels between 2007–2009 and 2010–2012 due to their decrease in the value for the attribute N&S. Figure **(b)** shows Northeastern University which is the university that shows the largest improvement in level position in this comparison. As can be seen in this figure, their improvement can be attributed to their high values for N&S and SCI. Next, in figure **(c)**, the University of Tasmania shows that their rise from Level 16 to 13 is mainly due to their increase in the category N&S. Finally, figure **(d)**, the City University of Hong Kong jumps from Level 13 to 10 although they have a value of zero for the attribute N&S after 2009. The figure shows their level improvement is solely due to the increase in Highly Cited work HiCi

This is because this graph compiles all the scores for all attributes for each year. With more information available in one ranking outcome like this, more inferences can be made for institutions, strategic planning activities, and research policy making. As stated in section "Materials and Methods", symbolic regression analysis is used on the 2007–2012 ranking results to find those variables that are important in predicting a universities' rank as well as test for consistencies using data from

Table 2 Universities that change more than two levels in the Pareto dominance graph in the two periods of years considered

University	Level		Change
	2007–2009	2010–2012	
Medical University of Innsbruck	12	7	−5
Sao Paulo State University	11	7	−4
Thomas Jefferson University	10	6	−4
City University of Hong Kong	10	7	−3
Hanyang University	11	8	−3
Northeastern University	12	9	−3
Scuola Normale Superiore di Pisa	6	3	−3
Stockholm School of Economics	7	4	−3
Sun Yat-sen University	11	8	−3
University of Alaska Fairbanks	7	10	3
University of Montpellier 2	4	8	4

each year to predict the ranking of the 2007–2012 outcome. Using *Eureqa*, the aim is to find a linear function using the existing variables in the dataset to predict the level each university is a "member" of. Two separate error metrics are used to optimize the R^2 goodness of fit (GoF) and the squared error [AIC]. In each experiment the objective was to fit a model to predict the levels of the dominance graph shown in Fig. 6 using, in turn, the variables from each of the years 2007–2012. Results for the experiments optimizing R^2 GoF are shown in Table 3. Results optimizing the squared error [AIC] are shown in Table 4. As can be seen in these tables, all models contain the same variables, Alumni, PUB (research output), and PCP (related to size), for one exception which includes N&S (publications in Nature and Science) instead of PUB. However, it is noted that both PUB and N&S relate to a universities' research output and could therefore be seen as interchangeable as different institutions have higher research outputs in varying disciplines.

Furthermore, all models have highly similar coefficients. All constants at the start of each model range between 9.59 and 10.91 followed by highly similar coefficient attached to the Alumni attribute (ranged between 0.02 and 0.04). Following this, all models, except for that one using 2011 data and the Square Error [AIC], contain the attribute PUB with a coefficient ranging between 0.05 and 0.06. For the exception, N&S is included with a coefficient of also 0.05. This shows that for almost all models, PUB has a greater weighting in predicting universities' levels than the Alumni attribute does. Finally, for all models other than the 2011 model in Table 4, PCP is the last included attribute with the heaviest weighted coefficient between 0.09 and 0.10. Again, the exception is the model using squared error [AIC] using 2011 data which includes PUB as its last variable with a coefficient of 0.06 (which is similar to the coefficient attached to PUB for all other models). Besides this one exception, these models show a high level of

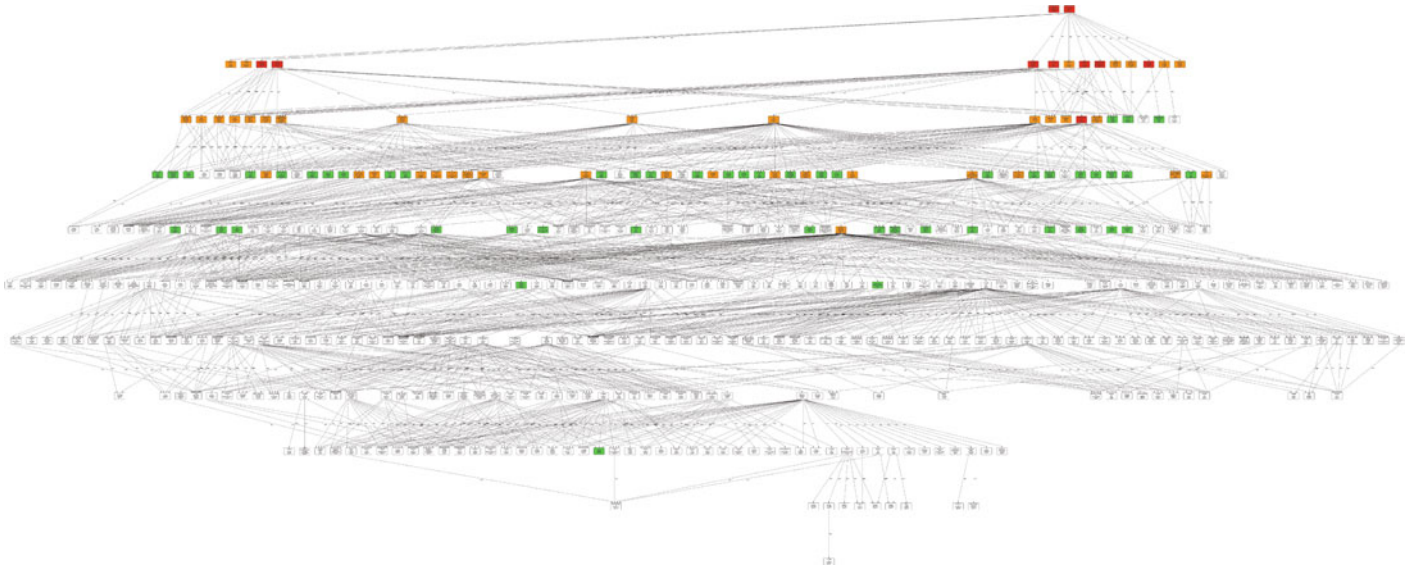


Fig. 6 Results of the dominance graph for the whole period 2007–2012. This graph contains a total of 11 levels, and consistent to the other results, Harvard University is still in Level 1 and directly dominates multiple universities in Level 2. The California Institute of Technology is again the only other institution on Level 1 but does not directly dominate and institutions in Level 2. Again, the coloring is consistent to the other dominance graph figures to highlight the ARWU ranking (NB: A high resolution quality of this image is available in Supplementary material)

Table 3 Results of the symbolic regression analysis predicting "level membership" or position. In this table the best linear models found using the R-squared goodness-of-fit measure are presented including the respective error metric. The experiment was run six times to include variables from each of the years separately to find consistencies in the variables used by *Eureqa*. As can be seen, all models are very similar; include the same three variables, Alumni, PUB (related to number of publications), and PCP (related to size); and even have almost identical coefficients

Year of variables used	Best linear model for level prediction	R^2 GoF
2007	Level = 10.40 − 0.02(Alumni) − 0.05(PUB) − 0.10(PCP)	0.71
2008	Level = 10.39 − 0.03(Alumni) − 0.05(PUB) − 0.09(PCP)	0.69
2009	Level = 10.41 − 0.02(Alumni) − 0.06(PUB) − 0.09(PCP)	0.71
2010	Level = 10.91 − 0.02(Alumni) − 0.06(PUB) − 0.10(PCP)	0.71
2011	Level = 10.90 − 0.02(Alumni) − 0.06(PUB) − 0.10(PCP)	0.71
2012	Level = 10.87 − 0.02(Alumni) − 0.06(PUB) − 0.10(PCP)	0.71

Table 4 Results of the symbolic regression analysis predicting "level membership." In this table the best linear models found using the squared error [AIC] measure are presented including the respective error metric. As in Table 3, the experiment was run six times to include variables from each of the years separately to find consistencies in the variables used by *Eureqa*

Year of variables used	Best linear model for level prediction	Mean squared error
2007	Level = 10.56 − 0.02(Alumni) − 0.05(PUB) − 0.10(PCP)	1.04
2008	Level = 10.40 − 0.03(Alumni) − 0.05(PUB) − 0.09(PCP)	1.11
2009	Level = 10.75 − 0.02(Alumni) − 0.06(PUB) − 0.10(PCP)	1.03
2010	Level = 10.87 − 0.03(Alumni) − 0.06(PUB) − 0.10(PCP)	1.04
2011	Level = 9.59 − 0.04(Alumni) − 0.05(N&S) − 0.06(PUB)	1.21
2012	Level = 10.76 − 0.03(Alumni) − 0.06(PUB) − 0.10(PCP)	1.05

consistency across which variables are used to predict "level membership" for universities. These results indicate that PCP (size) has a big impact on predicting which level a university is part of in the dominance graph of the ARWU dataset. Furthermore, it can be observed that PUB has a stronger impact on determining which level a university is on than Alumni and that these three attributes combined predict the levels of the 2007–2012 dominance graph to reasonable error values (as shown in Tables 3 and 4). Finally, these symbolic regression results could assist a university in determining strategies for improvement in the ARWU ranking as they provide the strongest impacting attributes on which level a university is on.

To further illustrate the usefulness of the proposed ranking method in this chapter in detail, various examples are outlined from the 2007–2012 dominance graph. The universities selected to provide case examples are the University of Copenhagen, the University of Amsterdam, and the Australian National University (ANU) as these institutions provide varying interesting cases.

Fig. 7 **The position of the University of Copenhagen at Level 4 in the dominance graph of the period 2007–2012**. The University of Copenhagen provides an interesting example as it is only directly dominated by one university in Level 3, Yale, and it dominates a total of 30 universities directly

Case Example of the University of Copenhagen

In the total period of 2007–2012, the University of Copenhagen was positioned on Level 4 of the dominance graph. As shown in Fig. 7, the University of Copenhagen is only closely dominated by one university, Yale, while it closely dominates 30 other institutions in Level 5. The fact that only Yale directly dominates the University of Copenhagen in our ranking result is a satisfactory achievement for the University of Copenhagen as this means the "inbound degree" of arcs for Copenhagen is low. However, it must be outlined that as the weight of the arc between Yale and Copenhagen equals to 36, it means that Yale dominates Copenhagen in every attribute for every year ($6 \times 6 = 36$). This shows that the weight of each arc is important in the dominance graph and provides information to the institutions wishing to directly benchmark with other institutions having very similar, though superior, profiles.

Case Example of the University of Amsterdam

In the total period of 2007–2012, the University of Amsterdam (UvA) was positioned on Level 5 of the dominance graph. It is one of the 30 universities that is closely dominated by the University of Copenhagen. Those universities that dominate the UvA and those that the UvA closely dominates are shown in Fig. 8 and create a cone-like shape that the UvA can analyze further for benchmarking purposes. As can be seen in the figure, the UvA is only closely dominated by Northwestern University, the University of Copenhagen, and the University of Illinois at Urbana-Champaign. The University of Amsterdam closely dominates nine universities at Level 6. Five of these universities are from Asia, Fudan University in China, Nanyang Technological University in Singapore, the University of Hong Kong, and the City University of Hong Kong. One such university is from Brazil, State University of Campinas; two universities from Europe, University of Erlangen-Nuremberg in Germany and the Swedish University of Agricultural Sciences; and, finally, one university from Israel, Yeshiva University.

Analyzing the relationship of UvA and the universities it is closely dominated by provides higher education decision-makers and research policy makers with further information about where to exert their resources more heavily. For instance, in the example of the UvA, it is shown how "close" UvA is to those universities that dominate it and possible areas of concentration for UvA to gradually improve its rank according to this system. For this, the scores of attributes only for the year 2012 are used as this is the most recent year in this dataset and therefore provide the most up-to-date answers. The attribute with the smallest difference in score for UvA and one of the universities is dominated by is "PCP" (Performance Per Capita). The UvA has a score in 2012 of 25.8, while the Northwestern University has a score of 26.4 making it only a 0.6 difference in score. This means that if the UvA improves in this item by just a mere 0.6 of a point score, it can change the domination status of one of the three institutions at a higher level that closely dominate it.

Furthermore, the University of Illinois at Urbana-Champaign has a score of 27.8 for PCP meaning that UvA is only 2% points lower. The University of Copenhagen

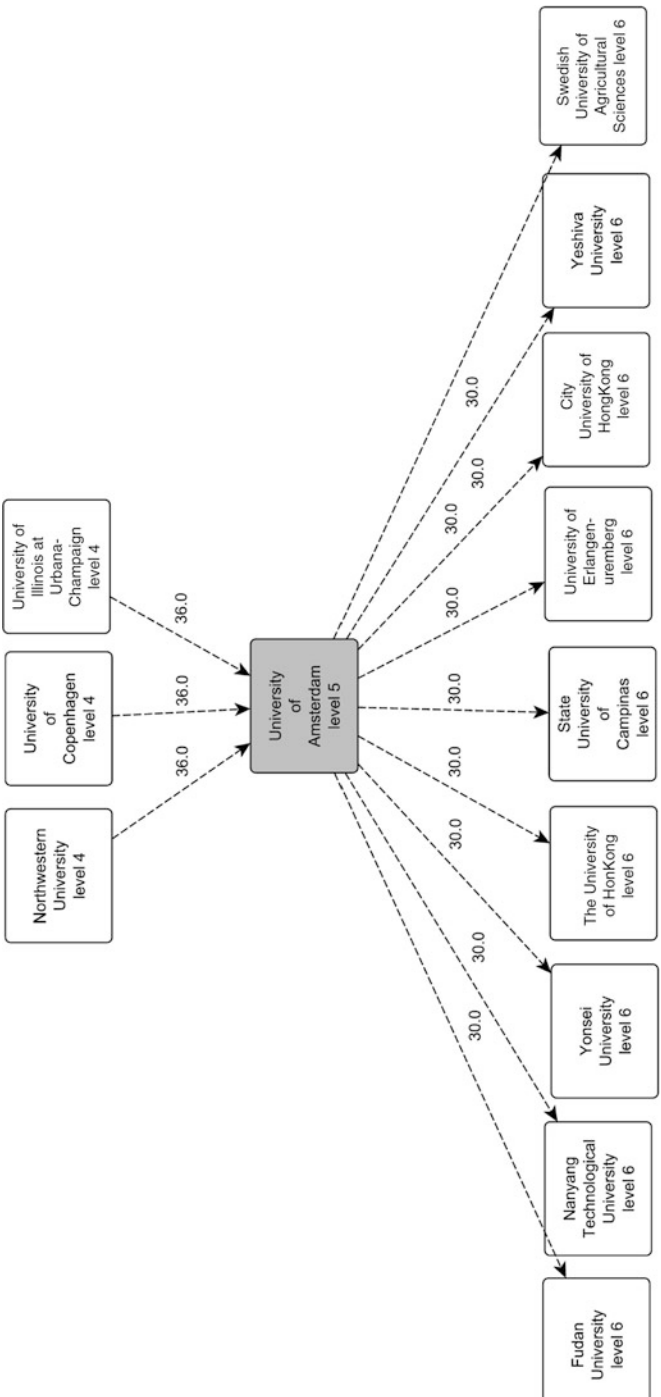


Fig. 8 The position of the University of Amsterdam at Level 5 in the dominance graph of the period 2007–2012. It shows that the University of Amsterdam is directly dominated by three universities at Level 4, Northwestern University, the University of Copenhagen, and the University of Illinois at Urbana-Champaign. In turn, the University of Amsterdam dominates nine other universities directly in Level 6 as shown in the figure

has a score of 32.3 for PCP making it 6.5 points higher than UvA. The second closest attribute score for UvA and its dominating universities is in "PUB." UvA's score in this attribute in 2012 is 52.8 while the University of Illinois at Urbana-Champaign has a score of 56.1 meaning there is only a 3.3 point difference. Northwestern University has a score of 58.1 (5.3 point difference with UvA) and the University of Copenhagen has a score of 58.6 (5.8 point difference with UvA). These results tell us that the UvA is closest to its dominating institutions in these attributes. This can inform management at UvA to focus attention and resources on increasing publications in certain journals and in doing so improve their Performance Per Capita (PCP) instead of investing heavily in other attributes in which the differences are pronounced.

Case Example of the Australian National University

In the analysis using the total period of 2007–2012, the Australian National University (ANU) was positioned on Level 4 of the dominance graph which is the same level as the University of Copenhagen (see Fig. 9). ANU is closely dominated by six universities at Level 3, while it, in turn, closely dominates five universities at Level 5. The universities that dominate ANU are all from the USA or the UK, University of California San Diego; Yale University; University College London;

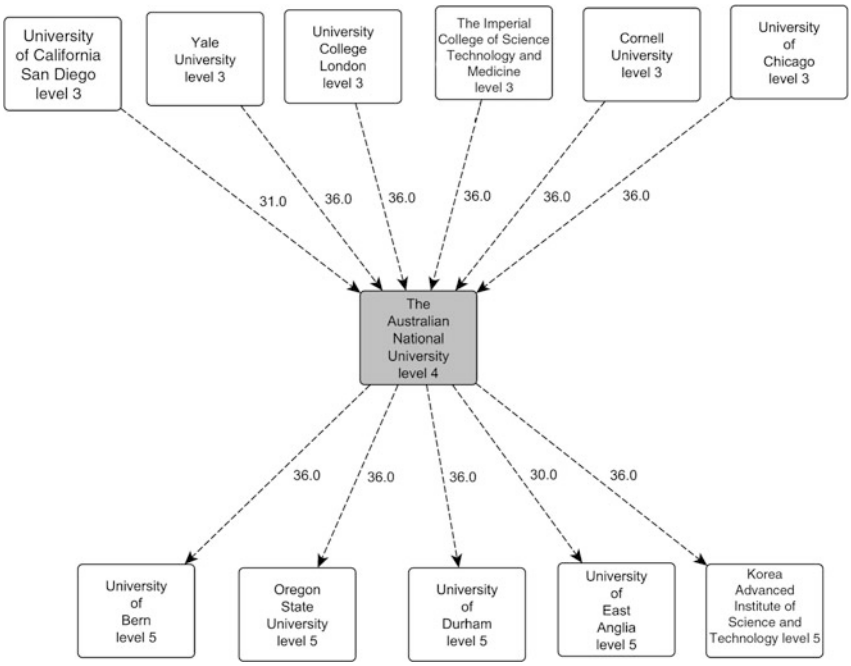


Fig. 9 The position of the National University of Australia at Level 4 in the dominance graph of the period 2007–2012. It is closely dominated by six universities at Level 3, while it, in turn, closely dominates five universities at Level 5

the Imperial College of Science, Technology, and Medicine; Cornell University; and the University of Chicago. The universities that ANU closely dominates are a more international mix, the University of Bern in Switzerland, Oregon State University from the USA, the University of Durham and the University of East Anglia in the UK, and Korea Advanced Institute of Science and Technology.

ANU is taken as an example of how to investigate the relationship with those universities a given institution closely dominates. In the case of ANU, there are five. In this scenario it is important to investigate which universities would be close in certain attributes or in some cases almost equal. Once the university of interest is no longer better in all attributes, it will no longer closely dominate those universities. Again, only the scores from 2012 will be investigated to provide a scenario as up to date as possible. In the case of ANU, three of the closest scores are in the attribute "Alumni" which relates to the alumni of the institution that obtain a Nobel Prize. In most cases, this is extremely difficult to alter, and a discussion on this attribute is provided in the following section. The next attributes in which universities closely "compete" with ANU are the attributes N&S and PCP.

The closest university in terms of the attribute N&S is the University of Bern followed by Oregon State University. These universities have scores less than 4 points below that of ANU making them competitors for ANU in this system. In terms of PCP, the Korea Advanced Institute of Science and Technology and the University of Bern again come close to ANU. The reason these attributes are outlined rather than Alumni is because these attributes are more in the control of the university than the number of Nobel Prize winners they "produce." For instance, ANU could encourage and enable its researchers to publish articles in Nature and Science rather than other journals in the field. Furthermore, when improving research output in terms of quality and volume without growing in size, it would be likely that Performance Per Capita would increase. These results show us the attributes that ANU needs to focus on in order to keep their competitive advantage and higher position in the dominance graph due to the universities they closely dominate.

The Case of the Alumni Attribute

In the ARWU ranking, the attribute measuring "Alumni," "the total number of the alumni of an institution winning Nobel Prizes and Fields Medals," has been extensively scrutinized and criticized for not measuring actual research or teaching excellence [25, 45]. As Ioannidis et al. [24] state, there is no doubt that research excellence *could* indirectly be measured by the Nobel and Field Medal faculty members working at a given institution. However, there is no justification to assume the existence of any clear correlation with the quality of teaching and research as a whole of the institution. Perhaps, it is only a measure of maturity, sheer size, number of graduates, financial support, and capital investment on a particular discipline (only certain disciplines can give Nobel Awards or Field Medals). In addition, it is unclear why an event with such a low probability of occurrence (very few

individuals receive Nobel Prizes or Field Medals) has such a large impact on the activities measured over populations of individuals. Furthermore, Ioannidis et al. have well argued that, in many cases, the affiliation of the Nobel and Field Medals differs at the time they actually conducted the award-winning work and the time that they were awarded the prize [24].

To prove that an attribute in the ranking process may be a spurious one, another type of sensitivity analysis is conducted as part of this chapter's proposed heuristic method. The attribute *Alumni* is eliminated and its effects on the ranking of the universities and their levels over the whole period 2007–2012 are subsequently investigated. It was previously observed that institutions such as the Netherland's Radboud University Nijmegen may have obtained dramatic changes in overall level position due to the attribution of Alumni alone (as can be seen in Fig. 4a shown by the black bar) which provides yet another reason for the comparison of these two outcomes.

According to the definition, the category Alumni represents *"the total number of the alumni of an institution winning Nobel Prizes and Fields Medals. Alumni are defined as those who obtain bachelor, Master's or doctoral degrees from the institution"*. In order to analyze the effect of this category in the ranking analysis, the dominance graph without the Alumni category is recalculated. Thus, each of the 444 universities performance is only observed in the remaining five scores in six consecutive years.

The resulting dominance graph on this dataset has 11 levels which is the same number of levels as the result of 2007–2012 period with all attributes included. This means that any changes in levels by a university is significant and provides useful information. However, here only those universities that increased or dropped by two or more levels are presented for the sake of brevity.

From the 444 universities, 196 universities (44.14%) do not change in level when the attribute Alumni is taken out of the dominance graph algorithm. There are 226 universities (50.9%) that change in only one level and only 21 universities (4.73%) that fall two or three levels. Furthermore, there is only one university (City University of New York City College) that falls four levels, indicating the robustness of this scheme. This points out the irrelevance of the Alumni attribute for the presented heuristic ranking method. This may please those opposed to Alumni being included as a proxy for measuring quality.

However, it is interesting to highlight how the performance of individual institutions may be affected by this choice. Looking at the City University of New York City College, without its impressive legacy of Nobel Prize winners, falls from Level 5 to Level 9. It was clearly in a higher level due to its high performance in the Alumni attribute (with a score of more than 37 in the three years), but the score in the rest of the categories is no higher than 18.

The three universities that fall more than three levels are Eotvos Lorand University in Hungary which fell from Level 7 to Level 10, University of Chile which fell from Level 8 to Level 11, and Saint Petersburg State University from Russia which fell from Level 6 to Level 9. It is clear that these institutions, without their successful score in the Alumni attribute, fall down to lower levels as their scores reflect much

Table 5 Universities that change more than two levels in the Pareto dominance graph in the period 2007–2012 when the attribute Alumni is not considered. There are another 226 universities that drop one level

University	Levels dropped
City University of New York City College	−4
Eotvos Lorand University	−3
University of Chile	−3
Saint Petersburg State University	−3
Case Western Reserve University	−2
University of Nebraska Lincoln	−2
University of Tuebingen	−2
Queen Mary U. of London	−2
Technical University of Berlin	−2
The University of Connecticut Storrs	−2
University of Cape Town	−2
Brigham Young University	−2
Ecole Polytechnique	−2
Technical University Darmstadt	−2
The University of Montana Missoula	−2
University of Oulu	−2
Moscow State University	−2
Complutense University of Madrid	−2
University of the Witwatersrand	−2
University of Warsaw	−2
Istanbul University	−2
Technical University of Braunschweig	−2

lower results in the other categories. It is also interesting to note that the Radboud University Nijmegen in the Netherlands, which, as pointed out, increased in levels over the two separated periods (2007–2009 and 2010–2012), actually falls from Level 5 to Level 6 over the whole period (2007–2012) when the attribute Alumni is not considered. This strengthens the argument that their climb in levels over the six-year period is merely due to their Alumni winning the Nobel prize in Physics rather than a total improvement in "academic excellence" supporting those who argue the superfluous nature of the Alumni attribute in measuring "academic excellence." The results of all universities which dropped or gained two or more levels with the removal of the Alumni attribute are shown in Table 5.

The Case of the Performance Per Capita Attribute (Size)

In the ARWU ranking, the only way in which the "size" of an institution is taken into account is through the only per capita measure: Performance Per Capita (PCP). In the ARWU ranking, PCP is defined by as "*The weighted scores of the above five indicators divided by the number of full-time equivalent academic staff. If the number of academic staff for institutions of a country cannot be obtained,*

the weighted scores of the above five indicators is used." [3]. Since this attribute is the only variable in ARWU's ranking method which takes an institution's size into account, it is an interesting image to see the effect of its removal on our methodology.

The sample of 444 universities over the total period of 2007–2012 has been used for the dominance graph algorithm in order to compare its outcome to the result of the 2007–2012 results for all attributes. There are more changes to the outcome when PCP is removed than when Alumni was removed. Firstly, the resulting dominance graph has 12 levels which means that some universities may change in levels due to a total reorganization of the graph. This means that all universities which changed more than two levels are significant; however, this presents a total of 124 universities; therefore, again for the sake of brevity, only those universities which changed in more than three levels are reported here. In these results, only 138 universities (31.08%) did not change a level, 182 universities (40.99%) changed in only one level, 84 universities (18.92%) changed 2 levels, and exactly 40 universities (9%) changed three or more levels.

In the case of deleting the PCP attribute from the method, the most extreme result is the Scuola Normale Superiore di Pisa in Italy, which drops an impressive 8 levels from Level 3 to Level 11. This incredible drop is likely due to its extremely small size in terms of staff and students. The Scuola Normale Superiore (SNS) has only approximately 150 undergraduate students, 120 postgraduates, and only 190 doctoral students as it only takes approximately 60 new students annually through its highly selective entry exam. However, it is important to note that the SNS in Italy is an entirely special case. In order to attend this institution, students sit a rigorous entry exam and must obtain certain levels of grades to remain enrolled in SNS. Furthermore, what makes SNS a special case is that its undergraduate students follow courses at the public university, the University of Pisa (Level 6 in the results without PCP), while living on the SNS campuses. This means that its research academics are able to devote their time to research and that they only have high-caliber students. This explains their impressive result in terms of Performance Per Capita and the incredible drop when PCP is removed from the method.

Furthermore, a small Swiss Institution the Swiss Federal Institute of Technology (EPFL) located in Lausanne has previously been highlighted. Another small Swiss institution the Swiss Federal Institute of Technology Zurich (ETHZ) is highlighted for a different reason. ETHZ was located on Level 3 in the result when all attributes were included in the method; however, with PCP removed, ETHZ fell to Level 4. The removal of the PCP attribute also had a negative effect for EPFL which fell from Level 4 to Level 7. This shows that for many small institutions, the attribute which takes Size into account is important for their position on the ranking levels.

The last small institution that is discussed here provides a particularly interesting case, Princeton University. Between the result for 2007–2012 with all attributes included and the result with PCP removed, Princeton remains on Level 2. In both results, Princeton University is only closely dominated by Harvard. The only change is in the universities Princeton closely dominates as it dominates the Scuola Normale Superiore di Pisa just discussed and Ecole Superieure Paris in the result of all

attributes but does not closely dominate any institution when PCP is removed from the method. This is because both these institutions drop in levels when PCP is removed leaving no Level 3 institution closely dominated by Princeton. From this result for Princeton University, it can stated that even though it is a small institution when compared to the likes of Harvard, it "holds its ground" even when the only *per capita* measure is removed unlike the other small institutions that have been discussed.

Discussion

This section elaborates on the findings of the proposed heuristic ranking method and reflects on the contribution of this chapter from a mathematical standpoint as well as a research policy making standpoint which is the application in this chapter to highlight the usefulness of this method. Furthermore, a brief overview of other existing multi-objective ranking methods is provided as a means for comparing the method presented in this chapter. This brief review is then used to highlight several benefits of this chapter's methodology as well as define any future research possibilities to improve the method. Firstly, the two main contributions of this chapter are described in further detail which are a mathematical contribution in terms of a transparent, user-centric heuristic ranking method through the use of a dominance graph and the provision of implications for decision-makers, managers, and research policy makers at educational institutions and governmental offices.

A Transparent and User-Centric Ranking Method

As stated, many authors argue that current ranking systems which include arbitrarily assigned weights and measurement approaches are generally naive, biased, flawed, or a combination of all.

This chapter has attempted to provide a different approach to the ranking of objects with multiple attributes through the use of the global universities ranking example. Needless to say, the methodology is applicable to other types of problems and would allow users of this method to naturally deal with attributes that are numerical and/or ordinal in nature. For example, scores or attributes that have ordinal values in categories could be taken into consideration for this method allowing a much wider adoption of this method for different ranking needs that include a variety of attributes or objects. The aim of discussing particular case examples was to provide an illustration of the user-centric approach of this method which allows alternative ranking outcomes to be investigated according to the user's interests and needs. It is for these reasons that the simple method used in this chapter is easily transferable to other situations.

One of the main benefits of the heuristic ranking method proposed here is that each user with differing interests can be satisfied by selecting which of the attributes must be included in the ranking. As it has been pointed out, the position

of several universities changed with the deletion of an attribute. This means that this system allows users to look at the world ranking of universities based on attributes that matter to them and exclude those that do not. Naturally, this would also apply to other ranking and ordering exercises in different situations where users may have varying interests. Such instances could include the ranking of medical institutions for possible medical treatments, companies or organizations for prospective employment, or even the analysis of countries or cities with attributes relating to "livability."

Furthermore, the approach to ranking of global universities in this study provides a more transparent method as it uses raw data rather than accumulated, normalized, or weighted data. As it is the weighting of the data in the ARWU ranking of world universities that concerns a large number of authors [25], it is argued that the proposed heuristic hierarchical method presented in this chapter satisfies a missing need for transparently ranking universities based on raw data available publicly. As it has been explained, although current university ranking methods seem to be flawed and subject to data inaccuracy, scholars and educational institution leaders agree that these activities are "here to stay" [52]. Hence, utilizing a methodology which is transparent and user centric may be more appropriate for the world "need" of global university rankings and other ranking activities.

Implications for Research Policy Makers

As stated before, well-known annual rankings of global universities, such as the ARWU, are often criticized due to their level of volatility [52]. The advantage of the methodology proposed in this chapter is that it organizes global universities in levels based on a dominance graph rather than a "string list." This means that the real nature of slow and steady university growth and improvement is reflected in this method rather than highly volatile changes. As such, this methodological approach correlates well with the subjective assumption that large institutions, like universities, when compared with other institutions of similar size and prestige, do not experience high levels of volatility in terms of their status over periods spanning just a few years. This level-based scheme induced by a dominance graph shows low volatility over consecutive periods of three years and therefore reflects the *real-life* nature of large educational institution growth.

The ability of an educational institution to compare and benchmark itself within the "ladders" of the hierarchy of the presented layers is what actually brings university leaders a clear path for improvement as outlined in the case of the University of Amsterdam in section "Case Example of the University of Amsterdam". Or conversely, a clear image of close competitors in levels below that institution can be examined, as highlighted with the example of the Australian National University in section "Case Example of the Australian National University". In these examples it was easily and clearly identified that these universities' nearest objectives (subset of the top layer that dominate that university) and those attributes they should consider to "protect their standing" (the ones a university closely dominates in the layer

immediately below). This knowledge is of considerable value to universities and its research policy makers as it aides strategic decisions regarding the distribution of resources and funding. Although many different approaches to find the optimal allocation of resources by universities exist, for example, such as in [33], since global ranking systems are here to stay, ways of improving on these systems will become more valuable.

Furthermore, it has been found that changes in resource allocation have an impact on the type of activities academics concentrate on [33]. This means that those universities in sections "Case Example of the University of Amsterdam" and "Case Example of the Australian National University", who needed to improve in areas such as publications in Nature and Science, can base their decisions for allocations of resources on numbers that will actually help them in the global standing of universities.

Overview of Existing Multi-objective Ranking Methods

The problem of multicriteria ranking has been active in the decision-makers research community for several years. In most cases, authors aim to produce a total order of the alternatives, or as it has previously been referred to in this chapter, a "string list" of object number one to the bottom ranked object. It is possible to classify the methods presented in two approaches: the first one, and most common, is the use of different ways of aggregating the criteria (features) such that different ways of combining allow to obtained better rankings. Examples of the above can be found in [9, 10, 18, 19, 21, 26, 27, 61, 65, 66]. It is arguable whether a specific way of combining or weighting criteria is better than other. The second approach to solving this problem is by modeling it as a multiobjective optimization problem and to solve it by using, for example, evolutionary algorithms. Examples of the above can be found in [15, 23, 35, 36, 46]. This approach has not been fully exploited, and recent advances in evolutionary algorithms and other techniques to solve multi-objective optimization can be applied. It is also well known that the use of specific knowledge of the problem in the solution leads to good performing methods. In the literature, several applications of domain-specific multicriteria ranking methods can also be found, which incorporate characteristics of the problem at hand in order to combine the different criteria to make the decision. Examples of the above can be found in [11, 12, 31, 38, 42, 44, 48, 51, 53, 55].

Independently of the approach used, all methods aim to produce more reliable rankings and benefit decision-makers. The approach presented in this chapter differs from the ones presented above in that it neither aggregates criteria or produces a total ranking of the alternatives. The method uses multiple criteria (features) in the way they are delivered and only compares values of alternatives into each criteria. This simple method allows the user to scale to several alternatives or objects to be ranked and produces groups or layers of alternatives that are distinguishable from the others, but that between them there is no single alternative that is better than other, according to the criteria considered. In other words, the results of the dominance

graph heuristic approach "ranking" provides the user with more information for comparison as the different layers can be compared, as well as the outcomes within the layers.

Conclusion

This chapter has contributed a simple method for hierarchically ranking a set of objects according to multiple characteristics and with multiple objectives in mind. Several extensions and improvements can be made to this method in future efforts. Firstly, in this study only one dataset is used to present the methodology, those produced by the annual Shanghai ranking, to illustrate on the use of our methodology and its implications for research policy. However, even in the case of ranking global universities, other datasets and information sources are available. For instance, the Times Higher Education [60] ranks universities annually, utilizing a different approach. Future research could investigate the combination of information by the use of attributes that come from the different ranking organizations.

Secondly, as stated multiple times, the use of a dominance graph in order to investigate ranking problems is transferrable to different situations. For instance, other researchers could apply this method and test it on other datasets, potentially including a combination of numerical and categorical (ordinal only) data, without the need of an ad hoc weighting scheme, which is very useful under different circumstances. Any research problem in which the objective is to rank or order based on dominance using multiple variables could implement this method as part of the process.

In order to highlight the flexibility of the method presented in this chapter, some extremely different examples are explained. A different use of this method could be for the increased interest in dealing with protected area zoning for conservation as a multi-objective optimization problem [17]. For many years, when multiple geographic areas are examined, the objective is to create a priority ranking of these natural areas based on a variety of variables such as climax condition, educational suitability, species significance, community representation, and human impact [16]. Furthermore, as Smith and Theberge [56] explain, when evaluating natural areas using measurements for a series of criteria, it involves deciding which criteria are important and most significant based on their measurements. The use of a dominance graph would allow alteration and the inclusion or exclusion of certain criteria for each natural area individually and would thus be useful to many different instances across various natural circumstances. For instance, another circumstance for such analysis is the prioritization of farmland preservation for multiple objectives [37] in which the purpose is to preserve it from urbanization.

Another area of obvious application is that of sports [13, 22, 28, 49] and other ranking games in academia, even if they can turn futile when using the wrong assumptions [1]. We are also aware that, after the publication of this paper, a number of other alternative techniques will soon follow that may explore the directed acyclic graph construction hereby presented. We also envision that some researchers may

try to exploit the transitive relations of dominance to build up different ranking scores even for objects at the same level. Given that this method is utilized in a global ranking of educational institutions could possibly be transferable to an area so different such as natural area conservation zoning or sports-related ranking activities, it becomes easy to imagine the wide use of dominance graphs in many other ranking activities.

In conclusion, this study adds to the literature a diverse and dynamic heuristic method for ranking multi-variable problems that is transparent and not biased by arbitrary weightings unilaterally and perpetually selected by the ranking organization. We have outlined the utility of this method in the case of ranking global universities using open, publicly available data and, in doing so, provided a novel methodology to approach the ranking of universities globally. This system allows for clearer benchmarking of universities which are at a similar standing. Overall it helps to provide assistance to research policy and decision-makers while at the same time enables a global standing of the universities in the world's landscape of diversified academic offers.

Cross-References

▸ Multi-objective Optimization

Acknowledgments Pablo Moscato acknowledges support from the Australian Research Council Future Fellowship FT120100060 and Australian Research Council Discovery Projects DP120102576 and DP140104183.

References

1. Adler NJ, Harzing AW (2009) When knowledge wins: transcending the sense and nonsense of academic rankings. Acad Manag Learn Edu 8(1):72–95. https://doi.org/10.5465/AMLE.2009.37012181, http://amle.aom.org/content/8/1/72.abstract, http://amle.aom.org/content/8/1/72.full.pdf+html
2. Agnew T, Whitlock R, Neutze J, Kerr A (1994) Waiting lists for coronary artery surgery: can they be better organised? N Z Med J 107(979):211–215
3. ARWU (2013) ARWU methodology: definition of indicators. http://www.shanghairanking.com/ARWU-Methodology-2013.html
4. Bang-Jensen J, Gutin G (2001) Digraphs: theory, algorithms and applications. Springer, London
5. Baty P (2014) Caltech: secrets of the worlds number one university: how does a tiny institution create such outsized impact? http://www.timeshighereducation.co.uk/features/caltech-secrets-of-the-worlds-number-one-university/2011008.fullarticle
6. Braga-Neto U, Hashimoto R, Dougherty ER, Nguyen DV, Carroll RJ (2004) Is cross-validation better than resubstitution for ranking genes? Bioinformatics 20(2):253–258. https://doi.org/10.1093/bioinformatics/btg399, http://bioinformatics.oxfordjournals.org/content/20/2/253.abstract
7. Buela-Casal G, Gutirrez-Martnez O, Bermdez-Snchez M, Vadillo-Muoz O (2007) Comparative study of international academic rankings of universities. Scientometrics 71(3):349–365. https://doi.org/10.1007/s11192-007-1653-8

8. Burden DJ (1995) The ranking of dental aesthetics. J Orthod 22(3):259–261. http://jorthod.maneyjournals.org/content/22/3/259.abstract
9. Carrillo VM, Taboada HA (2012) A general iterative procedure of the non-numerical ranking preferences method for multiple objective decision making. In: Proceedings of the complex adaptive systems 2012 conference, Washington, DC, pp 135–139. https://doi.org/10.1016/j.procs.2012.09.043
10. Chen Y, Kilgour DM, Hipel KW (2011) An extreme-distance approach to multiple criteria ranking. Math Comput Model 53(5–6):646–658. https://doi.org/10.1016/j.mcm.2010.10.001
11. Dadelo S, Turskis Z, Zavadskas EK, Dadeliene R (2014) Multi-criteria assessment and ranking system of sport team formation based on objective-measured values of criteria set. Expert Syst Appl 41(14):6106–6113. https://doi.org/10.1016/j.eswa.2014.03.036
12. Dalal O, Sengamedu SH, Sanyal S (2012) Multi-objective ranking of comments on web. In: Proceedings of the 21st World Wide Web conference, WWW 2012, Lyon, pp 419–428. https://doi.org/10.1145/2187836.2187894
13. DeOliveira E, Callum R (2004) Who's the best? Data envelopment analysis and ranking players in the national football league. In: Economics, management and optimization in sports, 1st edn. Springer, Berlin/Heidelberg, pp 15–30. https://doi.org/10.1007/978-3-540-24734-0
14. Docampo D (2013) Reproducibility of the Shanghai academic ranking of world universities results. Scientometrics 94(2):567–587. https://doi.org/10.1007/s11192-012-0801-y
15. Fernandez E, Leyva JC (2002) A method based on multiobjective optimization for deriving a ranking from a fuzzy preference relation. Eur J Oper Res 154(1):110–124. https://doi.org/10.1016/S0377-2217(02)00705-1
16. Gehlbach FR (1975) Investigation, evaluation, and priority ranking of natural areas. Biol Conserv 8(2):79–88. https://doi.org/10.1016/0006-3207(75)90033-6, http://www.sciencedirect.com/science/article/pii/0006320775900336
17. Geneletti D, van Duren I (2008) Protected area zoning for conservation and use: a combination of spatial multicriteria and multiobjective evaluation. Landsc Urban Plan 85(2):97–110. https://doi.org/10.1016/j.landurbplan.2007.10.004, http://www.sciencedirect.com/science/article/pii/S0169204607002496
18. Gerani S, Zhai C, Crestani F (2012) Score transformation in linear combination for multi-criteria relevance ranking. In: Advances in information retrieval – proceedings of 34th European conference on IR research, ECIR 2012, Barcelona, pp 256–267
19. Greco S, Mousseau V, Slowinski R (2008) Ordinal regression revisited: multiple criteria ranking using a set of additive value functions. Eur J Oper Res 191(2):416–436. https://doi.org/10.1016/j.ejor.2007.08.013
20. Harvey L, Green D (1993) Defining quality. Assess Eval High Educ 18(1):9–34. https://doi.org/10.1080/0260293930180102, http://www.tandfonline.com/doi/abs/10.1080/0260293930180102
21. Hinloopen E, Nijkamp P, Rietveld P (2004) Integration of ordinal and cardinal information in multi-criteria ranking with imperfect compensation. Eur J Oper Res 158(2):317–338. https://doi.org/10.1016/j.ejor.2003.06.007
22. Hu ZH, Zhou JX, Zhang MJ, Zhao Y (2015) Methods for ranking college sports coaches based on data envelopment analysis and pagerank. Expert Syst 32(6):652–673. https://doi.org/10.1111/exsy.12108, eXSY-Jun-14-124.R1
23. Hughes EJ (2001) Evolutionary multi-objective ranking with uncertainty and noise. In: Evolutionary multi-criterion optimization, proceedings of first international conference, EMO 2001, Zurich, pp 329–343. https://doi.org/10.1007/3-540-44719-9_23
24. Ioannidis J, Patsopoulos N, Kavvoura F, Tatsioni A, Evangelou E, Kouri I, Contopoulos-Ioannidis D, Liberopoulos G (2007) International ranking systems for universities and institutions: a critical appraisal. BMC Medicine 5(1):30. https://doi.org/10.1186/1741-7015-5-30, http://www.biomedcentral.com/1741-7015/5/30
25. Jeremic V, Bulajic M, Martic M, Radojicic Z (2011) A fresh approach to evaluating the academic ranking of world universities. Scientometrics 87(3):587–596. https://doi.org/10.1007/s11192-011-0361-6

26. Kadzinski M, Tervonen T (2013) Robust multi-criteria ranking with additive value models and holistic pair-wise preference statements. Eur J Oper Res 228(1):169–180. https://doi.org/10.1016/j.ejor.2013.01.022
27. Kadzinski M, Greco S, Slowinski R (2012) Selection of a representative value function in robust multiple criteria ranking and choice. Eur J Oper Res 217(3):541–553. https://doi.org/10.1016/j.ejor.2011.09.032
28. Karminsky A, Polozov A (2016) Evolution of ideas about rating and ranking in sports. Springer International Publishing, Cham, pp 187–200. https://doi.org/10.1007/978-3-319-39261-5_7
29. Kee D, Karwowski W (2003) Ranking systems for evaluation of joint and joint motion stressfulness based on perceived discomforts. Appl Ergon 34(2):167–176. https://doi.org/10.1016/S0003-6870(02)00141-2, http://www.sciencedirect.com/science/article/pii/S0003687002001412
30. Kellenberger E, Foata N, Rognan D (2008) Ranking targets in structure-based virtual screening of three-dimensional protein libraries: methods and problems. J Chem Inf Model 48(5):1014–1025. https://doi.org/10.1021/ci800023x, http://pubs.acs.org/doi/abs/10.1021/ci800023x
31. Lerche DB, Brüggemann R, Sørensen PB, Carlsen L, Nielsen OJ (2002) A comparison of partial order technique with three methods of multi-criteria analysis for ranking of chemical substances. J Chem Inf Comput Sci 42(5):1086–1098. https://doi.org/10.1021/ci010268p
32. Li W, Suh YJ, Zhang J (2006) Does logarithm transformation of microarray data affect ranking order of differentially expressed genes? In: 28th annual international conference of the IEEE Engineering in medicine and biology society, EMBS'06, vol Supplement, pp 6593–6596. https://doi.org/10.1109/IEMBS.2006.260896
33. Liefner I (2003) Funding, resource allocation, and performance in higher education systems. High Educ 46(4):469–489. https://doi.org/10.1023/A:1027381906977
34. Liu NC, Cheng Y (2005) The academic ranking of world universities. High Educ Eur 30(2):127–136. https://doi.org/10.1080/03797720500260116
35. López JCL, Aguilera-Contreras MA (2005) A multiobjective evolutionary algorithm for deriving final ranking from a fuzzy outranking relation. In: Evolutionary multi-criterion optimization, proceedings of third international conference, EMO 2005, Guanajuato, pp 235–249. https://doi.org/10.1007/978-3-540-31880-4_17
36. López JCL, Chavira DAG, Noriega JJS (2014) A multiobjective genetic algorithm based on NSGA II for deriving final ranking from a medium-sized fuzzy outranking relation. In: 2014 IEEE symposium on computational intelligence in multi-criteria decision-making, MCDM 2014, Orlando, pp 24–31. https://doi.org/10.1109/MCDM.2014.7007184
37. Machado EA, Stoms DM, Davis FW, Kreitler J (2006) Prioritizing farmland preservation cost-effectively for multiple objectives. J Soil Water Conserv 61(5):250–258. http://www.jswconline.org/content/61/5/250
38. Malekmohammadi B, Zahraie B, Kerachian R (2011) Ranking solutions of multi-objective reservoir operation optimization models using multi-criteria decision analysis. Expert Syst Appl 38(6):7851–7863. https://doi.org/10.1016/j.eswa.2010.12.119
39. Marginson S (2007) Global university rankings: implications in general and for Australia. J High Educ Policy Manag 29:131–142. https://doi.org/10.1080/13600800701351660
40. Merisotis J, Sadlak J (2005) Higher education rankings: evolution, acceptance, and dialogue. High Educ Eur 30:97–101. https://doi.org/10.1080/03797720500260124, http://0-www.tandfonline.com.library.newcastle.edu.au/doi/full/10.1080/03797720500260124.U2hKHPmSx8F
41. Moyles DM, Thompson GL (1969) An algorithm for finding a minimum equivalent graph of a digraph. J ACM 16(3):455–460. https://doi.org/10.1145/321526.321534, http://doi.acm.org/10.1145/321526.321534
42. Nguyen LT, Yee WG, Liew R, Frieder O (2010) Experiences with using SVM-based learning for multi-objective ranking. In: Proceedings of the 19th ACM conference on information and knowledge management, CIKM 2010, Toronto, pp 1917–1920. https://doi.org/10.1145/1871437.1871763, http://doi.acm.org/10.1145/1871437.1871763

43. Nijmegen RU (2011) Nobel prize in physics 2010. http://www.ru.nl/english/research/researchers/nobel-prize/
44. Nwamadi O, Zhu X, Nandi AK (2012) Multi-criteria ranking based greedy algorithm for physical resource block allocation in multi-carrier wireless communication systems. Signal Process 92(11):2706–2717. https://doi.org/10.1016/j.sigpro.2012.04.020
45. Oddershede J (2013) What rankings dont tell you about university excellence. http://theconversation.com/what-rankings-dont-tell-you-about-university-excellence-18704
46. di Pierro F, Khu S, Savic DA (2007) An investigation on preference order ranking scheme for multiobjective evolutionary optimization. IEEE Trans Evol Comput 11(1):17–45. https://doi.org/10.1109/TEVC.2006.876362
47. Pulugurtha SS, Krishnakumar VK, Nambisan SS (2007) New methods to identify and rank high pedestrian crash zones: an illustration. Accid Anal Prev 39(4):800–811. https://doi.org/10.1016/j.aap.2006.12.001, http://www.sciencedirect.com/science/article/pii/S000145750600217X
48. Rad A, Naderi B, Soltani M (2011) Clustering and ranking university majors using data mining and AHP algorithms: a case study in Iran. Expert Syst Appl 38(1):755–763. https://doi.org/10.1016/j.eswa.2010.07.029
49. Radicchi F (2011) Who is the best player ever? A complex network analysis of the history of professional tennis. PLoS ONE 6(2):1–7. https://doi.org/10.1371/journal.pone.0017249
50. Razvan F V (2007) Irreproducibility of the results of the Shanghai academic ranking of world universities. Scientometrics 72(1):25–32. https://doi.org/10.1007/s11192-007-1712-1
51. Reba MNM, Rosli AZ, Makhfuz MA, Sabarudin NS, Roslan NH (2013) Determination of sustainable land potential based on priority ranking: multi-criteria analysis (MCA) technique. In: Computational science and its applications – ICCSA 2013 – proceedings of 13th international conference, Ho Chi Minh City, part VI, pp 212–218. https://doi.org/10.1109/ICCSA.2013.44
52. Saisana M, dHombres B, Saltelli A (2011) Rickety numbers: volatility of university rankings and policy implications. Res Policy 40(1):165–177. https://doi.org/10.1016/j.respol.2010.09.003, http://www.sciencedirect.com/science/article/pii/S0048733310001812
53. Schall D (2014) A multi-criteria ranking framework for partner selection in scientific collaboration environments. Decis Support Syst 59:1–14. https://doi.org/10.1016/j.dss.2013.10.001
54. Schmidt M, Lipson H (2014) Eureqa (version 0.98 beta) [software]. http://www.nutonian.com/research/reference/
55. Shi Z, Hao F (2013) A strategy of multi-criteria decision-making task ranking in social-networks. J Supercomput 66(1):556–571. https://doi.org/10.1007/s11227-013-0934-7
56. Smith PGR, Theberge JB (1987) Evaluating natural areas using multiple criteria: theory and practice. Environ Manag 11(4):447–460. https://doi.org/10.1007/BF01867653, http://link.springer.com/article/10.1007/BF01867653
57. Smits G, Kotanchek M (2005) Pareto-front exploitation in symbolic regression. In: OReilly UM, Yu T, Riolo R, Worzel B (eds) Genetic programming theory and practice II. Genetic programming, vol 8. Springer, pp 283–299. https://doi.org/10.1007/0-387-23254-0_17
58. Sun J, Kuo PH, Riley BP, Kendler KS, Zhao Z (2008) Candidate genes for schizophrenia: a survey of association studies and gene ranking. Am J Med Genet B Neuropsychiatr Genet 147B(7):1173–1181. https://doi.org/10.1002/ajmg.b.30743
59. Thakur M (2007) The impact of ranking systems on higher education and its stakeholders. J Inst Res 13(1):83–96
60. THE (2013) Times higher education: world university rankings. http://www.timeshighereducation.co.uk/world-university-rankings/
61. Toma I, Roman D, Fensel D, Sapkota B, Gómez JM (2007) A multi-criteria service ranking approach based on non-functional properties rules evaluation. In: Proceedings of fifth international conference on service-oriented computing – ICSOC 2007, Vienna, pp 435–441. https://doi.org/10.1007/978-3-540-74974-5_40

62. Vladislavleva EJ dHD Smits GF (2009) Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. IEEE Trans Evol Comput 13(2):333–349. https://doi.org/10.1109/TEVC.2008.926486, http://dl.acm.org/citation.cfm?id=1650365
63. Voll CA, Goodwin JE, Pitney WA (1999) Athletic training education programs: to rank or not to rank? J Athl Train 34(1):48–52. http://search.proquest.com/docview/206648692?accountid=45394
64. de Vries NJ, Carlson J, Moscato P (2014) A data-driven approach to reverse engineering customer engagement models: towards functional constructs. PLoS ONE 9(7):e102,768. https://doi.org/10.1371/journal.pone.0102768
65. Wu J, Liang L (2012) A multiple criteria ranking method based on game cross-evaluation approach. Annals OR 197(1):191–200. https://doi.org/10.1007/s10479-010-0817-8
66. Xu X (2001) The SIR method: a superiority and inferiority ranking method for multiple criteria decision making. Eur J Oper Res 131(3):587–602. https://doi.org/10.1016/S0377-2217(00)00101-6

Index

A
Academic ranking, 1337
Academic Ranking of World Universities (ARWU), 1338, 1339, 1353, 1354, 1359, 1361, 1362, 1364
Acceptance criterion, 580, 583, 584, 586
ACO, *see* Ant colony optimization (ACO)
Action selection, 511, 515–518
Active sub-neighborhoods, 270
Adaptive heuristic selection, 510–515
Adaptive memory algorithm, 1275
Adaptive memory methods, 830
Adaptive memory search, 160
Adaptive metaheuristics
  definition, 6
  GRASP, 11–12
  large neighborhood search, 12–13
  reactive search, 11
  simple adaptive mechanisms, 10–11
Adaptive pursuit (AP), 516–517, 519, 521, 522, 524, 529, 530
Adaptive stochastic gradient descent (ASGD), 1086
Advanced traffic information system, 916
Allele, 1226
Alpha-trace, 1002
Amino acids, 1001–1002
Analysis, 243, 249
Ant colony optimization (ACO), 187–188, 277, 372, 410, 1114–1117, 1275–1276
  algorithms, 376–377, 387
  dynamic problems, 396
  and exact methods, 397–398
  QAP, 373, 383, 388–389, 395, 398, 399
  stochastic problems, 396–397
  and surrogate models, 398
  and tree search methods, 397
  TSP, 373, 383, 388–389, 395, 398, 399
Ant colony system (ACS) algorithm, 373, 379
Anticipatory routing problem, 898
Artificial fitness levels (AFL) and level-based method, 862–866, 874
Artificial immune systems (AIS), 186–187
Artificial pheromone trails, 372
Aspiration criteria, 912
Assignment problems, 269
Associated reduced network, 1108, 1113
Asynchronous cooperation, 825–829, 838
Attributive memory, 744
Automated parameter control, 502–503
Automatic design of algorithms, 497–498
Automatic tuning, 467, 481–482
Automatic vehicle location (AVL) technologies, 916
Autonomous systems (AS), 1129

B
Bacterial foraging optimization algorithm (BFOA), 1092
Bandwidth Minimization Problem (BMP), 1040–1043
Bayesian optimization algorithms (BOAs), 415
Bean's algorithm, 705, 706
Berth allocation problem (BAP), 697, 1055–1061
Berth allocation problem under time-dependent limitations (BAP-TL), 697
Best bound first strategy, 1225
Best improvement, 764
Best in algorithms, 1261
Best-worst ant system (BWAS), 373
BESTUFS, 889, 899
Bias-correcting methods, 163
Biased random-key genetic algorithms (BRKGA), 26–29
  API, 711
  implementation of, 705–707
  parametrized uniform crossover, 705
  probability, 705

© Springer International Publishing AG, part of Springer Nature 2018

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4

Biased random-key genetic programming (BRKGP), 26–27
Big and open data, 1253
Bin packing (BP), 519–521, 527, 532
Biological evolution, 432
Black-box optimization, 496
Boolean expressions, 425
Brain deep nuclei, atlas-based segmentation of, 1095–1096
Brain magnetic resonance images, registration of, 1092–1095
BrainWeb repository, 1092
Branch & bound algorithm, 1225
Breakout local search (BLS), 1271
Brooks' theorem, 1262

C
Candidate list(s), 380
Candidate list strategy, 743
Canonical particle swarm, 644
Capacitated Clustering Problem (CCP), 983
Capacitated vehicle routing problem (CVRP), 689, 1171, 1249
Center string, 1224
Central memory, 821, 827, 828, 831
Chromatic number, 1260
Chromosome, 1226
City logistics
  computation of feasibility, 917–919
  definition, 888
  2-echelon location-routing, 905
  2-echelon single source location problem, 899–904
  environmental concerns and sensitivities, 889
  fleet management and ICT applications, 895–899
  heuristic approach, 919–924
  organizational aspects and decision making, 892–895
  PDVRPTW, 906–915
  relevance of, 889–890
  spatial restrictions, 889
  systems approach, 890–891
  traffic infrastructure, 889
Classification of parallel meta-heuristic strategies, 813–814
Clique, 1259–1261, 1263–1266, 1268–1274, 1276–1278, 1281
  definition, 1260
  maximum clique (*see* Maximum clique)
  number, 1260
Closest string problem (CSP), 1222, 1224–1228
Close to most string problem (CTMSP), 1222, 1228–1229
Clustering and mobile sink, 1149–1151
Clustering problem, 1146, 1153–1154
Coarse-grained parallelism, 811, 827
Cognition-only particle swarm optimization, 1169
Collegial, 814, 824, 825
Coloring, 1259–1265, 1267, 1269, 1270, 1274–1276, 1278, 1281
Column generation (CG) heuristic, 1152
Combination strategies, 721
Combinator(s), 225, 230, 235
Combinatorial optimization, 495, 497, 509, 536, 624, 760, 761, 768, 772, 774, 783
Combinatorial optimization problems (COPs), 270, 374–376, 466
Compact genetic algorithm (CGA), 415
Complement graph, 1260
Complexity, 730
Computational complexity of stochastic search algorithms, 852–853
Configuration, 5, 7–10
Congestion, 1029, 1032
Constraint(s)
  conversion, 230–231
  definition, 229
  programming method, 277
  programming techniques, 398
  satisfaction, 263
Constraint based local search, 227–228
  basics, 229–230
  benefits, 225–226
  car sequencing, 245–248
  constraint hardness, 234–235
  differentiation, 232–233, 254–256
  empirical results, 257–258
  evaluations and violations, 230–232
  invariants, 250–254
  model, 227
  objective functions, 233
  problem, 226–227
  programs, 235–240
  progressive party problem, 241–244
  scene allocation, 248–249
Constriction coefficient, 644
Constriction particle swarm optimization, 1168
Construction heuristics, 1261–1264
Construction method, 265, 282
Constructive search, 565, 568

Container(s), 1052
heuristic framework to store and retrieve, 1073
relocation problem, 1072
storage, 1070–1072
target positions, 1073–1074
transshipment flows of, 1054
Continuous berth allocation problem, 1055
Continuous optimization, 761
Contour matching, 416
Cooperating local search (CLS), 1271
Cooperative search, 812, 819–822
Corridor method (CM), 698
Cost perturbations, 474
Coverage and density control problems, 1147–1148
Coverage problem, 1145
CPLEX v.12.6, 1298, 1301
Credit assignment, 496, 511, 512, 514–516
Crossover operator, 438–440
Customer relationships management (CRM), 1248, 1252
Customer service management (CSM), 1249
Cutting and Packing (C&P) problems, 932
characteristics, 939
irregular packing problems, 957–973
metaheuristic algorithms, 948–956
model-based algorithms, 956–957
no-fit polygon, 937
non-geometric point of view, 937
one-dimensional, 934
output maximization, 937, 940
rectangular problems, 942–948
three-dimensional, 934
two-dimensional, 934
Cut-values, 1030
Cutwidth Minimization Problem (CMP), 351, 1029–1032
Cyclic topology, 1108

D
DAO-QC NSGA-II algorithm, 1010, 1011, 1013, 1014
Data decomposition, 812, 817–818
Data mining, 41–44
in Heuristics, 41, 44–48, 53, 73
Data structure, 1314–1315
Dead-end-elimination, 1005
Decision support system (DSS), 895, 896, 925
Decoder, 704
Deep adaptive greedy search (DAGS), 1263
Define secondary structure of proteins (DSSP) tool, 1009
Deflection, 657
Demand management process, 1249
Density control problem (DCP), 1146
DESIGNER, 1005
Differentiability, 250
Differential evolution (DE), 415–416, 1089
Differentiation, 232–233, 254–256
Diffusion, 825, 827, 828
Dilation, 1041
Direct encoding, 1274
Directed acyclic graph (DAG), 1340, 1341
Directed dominance graphs, 1343
Directed graph (DAG), 1125
Discrete berth allocation problem, 1055
Diversification, 245, 247, 723
phase, 912
refining and tight-refining (DRT), 699
techniques, 433
Diversity, 1003–1004
and equity models, 1003, 1010
Domain barrier, 494, 534
Dominance graph, 1338, 1340–1341, 1349–1362
Drift analysis, 871–880
DSATUR, 1262, 1267, 1276
Dynamic and uncertain environments, 500–501
Dynamic berth allocation problem (DBAP), 697, 1055, 1056
Dynamic insertion heuristic (DINS), 920–922
Dynamic local search (DLS), 1270, 1271
Dynamic optimization, 500
Dynamic tabu search (DTS), 922–924
Dynamic vehicle routing problem (DVRP), 1172

E
(1+1) EA, 854–856, 859, 860
Edge projection, 1278
Educational timetabling, 507
Elite set, 704, 705, 719
generation phase, 45
Elitist ant system (EAS), 373
Embedding, 1026, 1027, 1029, 1041, 1044
Empowerment scheduling, 286
Estimation of distribution algorithms (EDAs), 415
Events, 247
Evolutionary algorithms (EAs), 433, 851, 853–857, 1271–1275, 1314
coevolving solutions, 421–422
combinatorial optimization, 424
differential evolution, 415–416
dynamic optimisation, 421

Evolutionary algorithms (EAs) (*Cont.*)
estimation of distribution algorithms, 415
evolution strategies and evolutionary programming, 414
genetic algorithms, 412–413
hybridisation with local search, 418
methodology, 422
multimodal optimisation, 418–419
multiobjective optimisation, 419–420
niching and co-evolution, 424–425
open source frameworks, 423
parameters, 422–423
Parkinson's disease diagnosis, 426
Evolutionary computation, 494, 1342
Evolutionary optimization, 718
Evolutionary programming (EP), 414
Evolution strategy, 414, 442
Exact neighborhood functions, 306–310
Explicit memory, 744

F
Facebook, 1124
Far from most string problem (FFMSP), 1222, 1230–1232
Farthest string problem (FSP), 1222, 1229–1230
Fast local search, 270–271, 274, 275
Feasible solutions, 40
Feature-based approach, 1080
Fine-grained parallelism, 811, 815
First improvement strategy, 354
Fitness level, 1226
Flexibility optimization, 1074
Flow shop (FS), 519, 520, 522, 532
Forests, 1308
FPmax* algorithm, 44
Framework-centric period, 801–803
Frequency-based memory, 748, 749
Frequency memory, 912
Frequent itemset mining (FIM) problem, 43
Functional parallelism, 811, 814, 837
Functions, 25
of unitation, 857, 862, 872, 879

G
gbest model, 652
Gene expression programming (GEP), 24
Generalized assignment problem, 289–290, 904
Generalized Max-Mean Dispersion Problem, 982
Genes, 1226
GENEsYs, 1272
GENET, 263
Genetic algorithm (GA), 412–413, 432, 608, 1005, 1089–1091, 1226, 1271–1275
Genetic operators, 438
Genetic programming (GP), 417, 779
Genotype-phenotype mapping, 417
Global distance test (GDT), 1019
Global optimization problem (GOP), 345, 775–779
GLS, *see* Guided local search (GLS)
Goods distribution centers, 893
Google, 1124
Gradient(s), 232–233, 254
algorithm, 158
Gradient descent (GD) method, 1086
Graph, 1026, 1027, 1035, 1037, 1042
Graph layout, 1026, 1043
Greedy adaptive search procedures, 156, 160
Greedy maximum residual energy, 1154
Greedy methods, 564, 565
Greedy partition crossover (GPX), 1275
Greedy randomized, 729
Greedy randomized adaptive search procedure (GRASP), 48, 548, 551, 554, 566, 594, 709, 1231, 1232, 1266, 1269
framework, 472
hybridizations of, 476–481
procedures, 471
reactive, 471
Grid data set, 1032
Grouping representation, 1273
Guided fast local search (GFLS), 271, 274–275, 284
Guided genetic algorithm (GGA), 276
Guided local search (GLS), 263
applications, 288–291
in commercial packages, 291
extensions, 275
hybrid, 276–277
multi-objective optimization, 278–281
on TSP, 281–284
variations, 277–278
WSP, 284–288
Guided Pareto local search (GPLS), 279–281
Guided tabu search (GTS), 277
Guiding solutions, 755

H
Hamiltonian cycle (HC), 304–306
Harwell-Boeing data set, 1032
Hazard rate function, 212

Heuristic(s), 225, 226, 236, 239, 248, 257, 1310
construction, 1261–1264
on continuous formulations, 1276–1278
generation, 491
hyper-heuristics (*see* Hyper-heuristics)
information, 372
and metaheuristic algorithms, 1226
meta-heuristics (*see* Meta-heuristic(s))
methods, 262, 1144–1156
ranking (*see* Ranking Heuristic-feasibility operator)
selection, 491, 492, 495, 510–515
See also Meta-heuristic(s)
Hill-climbing, 262
History of meta-heuristics
early period, 797–798
framework-centric period, 801–803
metaphor-centric period, 803–804
method-centric period, 798–801
periods, 794
pre-theoretical period, 795–796
scientific period, 804–805
Homologs, 1007
Hybrid algorithms, 1269
Hybrid berth allocation problem, 1055
Hybrid data mining, 41
Hybrid DM-GRASP, 49–56
Hybrid DM-GRASP+VND heuristic, 58–64
Hybrid DM-ILS, 63–70
Hybrid GAs, 434
Hybridised metaheuristics, 609
Hybrid metaheuristic, 801
Hybrid particle swarm optimization (HybPSO), 1177–1178
Hybrid phase, 46
Hyper-heuristics, 14–17, 491
automated parameter control and tuning, 502–503
automatic design of algorithms, 497–498
characteristic of, 491
definition, 491
developments, 508–509
dynamic and uncertain environments, 500–501
educational timetabling applications, 507
games and education, 508
iterated local search (*see* Iterated local search based hyper-heuristics (HHILS))
machine learning methodologies, 501–502
methodology of, 494–497
multi-objective optimization, 499–500
off-line learning, 491
on-line learning, 491
scheduling problems, 503–507
theoretical results in, 498–499
Hypervolume, 185

I

Image alignment, 1080
Image registration, 1080
early evolutionary method, 1088
optimization procedure, 1085
problem statement, 1082
similarity metric, 1083–1085
suitability of MHs in, 1086–1087
transformation model, 1082–1083
Immigrants, 704
Impasse neighborhood, 1267, 1275
Improvement methods, 265–266, 282–283
Incremental, 225, 226, 232, 239, 249, 252–254, 257
Independence number, 1260
Independent multi search, 818–819
Independent set, 1260, 1266, 1268–1273, 1277, 1278
Indicator-Based Evolutionary Algorithm (IBEA), 185
Individual improvement heuristic, 611
Inertia particle swarm optimization (IPSO), 1168
Inertia weight, 644
Information and communications technologies (ICT), 895–899
Initiating solution, 755
Integer linear program (ILP), 1224, 1229
CPLEX, 1235
CSP, 1225
FFMSP, 1230–1232
Lagrangian relaxation of, 1228
linear programming relaxation of, 1226
Integrated clustering and routing problem (ICRP), 1155
Integrative cooperative search, 833–836
Intensification, 243, 244, 257
phase, 911
Intensity-based approach, 1081
Internal vehicle scheduling problem (IVSP), 1065–1070
Invariants, 224, 250–254, 256
Inventory routing problem (IRP), 1173
Inverse folding problem (IFP), 1003, 1007
I-TASSER, 1006, 1016
Iterated greedy (IG), 549, 570–571
acceptance criterion, 554
algorithmic outline of, 551, 555

Iterated greedy (IG) (*Cont.*)
construction, 553–554
destruction, 552–553
flow shop scheduling, 558–563
greedy construction heuristics, 550–551
historical development, 563–565
local search, 554–555, 567–568
PFSP, 557–558
principle of, 551
repeated (greedy) construction algorithms, 566–567
for routing problems, 570
for scheduling problems, 569–570
SCP, 556–557
tree search algorithms, 568–569
TSP, 555–556
Iterated local search (ILS), 48, 554, 555, 580–586, 1270
algorithms, 587–590
framework, 583–586
historical development of, 591–592
local search heuristics, 581–583
for other problems, 598–599
relationship of, 592–595
for routing problems, 596
for scheduling problems, 597–598
Iterated local search based hyper-heuristics (HHILS)
action selection models, 515–518
adaptive heuristic selection, 510–515
HHILS-AP, 519, 521, 522, 524, 529, 530, 532
HHILS-MAB, 519–522, 524, 527, 529, 530
HHILS-PM, 519, 521, 522, 524, 527, 529, 530
HHILS-SA, 519–522, 524, 529, 532
HyFlex, 509–510, 532–533
performance distribution of, 527
Iterations-to-target-solution plot, 708
Iterative closest point (ICP), 1081, 1085–1086
Iterative construction heuristic, 565
Iterative flattening, 565

K
Karush-Kuhn-Tucker conditions, 345
Kempe chains, 1267
Knapsack problem, 904
Knowledge collegial, 814, 830
Knowledge synchronization, 814, 818, 823, 824

L
Labeling, 1027
Lagrangian heuristics, 1312
Lagrangian relaxation, 1310
Landside, 1053
Large neighborhood search method, 565
Large-stop Markov chains (LSMC) algorithm, 591
lbest model, 652
Left vertices, 1035
Legal neighbors, 236
Linear arrangement, 1027, 1031–1035
Linear complementary problem (LCP), 1277
Linear genetic programming (LGP), 24
Linear layout problems
BMP, 1040–1043
CMP, 1029–1032
MinLA, 1032–1035
SumCut, 1038–1040
VSP, 1035–1038
Linear ordering, 1027, 1033
Linear ordering problem (LOP), 165, 747
Linear program (LP), 1156
Linear programming relaxation, 1226, 1230
Linear representation, 25
Linear topology, 1108
Load, 1027, 1032
Local branching, 699
constraint, 774
Localization problem, 1145
Local optima, number of, 157
Local pheromone update, 385
Local search (LS), 262, 548, 550, 552, 558, 559, 561, 565, 567–568, 570, 572, 1311
algorithms, 386
heuristics, 581–583
hybridization, 554–555
parallelization, 815
procedure, 585–586
results, 561–563
strategies, 732
technique, 609
Local search based methods, 352–355, 1265–1266
BLS, 1271
CLS, 1271
DLS, 1270, 1271
GRASP, 1269
iterated local search, 1270
PLS, 1271
simulated annealing method, 1266, 1268
tabu search, 1268–1269
VNS, 1270
VSS, 1270
Location and network design, 1245
Location routing problem (LRP), 691, 1172

Location routing problem with stochastic demands (LRPSD), 1172
Logistics
  city logistics systems (*see* City logistics systems)
  definition, 888
Logistics management (LM), 1244
Lot sizing problem, 1292–1293
  computational experiments, 1301–1304
  determinism on demands, 1293
  dynamic demands, 1293
  inventory and backlogging, 1295
  items, 1294
  levels, 1294
  machines, 1294
  multi-plant, multi-item lot sizing problem, 1296–1299
  network flow-based formulation, 1299–1301
  non-deterministic/stochastic demands, 1293
  NP-complete, 1295
  planning horizon, 1293
  setup structure, 1294–1295
  stationary demands, 1293
Lot transfers, 1294, 1298
Lovász theta, 1278

M
Machine learning methodologies, 501–502
Manufacturing and resource strategies, 1246
Manufacturing flow management process, 1249
Maritime container terminals, 1052, 1053
  areas of, 1052–1053
  goal of, 1052
  landslide, 1053
  optimization problems, 1054–1055
  seaside, 1052
  suitable performance of, 1053
  yard, 1053, 1071
Master-slave, 814, 817, 823
Mathematical programming (MP), 122, 760, 769, 772
Matheuristics, 122, 397, 1310
Max-cut problem (MCP), 346, 347
Maximally diverse grouping problem (MDGP), 984
Maximum clique
  approximate solutions, 1279
  construction heuristics, 1261–1264
  local search, 1264–1265
  size of, 1260
  and vertex coloring construction heuristics, 1261 *See also* Meta-heuristic(s)
Maximum cut problem, 207
Maximum diversity problem (MDP), 980
Maximum flow problem (MFP), 1126–1127
Maximum MinSum Dispersion Problem, 983
Maximum satisfiability (Max-SAT), 509, 519, 532
  problem, 291
Max Mean Dispersion Problem, 982
Max-Mean model, 993
Max-min ant system (MMAS), 373
Max-Min Diversity Problem (MMDP), 980, 982
Max-Min model, 990–991
Max-MinSum model, 993–995
Max-residual-capacity, 1152
Max-Sum Diversity Model, 980
Max-Sum model, 991–993
MDM-MSH, 47, 53, 73
  pseudo-code, 47
Medical imaging, 1080
Medium-term decision, 1292
Memetic algorithms (MA), 418, 608–624
  applications, 620–621
  classical, 611–614
  complicated, 614–615
  future-generation, 622–623
  for network alignment, 616–617
  structure, 610–615
  for WCSPs, 617–619
Memory-based heuristics, 70–83
Memory-less heuristics, 41, 48–70
Meta-heuristic(s), 40, 122, 225, 236, 239, 257, 501, 505, 535, 558, 564, 567, 580, 582, 583, 592, 599, 760, 762, 775, 784, 1081, 1151, 1265, 1267, 1269, 1270, 1274–1276, 1278, 1281, 1310
  algorithm, 263
  approach, 613, 622, 623
  local search based methods (*see* Local search based methods)
  natural gas pipeline networks (*see* Natural gas pipeline networks, meta-heuristics)
  population based methods, 1271–1276
Metaphor-centric period, 803–804
Method-centric period, 798–801
Min-conflict, 239, 240
Min-Diff model, 995–996
Minimum cost flow problem (MCFP), 1127
Minimum equivalent subdigraph, 1340
Minimum fuel cost problem (MFCP), 1106

Minimum Linear Arrangement (MinLA) Problem, 1032–1035
Mixed algorithm portfolio, 217
Mixed-integer nonlinear programming (MINLP), 1105, 1107, 1114–1117
Mixed-integer programs, 760
Mixed variable neighborhood descent, 362–365
Model, 225, 227, 229–235, 240, 245, 246, 248
Monte Carlo method, 156
Motzkin-Straus formulation, 1276, 1277
Multi-armed bandit (MAB), 517, 519–522, 524, 527, 529, 530
Multi-commodity flow problems, 1128–1129
Multi depot vehicle routing problem (MDVRP), 1172
Multi-level meta-heuristics, 7, 13–17
Multi-level search, 828
Multi-level single linkage, 158
Multi-modal registration, 1085
Multi-objective, 1325
  hyper-heuristics, 499
  optimization, 499–500, 503
  ranking problem (*see* Ranking)
Multi-objective ACO (MOACO), 395–396
Multi-objective evolutionary algorithm (MOEA), 178, 281, 419–420
Multi-objective evolutionary algorithm based on decomposition (MOEA/D), 184–185
Multi objective genetic algorithm (MOGA), 1003
Multi-objectivization, 1007
Multi-plant capacitated lot sizing problem (MPCLSP), 1292, 1296–1299, 1304, 1305
Multi-plant uncapacitated lot sizing problem (MPULSP), 1292, 1295, 1297–1299, 1305
Multi search, 812
Multi-start (MS)-based heuristics, 1153
Multi-start heuristic (MSH), 45
Multi-start relaxation-based algorithm, 1228
Mutant(s), 704, 706
  vector, 416
Mutation, 608
  operator, 440–442

N

Natural gas pipeline networks, meta-heuristics, 1105–1106, 1108–1110
  ant colony optimization, 1114–1117
  basic model, description of, 1106–1108
  network topology, 1108–1110
  non-isothermal systems, PSO for, 1117
  tabu search, 1111–1114
  time-dependent systems, SA for, 1118
Natural gas transmission system, 1105
Nearest-to-base-station, 1152
Neighborhood, 225, 228, 235, 238, 241–244, 249, 643
  functions, exact, 306
  size, 163
  structures, 344
  topology, 652
Neighbor selection, 236
Nested (composite) variable neighborhood descent procedure, 359–362
Network topology, 1108–1110
Network virtualization, 1134
Niching, 419, 425
No free lunch theorem (NFLT), 416
Non-elite set, 704, 705
Nondominated sorting genetic algorithm II (NSGA-II), 183–184, 1325
Nonlinear programming (NLP), 1105, 1107, 1111–1114
Nonsequential dynamic programming, 1112
NP-complete, 1295
NP-hard, 1029, 1033, 1037, 1040
Numbering, 1027, 1040

O

Objective functions, 226, 229, 233, 235, 248–250, 254, 256
Off-line learning hyper-heuristics, 491
On-line learning hyper-heuristics, 491
Open shortest path first (OSPF), 1129
Open vehicle routing problem (OVRP), 1171
Optimal coloring, 1260
Optimization, 490, 498, 500, 503, 505, 509, 516, 536, 760, 761, 770, 771, 783
  ant colony, 505
  black-box, 496
  combinatorial, 495, 497, 509, 536
  dynamic, 500
  multi-objective, 499–500, 503
  particle swarm, 505, 506
  problems, 433
  single-objective, 502
Order based encoding, 1274
Order fulfillment process, 1249

P

Packing, 497, 498, 509, 521, 532, 535, 536
  BP, 495, 497, 501, 503, 509, 519, 521, 532

and cutting problems, 497, 535
two-dimensional, 498
Parallel computation, 811
Parallel computing, 622
Parallel Distributed Graphical Programming (PDGP), 24
Parallel meta-heuristics, 810, 811
Parallel speedup, 213
Parameter λ, 267
Parameterized flexibility optimization, 1074
Parameterized uniform crossover, 705
Parameters-based approach, 1081
Pareto archived evolution strategy (PAES), 183
Pareto envelope-based selection algorithm (PESA), 184
Pareto local optimum set, 280
Pareto local search (PLS), 278–279
Pareto optimal front, 419, 420
Partial OPtimization Metaheuristic Under Special Intensification Conditions, *See* POPMUSIC
Particles, 642
Particle swarm optimization (PSO), 159, 187, 410, 640, 1090, 1117–1118, 1166–1167, 1183, 1184
CNTPSO, 1178–1179
cognition-only particle swarm optimization, 1169
constriction particle swarm optimization, 1168
DAPSO, 1179
DHPD, 1181
DPSO, 1181
GLNPSO, 1180
h_PSO, 1179
HybGenPSO, 1177
HybPSO, 1177–1178
IPSO, 1168
local neighborhood topology particle swarm optimization, 1169
MODPSO, 1181
MOPSO, 1180–1181
2MPSO, 1180
NPSO, 1181
PMPSO, 1180
PVPSO, 1180
QPSO, 1177
social-only particle swarm optimization, 1169
SPSO, 1180
SR-PSO, 1176
StPSO, 1181
VRP (*see* Vehicle routing problem (VRP))
Path relinking, 477, 709, 755–756
Penalty-based approach, 262
Penalty function method, 1266
Performance evaluation, 812
Periodic construction-deconstruction, 565
Periodic vehicle routing problem (PVRP), 1172
Permutation
cost of, 282
problems, 661
Permutation-based representations, 350–352
Permutation flow shop scheduling problem (PFSP), 165, 557–558, 589–590
Personnel scheduling (PS), 519, 520, 523, 527, 532
Perturbation, 584–587
mechanism, 1227
Phased local search (PLS), 1271
Pheromone
deposition, 382
limits, 386
reinitialization, 384–385
trails, 372
p-hub
median problem, 725–728, 732
scatter search, 728–733
Pickup and delivery vehicle routing problem with time windows (PDVRPTW), 906–915
Pipeline optimization, 1114
Placement (deployment) problem, 1145
Point-to-point shortest path problem (PPSPP), 1125–1126
POPMUSIC
applications, 694–697
location-routing, adaptation for, 691–694
template, 688–691
Population-based incremental learning (PBIL), 415
Population based metaheuristic, 612
Population based methods
ant colony optimization, 1275–1276
evolutionary and genetic algorithms, 1272–1275
Prefix notation convention (P-rule), 25
Pre-marshalling problem, 1072
Pre-theoretical period, 795–796
Primer design, 1223
Primitive set, 25
Probability matching (PM), 515, 516, 519, 521, 522, 524, 527, 529, 530
Product development and commercialization process, 1249–1250
Production routing problem (PRP), 1173
Productivity, 1293

PROFphd, 1009
Program-expression (PE), 25
Protein, 1000
  design, 1004–1006
  primary structure, 1002
  secondary structure, 1002
  sequence, 1001
  tertiary structure, 1002
Proximate optimality principle (POP), 476
Pseudo-code, 705
Pseudo-random proportional rule, 379
Pure and hybrid multistart iterative heuristics, 1231

Q
Quadratic assignment problem (QAP), 165, 277, 373, 374, 581, 588–589
Quantile constraint (QC), 1011–1012
Quantum PSO (QPSO), 1177
Quay crane scheduling problem (QCSP), 1061–1065

R
Radio link frequency assignment problem (RLFAP), 290
Random insertion heuristic (RIH), 556
Randomised search heuristics, 850
Random-key genetic algorithms (RKGA), 704
  BRKGA (*see* Biased random-key genetic algorithms (BRKGA))
  multiple populations, 709–710
  restarting, 707–709
  specification, 710
Rank-based ant system (RAS), 373
Ranking, 1365–1366
  academic, 1337
  Alumni attribute, 1359–1361
  dominance graph, 1340–1341
  performance per capita attribute, 1361–1363
  quality of, 1336
  research policy makers, implications for, 1364–1365
  sensitivity analysis of mobility, 1347–1352
  symbolic regression analysis, 1342–1343, 1350–1359
  transitive reduction, 1340–1341
  transparent and user-centric ranking method, 1363–1364
  university ranking results, 1343–1347
Ranking Heuristic-feasibility operator, 1273
Re-routing algorithm (RRA), 916, 918
Reactive GRASP, 473–474
Reactive local search (RLS), 1269
Recency-based attributive memory, 744
Recombination, 608
Recursive largest first (RLF), 1262, 1263, 1267, 1276
Reference set, 724
Relative percentage deviation (RPD), 391, 392
Replacement policy, 442
Repulsion, 658
Residence count, 748
Residues, 1001
Resource allocation problem, 269–270
Restart distribution, 206, 210
Restart period, 210
Restart strategy, 208
Restricted candidate list (RCL), 1270
Restricted Hamiltonian cycle (RHC), 304, 306
Returns management, 1250
Right vertices, 1035
Rigid synchronization, 814
Root mean square error (RMSE), 782
Routing/connectivity, 1148–1149
Routing problem, 1145
Routing/scheduling problems, 269, 288–289
Ruin-and-recreate, 564
Runtime analysis, 859, 860

S
Sandwich theorem, 1278
Satisfiability (SAT) problem, 291
Scatter search (SS), 48, 718, 723, 1090–1091, 1275
  p-hub problem, 728–733
Schedule length, 388
Scheduling, 1198
  ATC, 1209
  dispatching rules, 1207
  due date, 1202
  EDD, 1208
  FCFS, 1207
  flow shop, 1202
  heuristics, 1200–1201
  hybrid shop, 1203
  iterated greedy, 1215–1216
  job shop, 1203
  makespan, 1205
  manufacturing, 1199
  metaheuristics, 1213
  NEH heuristic, 1210–1212
  parallel machine, 1202
  problems, 503–507
  processing route, 1201
  processing times, 1201

single machine, 1202
SPT, 1207–1208
weight, 1202
Scientific period, 804–805
SC-values, 1038
Search, constraint, *see* Constraint based local search
Search control and communications, 814
Search control cardinality, 813
Search differentiation, 814
Search, iterated local, *see* Iterated local search based hyper-heuristics (HHILS)
Search-space decomposition, 812, 817, 838
Seaside, 1052
Seed-part selection procedure, 693–694
Selective strategy, 735
Semidefinite programming (SDP), 1278
Sensor localization, 1151–1152
Sensor nodes, 1142
Sep-values, 1036
Sequence identity, 1007–1008
Sequencing, 1198
Set covering problem (SCP), 556–557
Shift operator, 913
Ship routing problem (SRP), 1173
Simheuristics, 1253
Similarity metric, 1081
Simulated annealing (SA), 263, 548, 550, 554, 557, 558, 563, 609, 1118, 1266–1268, 1273
Single algorithm portfolio, 212
Single machine total weighted tardiness problem (SMTWTP), 597
Single-objective optimization, 502
Sink, 1142
Small data set, 1032
S metric selection (SMS-EMOA), 185
Social-only particle swarm optimization, 1169
Solution construction, 379–381
Solution representation, 282, 438
Sourcing and inventory strategy, 1246
Spatial transformation, 1080
Speedup ratio, 217
Stability number, 1260
Stable set, 1260
STABULUS, 1268
Stacking problem, 1072
Static berth allocation problem, 1055
Statistical estimation methods, 163
Steady-state MINLP models, 1108
Steady-state replacement, 1273
Stochastic local search (SLS), 548, 550, 564, 565, 568, 570, 572, 586, 592
algorithms, 40
hybrid, 593–594
population-based, 595
simple, 593
Stochastic search algorithms, 850, 851
Stopping criterion, 268, 912
Storage location assignment problem, 508
Strategic oscillation, 161, 750–755
Strategy parameters, 414
Strength Pareto Evolutionary Algorithm 2 (SPEA2), 182–183
Stretching technique, 656
String problems
bacterial infections, diagnostic probes for, 1222
CSP, 1224–1228
CTMSP, 1228–1229
drug targets, 1223
experimental evaluation, 1233–1238
FFMSP, 1230–1232
FSP, 1229–1230
motif search, 1223–1224
primer design, 1223
Strings consensus, 1222
Subgradient methods, 1312
Sub-neighbourhoods, 270
SubneighbourhoodsForMove, 273
SumCut problem, 1038–1040
Super-linear speedup, 212
Supplier relationship management, 1249
Supply chain management (SCM), 1164, 1248
big and open data, 1253
customer relationships management, 1248
customer service management, 1249
definition, 1243–1244
demand management process, 1249
location and network design, 1245
manufacturing and resource strategies, 1246
manufacturing flow management process, 1249
metaheuristics algorithms, 1250–1252
order fulfillment process, 1249
product development and commercialization, 1249–1250
returns management, 1250
sourcing and inventory strategy, 1246
supplier relationship management, 1249
sustainability and green strategies, 1247
transportation and distribution strategies, 1247
uncertainty, 1253
Survival of the fittest, 1226
Sustainability and green strategies, 1247
Swap move, 348

Swarm, 642
explosion effect, 646
Sweep algorithm, 909
Symbolic regression analysis, 1342–1343, 1350–1354
Australian National University, 1358–1359
University of Amsterdam, 1356–1358
University of Copenhagen, 1355, 1356
Synchronous cooperation, 822–823
Syntax trees (ST), 24
Synthesized search, 243–244, 247, 257, 258

T
Tabu, 236, 243, 244, 246, 247, 249, 257
list, 911, 1268
tenures, 911, 1268
Tabu search (TS), 48, 156, 207, 263, 609, 909, 912, 913, 1105, 1111–1114, 1268–1269, 1273–1276
long term memory, 748–750
oscillating assignment, 742
path relinking, 755–757
scatter search, 742
short term memory, 742–748
strategic oscillation, 750–755
strongly determined and consistent variables, 742
TABUCOL, 1269
Tail inequalities, 860–862
Target vector, 416
Team orienteering problem (TOP), 1172
Terminals, 25
Theoretical foundations hyper-heuristics, 498–499
Threshold accepting algorithm, 162
Time-dependency, 898
Timetabling, 490, 493, 495, 498, 507
Time-to-target-solution-value, 1232
TM-score, 1018, 1019
Tournament, 438
Transformation model, 1082–1083
Transient models, 1109
Transition count, 748
Transitive reduction, 1340–1341
Transportation and distribution strategies, 1247
Transportation networks, 1104, 1108, 1118
Transshipment flows, 1054
Transshipment of freights, 1052
Traveling salesman problem (TSP), 288–289, 373, 374, 519, 520, 525, 532, 555–556, 581, 587–588, 689
metric, 304–306
Tree(s), 1308
representation, 24–25
topology, 1108
Trial vector, 416

U
Uncertainty, 1253
Uniform restart strategy, 208
Upper confidence bound (UCB) algorithm, 517

V
Variable depth search, 1270
Variable neighborhood decomposition search, 699
Variable neighborhood descent (VND), 344, 356–365, 1324
Variable neighborhood search (VNS), 265, 344, 355, 477, 594, 760, 762, 784, 1270, 1324
Variable objective search, 1277
Variable space search (VSS), 1270
Vehicle routing and scheduling problem (VRSP), 1173
Vehicle routing problem (VRP), 288, 289, 519, 520, 526, 527, 532, 596, 898, 909, 1065, 1174–1184
CVRP, 1171
DVRP, 1172
HVRP, 1173
MDVRP, 1172
OVRP, 1171
PVRP, 1172
VRPFD, 1172
VRPSD, 1171
VRPSPD, 1171
VRPSTT, 1172
VRPTW, 1171
VRPUD, 1173
VRSP, 1173
Vehicle routing problem with fuzzy demands (VRPFD), 1172
Vehicle routing problem with simultaneously pickup and delivery (VRPSPD), 1171
Vehicle routing problem with stochastic demands (VRPSD), 1171
Vehicle routing problem with stochastic travel times (VRPSTT), 1172
Vehicle routing problem with time windows (VRPTW), 898, 1171
Vehicle routing problem with uncertain demands (VRPUD), 1173

Vehicle scheduling problem (VSP) heuristic framework for, 1069
Velocity, 643
Vertex coloring, 1259–1264
  approximate solutions, 1280
  construction heuristics, 1261–1264
  local search, 1264–1265
  proper, 1260
  See also Meta-heuristic(s)
Vertex Separation Problem (VSP), 1035–1038
Violations, 230–233, 241, 243, 245, 247, 254
Virtual network embedded problem (VNEP), 1124, 1134–1135
  description, 1135–1136
  heuristic approaches, 1136–1138
VND, *see* Variable neighborhood descent (VND)
VNS, *see* Variable neighborhood search (VNS)
VSS, *see* Variable space search (VSS)

W
Webgraphs, 1124
Weighted constraint satisfaction problems (WCSPs), 617–619
Weight setting problem (WSP), 1124, 1129
  heuristic approaches, 1131–1134
  on telecommunication networks, 1130–1131
Wilcoxon test, 1013
Wireless sensor networks (WSN), 1142
Workforce scheduling problem (WSP), 284–288, 290–291
Worst out, 1261

X
XRLF, 1263

Y
Yard, 1053, 1070, 1071