

COMBINATORIAL OPTIMIZATION

The Traveling Salesman Problem and Its Variations

Gregory Gutin and Abraham P. Punnen (Eds.)

THE TRAVELING SALESMAN PROBLEM AND ITS VARIATIONS

Combinatorial Optimization

VOLUME 12

Through monographs and contributed works the objective of the series is to publish state of the art expository research covering all topics in the field of combinatorial optimization. In addition, the series will include books, which are suitable for graduate level courses in computer science, engineering, business, applied mathematics, and operations research.

Combinatorial (or discrete) optimization problems arise in various applications, including communications network design, VLSI design, machine vision, airline crew scheduling, corporate planning, computer-aided design and manufacturing, database query design, cellular telephone frequency assignment, constraint directed reasoning, and computational biology. The topics of the books will cover complexity analysis and algorithm design (parallel and serial), computational experiments and application in science and engineering.

Series Editors

Ding-Zhu Du, *University of Minnesota*
Panos M. Pardalos, *University of Florida*

Advisory Editorial Board

Afonso Ferreira, *Institut National de Recherche en Informatique et en Automatique (INRIA)-
MASCOTTE*
Jun Gu, *Hong Kong University of Science and Technology*
David S. Johnson, *AT&T Research*
James B. Orlin, *M.I.T.*
Christos H. Papadimitriou, *University of California at Berkeley*
Fred S. Roberts, *Rutgers University*
Paul Spirakis, *Computer Tech Institute (CTI)*

THE TRAVELING SALESMAN PROBLEM AND ITS VARIATIONS

Edited by

GREGORY GUTIN
Royal Holloway, University of London, UK

ABRAHAM P. PUNNEN
University of New Brunswick, Saint John, Canada

 Springer

Library of Congress Control Number: 2006933131

soft cover ISBN 0-387-44459-9 ISBN 978-0-387-44459-8

hard cover ISBN 1-4020-0664-0 ISBN 978-1-4020-0664-7

ISBN 0-306-48213-4 (eBook)

Printed on acid-free paper.

© 2007 Springer Science+Business Media, LLC

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

springer.com

Contents

Preface	xi
Contributing Authors	xv
1	
The Traveling Salesman Problem: Applications, Formulations and Variations	1
<i>Abraham P. Punnen</i>	
1.1 Introduction	1
1.2 Simple Variations of the TSP	7
1.3 Applications of TSP	9
1.4 Alternative representations of the TSP	15
1.5 Matrix Transformations	23
1.6 More Variations of the TSP	24
2	
Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP	29
<i>Denis Naddef</i>	
2.1 Introduction	29
2.2 Integer linear programming models	30
2.3 STSP polytope and relaxations	35
2.4 The graphical relaxation Framework	44
2.5 The Comb inequalities	58
2.6 The Star and Path inequalities	62
2.7 The Clique Tree and Bipartition inequalities	67
2.8 The Ladder inequalities	71
2.9 A general approach to some TSP valid inequalities	73
2.10 A unifying family of inequalities	77
2.11 Domino inequalities	78
2.12 Other inequalities	81
2.13 The separation problem	82
2.14 Greedy heuristic for minimum cut	84
2.15 Graph associated to a vector $x^* \in \mathbb{R}^E$	85
2.16 Heuristics for Comb Separation	86
2.17 Separation of multi-handle inequalities	94
2.18 Separation outside the template paradigm	100
2.19 Branch-and-Cut implementation of the STSP	105
2.20 Computational results	113
2.21 Conclusion	114

3

Polyhedral Theory for the Asymmetric Traveling Salesman Problem 117

Egon Balas and Matteo Fischetti

- 3.1 Introduction 117
- 3.2 Basic ATS inequalities 120
- 3.3 The monotone ATS polytope 128
- 3.4 Facet-lifting procedures 133
- 3.5 Equivalence of inequalities and canonical forms 142
- 3.6 Odd closed alternating trail inequalities 145
- 3.7 Source-destination inequalities 150
- 3.8 Lifted cycle inequalities 155

4

Exact Methods for the Asymmetric Traveling Salesman Problem 169

Matteo Fischetti, Andrea Lodi and Paolo Toth

- 4.1 Introduction 169
- 4.2 AP-Based Branch-and-Bound Methods 172
- 4.3 An Additive Branch-and-Bound Method 176
- 4.4 A Branch-and-Cut Approach 181
- 4.5 Computational Experiments 194

5

Approximation Algorithms for Geometric TSP 207

Sanjeev Arora

- 5.1 Background on Approximation 208
- 5.2 Introduction to the Algorithm 209
- 5.3 Simpler Algorithm 214
- 5.4 Better Algorithm 215
- 5.5 Faster Algorithm 219
- 5.6 Generalizations to other Problems 220

6

Exponential Neighborhoods and Domination Analysis for the TSP 223

Gregory Gutin, Anders Yeo and Alexei Zverovitch

- 6.1 Introduction, Terminology and Notation 223
- 6.2 Exponential Neighborhoods 228
- 6.3 Upper Bounds for Neighborhood Size 237
- 6.4 Diameters of Neighborhood Structure Digraphs 240
- 6.5 Domination Analysis 244
- 6.6 Further Research 254

7

Probabilistic Analysis of the TSP 257

A. M. Frieze and J. E. Yukich

- 7.1 Introduction 257
- 7.2 Hamiltonian Cycles in Random Graphs 259
- 7.3 Traveling Salesman Problem: Independent Model 274
- 7.4 Euclidean Traveling Salesman Problem 282

8

Local Search and Metaheuristics 309

César Rego and Fred Glover

- 8.1 Background on Heuristic Methods 309

<i>Contents</i>	vii	
8.2	Improvement Methods	313
8.3	Tabu Search	345
8.4	Recent Unconventional Evolutionary Methods	355
8.5	Conclusions and Research Opportunities	367
9		
Experimental Analysis of Heuristics for the STSP		369
<i>David S. Johnson and Lyle A. McGeoch</i>		
9.1	Introduction	369
9.2	DIMACS STSP Implementation Challenge	371
9.3	Heuristics and Results	381
9.4	Conclusions and Further Research	438
10		
Experimental Analysis of Heuristics for the ATSP		445
<i>David S. Johnson, Gregory Gutin, Lyle A. McGeoch, Anders Yeo, Weixiong Zhang and Alexei Zverovitch</i>		
10.1	Introduction	446
10.2	Methodology	447
10.3	Heuristics	457
10.4	Results	463
10.5	Conclusions and Further Research	486
11		
Polynomially Solvable Cases of the TSP		489
<i>Santosh N. Kabadi</i>		
11.1	Introduction	489
11.2	Constant TSP and its generalizations	489
11.3	The Gilmore-Gomory TSP (GG-TSP)	494
11.4	GG Scheme: a generalization of Gilmore-Gomory scheme for GG-TSP	506
11.5	Minimum cost connected directed pseudograph problem with node deficiency requirements (MCNDP)	539
11.6	Solvable cases of geometric TSP	547
11.7	Generalized graphical TSP	560
11.8	Solvable classes of TSP on specially structured graphs	564
11.9	Classes of TSP with known compact polyhedral representation	566
11.10	Other solvable cases and related results	576
12		
The Maximum TSP		585
<i>Alexander Barvinok, Edward Kh. Gimadi and Anatolii I. Serdyukov</i>		
12.1	Introduction	585
12.2	Hardness Results	588
12.3	Preliminaries: Factors and Matchings	589
12.4	MAX TSP with General Non-Negative Weights	590
12.5	The Symmetric MAX TSP	591
12.6	The Semimetric MAX TSP	595
12.7	The Metric MAX TSP	597
12.8	TSP with Warehouses	598
12.9	MAX TSP in a Space with a Polyhedral Norm	600
12.10	MAX TSP in a Normed Space	602

12.11	Probabilistic Analysis of Heuristics	605
13		
	The Generalized Traveling Salesman and Orienteering Problems	609
	<i>Matteo Fischetti, Juan-José Salazar-González and Paolo Toth</i>	
13.1	Introduction	609
13.2	The Generalized Traveling Salesman Problem	617
13.3	The Orienteering Problem	642
14		
	The Prize Collecting Traveling Salesman Problem and Its Applications	663
	<i>Egon Balas</i>	
14.1	Introduction	663
14.2	An Application	664
14.3	Polyhedral Considerations	667
14.4	Lifting the Facets of the ATS Polytope	668
14.5	Primitive Inequalities from the ATSP	671
14.6	Cloning and Clique Lifting for the PCTSP.	680
14.7	A Projection: The Cycle Polytope	686
15		
	The Bottleneck TSP	697
	<i>Santosh N. Kabadi and Abraham P. Punnen</i>	
15.1	Introduction	697
15.2	Exact Algorithms	699
15.3	Approximation Algorithms	705
15.4	Polynomially Solvable Cases of BTSP	714
15.5	Variations of the Bottleneck TSP	734
16		
	TSP Software	737
	<i>Andrea Lodi and Abraham P. Punnen</i>	
16.1	Introduction	737
16.2	Exact algorithms for TSP	739
16.3	Approximation Algorithms for TSP	741
16.4	Java Applets	745
16.5	Variations of the TSP	745
16.6	Other Related Problems and General-Purpose Codes	747
	Appendix: A. Sets, Graphs and Permutations	750
	<i>Gregory Gutin</i>	
A-1	Sets	750
A-2	Graphs	750
A-3	Permutations	753
	Appendix: B. Computational Complexity	754
	<i>Abraham P. Punnen</i>	
B-1	Introduction	754
B-2	Basic Complexity Results	756
B-3	Complexity and Approximation	758

<i>Contents</i>	ix
References	761
List of Figures	807
List of Tables	813
Index	817

Preface

The traveling salesman problem (TSP) is perhaps the most well known combinatorial optimization problem. The book “The Traveling Salesman Problem: A guided tour of combinatorial optimization” edited by Lawler, Lenstra, Rinooy Kan and Shmoys provides the state of the art description of the topic up to 1985. Since then, several significant developments have taken place in the area of combinatorial optimization in general and the traveling salesman problem in particular. This warrants the need for an updated book. We discussed this matter with many distinguished colleagues. These consultations culminated in the project of compiling a new book on the TSP. Enthusiastic support from the research community provided us with the courage and motivation to take up this challenging task.

The pattern and style of this new book closely resemble those of its predecessor [548]. In addition to standard and more traditional topics, we also cover domination analysis of approximation algorithms and some important variations of the TSP. The purpose of the book is to serve as a self-contained reference source which updates the book by Lawler et. al [548]. We believe that the book can also be used for specialized graduate and senior undergraduate courses and research projects.

Roughly speaking, the traveling salesman problem is to find a shortest route of a traveling salesperson that starts at a home city, visits a prescribed set of other cities and returns to the starting city. The distance travelled in such a tour obviously depends on the order in which the cities are visited and, thus, the problem is to find an ‘optimal’ ordering of the cities. There are several applications of the TSP that extend beyond the route planning of a traveling salesman. Chapter 1 describes several formulations, applications, and variations of the problem.

As one can see, it does not take much mathematical sophistication to understand many of the formulations of the TSP. However, TSP is a typical ‘hard’ optimization problem and solving very large instances of it is very difficult if not impossible. Nevertheless, recent developments in polyhedral theory and branch-and-cut algorithms have significantly in-

creased the size of instances which can be solved to optimality. Chapters 2–4 provides a thorough description of polyhedral theory, and implementation and testing of exact algorithms. Chapter 2 discusses polyhedral theory and experimental results of branch-and-cut algorithms for the symmetric TSP. Chapter 3 deals with polyhedral results for the asymmetric TSP developed mostly in last 15 years. Chapter 4 deals with implementation and testing of branch-and-bound and branch-and-cut algorithms for the asymmetric version of the TSP.

Despite the fact that significant progress have been made in our ability to solve TSP by exact algorithms, still there are several instances of the problem to be solved to optimality, that are hard for exact algorithms. Moreover, even when an instance is solvable by an exact algorithm, the running time may become prohibitively large for certain applications. In some cases, the problem data of the instance in hand may not be exact, for various reasons, and thus solving such an instance to optimality may not be viable, especially when it takes a significant amount of computational time. Therefore, researchers have investigated a large number of heuristic algorithms, some of which normally produce near optimal solutions. Chapters 5–10 are devoted to various aspects of heuristic algorithms for the TSP.

Chapter 5 presents a compact account on recent developments on approximation algorithms for the geometric TSP in general and Euclidean TSP in particular. Chapter 6 discusses very large, exponential, size neighborhoods for the TSP and domination analysis, a new tool to compare the quality of heuristics. Probabilistic approaches to the TSP are studied in Chapter 7. Probabilistic approaches try to elucidate the properties of typical rather than worst-case instances. Chapter 8 describes well-established and new heuristic approaches to the symmetric TSP. Chapter 9 provides results of extensive computational experiments with numerous heuristic algorithms for the Symmetric TSP. Computational experience with heuristics for the less well-studied Asymmetric TSP is described in Chapter 10.

Chapter 11 is an extensive survey on polynomial time solvable cases of the TSP. Chapters 12–15 are devoted to variations and generalizations of the TSP. Interesting differences between the TSP and its maximization version are described in Chapter 12, where several approximation algorithms for the maximum TSP are presented. Chapter 13 deals with the Generalized TSP and Orienteering Problem. Polyhedral results as well as implementation of exact algorithms, are discussed in detail. The Prize Collecting TSP, where the traveling salesperson needs to visit only part of the prescribed set of cities as long as he/she collected enough “prize”

items, is the topic of Chapter 14. Chapter 15 considers the Bottleneck TSP, where the largest inter-city distance along the route is minimized.

A summary of available TSP software is discussed in Chapter 16. This chapter will be of particular interest to the readers looking for a “quick” way to solve their TSP instances without getting deeply involved with algorithmic ideas and coding techniques. The book has two appendices. Appendix A is a short overview on graphs, sets and permutations for the benefit of readers who want to refresh their knowledge on these topics. Appendix B discusses complexity issues.

Our foremost thanks go to all authors for their enthusiasm and hard work without which this book would not have been completed. Every chapter of this book was read by several researchers who provided comments and suggestions. We would like to thank, among others, Norbert Ascheuer, Rafi Hassin, Gilbert Laporte, Adam Letchford, Francois Margot, Pablo Moscato, K.G. Murty, K.P.K. Nair, Prabha Sharma, Mike Steele, Stefan Voss, and Klaus Wenger for their valuable comments and suggestions. Some of the authors, in particular, Matteo Fischetti, Alan Frieze, Fred Glover, David Johnson, Santosh Kabadi, Andrea Lodi, Denis Naddef, Cesar Rego, Paolo Toth, Anders Yeo and Alexei Zverovitch also participated actively in refereeing chapters and our gratitude goes to each one of them. We highly appreciate the help and support from our publisher especially Gary Folven and Ramesh Sharda. Special thanks are due to John Martindale, for persuading us to publish the book with Kluwer and retaining his patience and optimism as one deadline after another did not materialize.

Contributing Authors

Sanjeev Arora is an Associate Professor of Computer Science at Princeton University. His doctoral dissertation, submitted at UC Berkeley in 1994, was cowinner of ACM's doctoral dissertation award in 1995. He has also received the NSF Career award, and Packard and Sloan fellowships. He was a cowinner of the ACM-Sigact Gödel prize in 2001.

Egon Balas is University Professor and the Lord Professor of Operations Research at Carnegie Mellon University. He received the John von Neumann Theory Prize of INFORMS in 1995, and the EURO Gold Medal in 2001.

Alexander Barvinok is Professor of Mathematics at the University of Michigan. His research interests include computational complexity and algorithms in algebra, geometry and combinatorics.

Matteo Fischetti is Full Professor of Operations Research at the Faculty of Engineering of the University of Padua. In 1987 his Ph.D. dissertation was awarded the first prize *Best PhD Dissertation on Transportation* by the Operations Research Society of America. He is the author of more than 50 papers published in international scientific journals.

Alan Frieze is Professor of Mathematics at Carnegie Mellon University. He was the joint recipient of the 1991 Fulkerson Prize for Discrete Mathematics, awarded jointly by the American Mathematical Society and the Mathematical Programming Society. He is the author of more than 200 scientific papers.

Edward Gimadi is Professor at Novosibirsk State University, PhD (1971), ScD (1988), and Head Researcher at Sobolev Institute of mathematics of Russian Academy of Sciences.

Fred Glover is the MediaOne Chaired Professor in Systems Science at the University of Colorado, Boulder. He has authored or co-authored more than three hundred published articles and four books in the fields of mathematical optimization, computer science and artificial intelligence. Professor Glover is the recipient of the distinguished von Neumann Theory Prize, as well as of numerous other awards and honorary fellowships, including those from the American Association for the Advancement of Science, the NATO Division of Scientific Affairs, the Institute of Management Science, the Operation Research Society, the Decision Sciences Institute, the U.S. Defense Communications Agency, the Energy Research Institute, the American Assembly of Collegiate Schools of Business, Alpha Iota Delta, and the Miller Institute for Basic Research in Science. He serves on the advisory boards of several organizations and is co-founder of OptTek Systems, Inc. and Heuristic, Inc.

Gregory Gutin is Professor of Computer Science at Royal Holloway, University of London. He earned his PhD from Tel Aviv University in 1993. He wrote, together with J. Bang-Jensen, *Digraphs: Theory, Algorithms and Applications*, Springer, London, 2000.

David S. Johnson is the Head of the Algorithms and Optimization Department at AT&T Labs – Research in Florham Park, New Jersey, and has been with the Labs in its various guises since 1973. He and Michael Garey wrote the Lancaster Prize-winning book *Computers and Intractability: A Guide to the Theory of NP-Completeness*. He has been active in the field of experimental analysis of algorithms since 1983, and has overseen the series of DIMACS Implementation Challenges since its inception in 1990.

Santosh Narayan Kabadi is Professor of Management Science at the Faculty of Administration, University of New Brunswick, Fredericton, NB, Canada. He earned his Ph.D from University of Texas at Dallas.

Andrea Lodi is Assistant Professor of Operations Research at the Faculty of Engineering of the University of Bologna. He earned the PhD in System Engineering from the same university in March 2000.

Lyle A. McGeoch is Professor of Computer Science at Amherst College, where he taught since 1987. He earned his Ph.D at Carnegie Mellon University.

Denis Naddef is Professor of Operations Research and Combinatorial Optimization at the Institut National Polytechnique de Grenoble. He received his Ph.D from the University of Grenoble in 1978.

Abraham P. Punnen is Professor of Operations Research at University of New Brunswick, Saint John. He earned his Ph.D at Indian Institute of Technology, Kanpur in 1990. He published extensively in the area of combinatorial optimization and serves on the editorial boards of various journals. He is a member of the board of directors of the Canadian Mathematical Society.

Cesar Rego is Associate Professor at School of Business Administration and a Senior Researcher at the Hearin Center for Enterprise Science, University of Mississippi, USA. He received his Ph.D. in Computer Science from the University of Versailles, France, after earning a MSc in Operations Research and Systems Engineering from the Technical School (IST) of the University of Lisbon. His undergraduate degree in Computer Science and Applied Mathematics is from the Portucalense University in Portugal. His publications have appeared in books on metaheuristics and in leading journals on optimization such as EJOR, JORS, Parallel Computing, and Management Science.

Juan José Salazar González is Associate Professor of the “Departamento de Estadística, Investigación Operativa y Computación”, University of La Laguna (Tenerife, Spain), since 1997. He earned his Ph.D in Mathematics from University of Bologna (Italy) in 1992, and Ph.D in Computer Science from University of La Laguna (Spain) in 1995.

Anatoly Serdyukov was Senior Researcher at Sobolev Institute of Mathematics of Russian Academy of Sciences, PhD (1980). He passed away on 7 February 2001.

Paolo Toth is Full Professor of Combinatorial Optimization at the Faculty of Engineering of the University of Bologna. He is the author of more than 90 papers published in international scientific journals, and co-author of the book *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, 1990. He was President of EURO (Association of the European Operational Research Societies) from 1995 to 1996, and is President of IFORS (International Federation of the Operational Research Societies) for the period 2001-2003.

Anders Yeo is a Lecturer in Computer Science, Royal Holloway, University of London. He earned his PhD from Odense University, Denmark. His main research interests are in graph theory and combinatorial optimization.

Joseph Yukich is Professor of Mathematics at Lehigh University. He is the recipient of two Fulbright awards and has authored the monograph *Probability Theory of Classical Euclidean Optimization Problems*, Lecture Notes in Mathematics, volume 1675, 1998.

Weixiong Zhang is an Associate Professor of Computer Science at Washington University in St. Louis. He earned his PhD from University of California at Los Angeles. He is the author of *State-space search: Algorithms, Complexity, Extensions, and Applications* published by Springer-Verlag in 1999.

Alexei Zverovitch is a PhD student at Royal Holloway, University of London, UK. His supervisor is Gregory Gutin.

Chapter 1

THE TRAVELING SALESMAN PROBLEM: APPLICATIONS, FORMULATIONS AND VARIATIONS

Abraham P. Punnen

Department of Mathematical Sciences

University of New Brunswick-Saint John

New Brunswick, Canada

punnen@unbsj.ca

This chapter is dedicated to the memory of my grandparents
Mr. Korah Abraham and Mrs. Sosamma Abraham

1. Introduction

The traveling salesman problem (TSP) is to find a routing of a salesman who starts from a home location, visits a prescribed set of cities and returns to the original location in such a way that the total distance travelled is minimum and each city is visited exactly once. Although a business tour of a modern day traveling salesman may not seem to be too complex in terms of route planning, the TSP in its generality represents a typical ‘hard’ combinatorial optimization problem.

TSP was known among mathematicians and statisticians under various names. Karl Menger considered a variation of TSP called the *messenger problem* (Botenproblem) [116, 451, 593]. It may be noted that the messenger problem and TSP are equivalent in the sense that one problem can be reduced to the other using very simple transformations. Bock [116] gives an English translation of the description of messenger problem from [593] as follows:

“We designate as the Messenger Problem (since this problem is encountered by every postal messenger, as well as by many travellers) the task

of finding, for a finite number of points whose pairwise distances are known, the shortest path connecting the points. This problem is naturally always solvable by making a finite number of trials below the number of permutations of the given points. The rule, that one should first go from the starting point to the nearest point, then to the point nearest to this etc., does not in general result in the shortest path.”

It is believed that the report by Menger [593] is the first published work on TSP. He pointed out that complete enumeration is a possible strategy to compute an optimal solution but a ‘nearest neighbor’ algorithm does not guarantee an optimal solution.

Mahalanobis [575] presented a talk at the Indian Science Congress in 1939 on the problem of estimating the area under jute (*Corchorus Capsularis*) farm in Bengal (currently spread over West Bengal in India and Bangladesh). The objective is to keep the cost involved in the land survey operations below a threshold level while maximizing the accuracy of the result. As part of the cost calculations, he was interested in a least cost routing through a finite number of points in the Euclidean plane. Although no attempt was reported in calculating the least possible cost for such a traversal, it is clear from the discussions that Mahalanobis was aware of the difficulties involved in finding such a minimum cost routing. His cost calculations however are based on expected values considering random points in the plane which forms a small part of a more general cost function.

In a 1955 paper, Morton and Land [608] indicated that “the TSP may be familiar to statisticians as *the mean minimum distances problem*” by citing supporting references [358, 458, 583]. In their brief historic account on TSP, Morton and Land wrote:

“In the United States this problem is known as the Traveling-Salesman problem; the salesman wishes to visit one city in each of the 48 States and Washington, D.C, in such a sequence as to minimize the total road distance traveled. Thinking of the problem independently and on a smaller geographical scale, we used to call it the laundry van problem, where the conditions were a daily service by a one-van laundry. Since the American term was used by Robinson (1949) we propose to adopt it here.”

The name “traveling salesman problem” for the optimization problem of our discussion is believed to have originated in the United States. Perhaps the first report using this name was published in 1949 [725]. However, it would be reasonable to say that a systematic study of the TSP as a combinatorial optimization problem began with the work of Dantzig, Fulkerson, and Johnson [239]. The survey papers [474, 590, 592, 591] and the books [548, 711] summarize various developments on the subject.

Let us now give a formal mathematical definition of the TSP. For details of basic terminology and notations in graph theory we refer to Appendix A. Let $G = (V, E)$ be a graph (directed or undirected) and \mathbb{F} be the family of all Hamiltonian cycles (tours) in G . For each edge $e \in E$ a cost (weight) c_e is prescribed. Then the traveling salesman problem is to find a tour (Hamiltonian cycle) in G such that the sum of the costs of edges of the tour is as small as possible.

Without loss of generality, we assume that G is a complete graph (digraph), for otherwise we could replace the missing edges with edges of very large cost. Let the node set $V = \{1, 2, \dots, n\}$. The matrix $C = (c_{ij})_{n \times n}$, is called the cost matrix (also referred to as the distance matrix or weight matrix), where the $(i, j)^{th}$ entry c_{ij} corresponds to the cost of the edge joining node i to node j in G .

The TSP can also be viewed as a permutation problem. Generally, two variations of the permutation representation of TSP are in use; a *quadratic representation* and a *linear representation*. Let \mathbb{P}_n be the collection of all permutations of the set $\{1, 2, \dots, n\}$. Then the TSP is to find a $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ in \mathbb{P}_n such that $c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$ is minimized. In this case $(\pi(1), \pi(2), \dots, \pi(n))$ gives the order in which cities are visited, starting with city $\pi(1)$. For example, if $n = 5$ and $\pi = (2, 1, 5, 3, 4)$ then the corresponding tour will be $(2, 1, 5, 3, 4, 2)$. Every cyclic shift of π also give the same tour. Thus there are n different permutations which represent the same tour. This representation of TSP is generally used in the context of some sequencing problems and gives rise to a binary quadratic programming formulation of the TSP (see Section 4). Thus, we refer to this the *quadratic permutation representation* of TSP. Chapter 12 of this book uses this quadratic permutation representation of the TSP.

In another permutation representation of TSP, referred to as *linear permutation representation*, only special types of permutations, called cyclic permutations, are considered feasible. Let \mathbb{C}_n be the collection of all cyclic permutations of $\{1, 2, \dots, n\}$. Then the TSP is to find a $\sigma = (\sigma(1), \sigma(2), \dots, \sigma(n)) \in \mathbb{C}_n$ such that, $\sum_{i=1}^n c_{i\sigma(i)}$ is minimized. Under this representation, $\sigma(i)$ is the successor of city i in the resulting tour, for $i = 1, 2, \dots, n$. For example if $n = 5$ and $(\sigma(1), \sigma(2), \dots, \sigma(5)) = (2, 4, 5, 3, 1)$, the resulting tour is given by $(1, 2, 4, 3, 5, 1)$. This representation of TSP leads to a binary liner programming formulation of TSP and hence we call it the *linear permutation representation*. Chapter 11 and part of Chapter 15 use the linear permutation representation of the TSP.

Depending on the nature of the cost matrix (equivalently the nature of G), TSP is divided in to two classes. If C is symmetric (i.e. G is

undirected) then the TSP is called the symmetric traveling salesman problem (STSP). If C is not necessarily symmetric (equivalently the graph G is directed) then it is called the asymmetric traveling salesman problem (ATSP). Since every undirected graph can be viewed as a directed graph, by duplicating edges one in the forward direction and the other in the backward direction, STSP can be viewed as a special case of ATSP. Interestingly, it is also possible to formulate ATSP as a STSP [471] by doubling the number of nodes. Consider the ATSP on a digraph $D = (N, A)$ with c_{ij} as the cost of arc (i, j) . Construct the undirected graph $G^* = (V^*, E^*)$ with node set $V^* = N \cup \{1^*, 2^*, \dots, n^*\}$ where $N = \{1, 2, \dots, n\}$. For each arc (i, j) in D , create an edge (i, j^*) in G^* with cost c_{ij} . Also introduce n dummy edges (i, i^*) of cost $-M$ where M is a very large number. All other edges have cost M . It can be verified that solving the ATSP on D is equivalent to solving the STSP on G^* .

1.1. The Shoelace Problem - A simple TSP

According to Bata shoe museum (Toronto, Canada) evidence of shoe making exists as early as 10,000 BC; the average person walks the equivalent of three-and-a-half times around the earth in a life time; and North Americans spend almost \$18 billion a year on footwear! Having a comfortable pair of shoes doesn't ensure comfortable walking. Proper lacing of the shoe is also important in increasing foot comfort and perhaps increasing the life span of the shoe. Needless to say, proper lacing of shoe is important for a traveling salesman. Our salesman however is interested in lacing his shoes with a lace of minimum length! Optimal shoe-lacing as a mathematical entertainment problem was introduced by Halton [431]. It can be described as follows:

Let $A_0, A_1, A_2, \dots, A_n$ and $B_0, B_1, B_2, \dots, B_n$ be the eyelets of a shoe arranged on two parallel lines. (See Fig: 1.1).

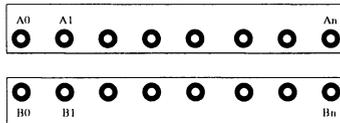


Figure 1.1. Eyelets on a shoe

A feasible lacing is a 'path' from A_0 to B_0 visiting each eyelet exactly once, choosing A_i 's and B_j 's alternately, and closing the path by moving from B_0 to A_0 . It can be represented as an ordered pair (π, σ) of two

permutations π and σ of $\{1, 2, \dots, n\}$. Note that (π, σ) represents the unique lacing

$$A_0 \rightarrow B_{\sigma(1)} \rightarrow A_{\pi(1)} \dots \rightarrow A_{\pi(n-1)} \rightarrow B_{\sigma(n)} \rightarrow A_{\pi(n)} \rightarrow B_0 \rightarrow A_0.$$

The length of a feasible lacing (π, σ) , denoted by $L(\pi, \sigma)$, is given by

$$\begin{aligned} & \sum_{i=1}^{n-1} [d(\sigma(i), \pi(i)) + d(\pi(i), \sigma(i+1))] + d(0, \sigma(1)) + d(\sigma(n), \pi(n)) \\ & + d(\pi(n), 0) + d(0, 0), \end{aligned}$$

where $d(i, j)$ is the Euclidean distance between A_i and B_j . Let the distance between two consecutive eyelets on the same side be α and the distance between A_i and B_i be β . Then, it can be verified that

$$\begin{aligned} L^{\alpha\beta}(\pi, \sigma) &= \sqrt{\sigma(1)^2\alpha^2 + \beta^2} + \sqrt{\pi(n)^2\alpha^2 + \beta^2} + \sqrt{(\pi(n) - \sigma(n))^2\alpha^2 + \beta^2} \\ &+ \sum_{i=1}^{n-1} (\sqrt{(\pi(i) - \sigma(i))^2\alpha^2 + \beta^2} + \sqrt{(\sigma(i+1) - \pi(i))^2\alpha^2 + \beta^2}) + \beta \end{aligned}$$

The *shoelace problem* $SP(\alpha, \beta)$ is to find a feasible lacing $(\bar{\pi}, \bar{\sigma})$ of minimum length $L^{\alpha\beta}(\bar{\pi}, \bar{\sigma})$. Note that there are $(n!)^2$ feasible lacing patterns. Interestingly we will see that one of the most commonly used lacing patterns is indeed optimal. Let us consider four special lacing patterns (π, σ) . When $\pi = (2, 4, \dots, 5, 3, 1)$ and $\sigma = (5, 3, 1, \dots, 2, 4)$, (π, σ) is called a *zig-zag lacing*. When $\pi = (2, 4, \dots, 3, 1)$ and $\sigma = (1, 2, 4, 6 \dots 5, 3)$ we have a *straight lacing*. *Quick lacing* corresponds to $\pi = (n, n-1, \dots, 3, 2, 1)$ and $\sigma = (n, n-1, \dots, 2, 1)$ and *diagonal lacing* corresponds to $\pi = (1, 2, 3 \dots n-1, n)$ and $\sigma = (n, n-1, \dots, 2, 1)$.

It can be verified that the zig-zag lacing is of length $2(\beta + n\sqrt{\alpha^2 + \beta^2})$, straight lacing is of length $(n+1)\beta + 2\sqrt{\beta^2 + \alpha^2} + (n-1)\sqrt{4\alpha^2 + \beta^2}$, quick lacing is of length $(n+1)\beta + n\sqrt{\alpha^2 + \beta^2} + \sqrt{n^2\alpha^2 + \beta^2}$, and diagonal lacing is of length $2(\beta + \sum_{i=1}^{n-1} \sqrt{(n-i)^2\alpha^2 + \beta^2})$. By choosing $n = 10, \alpha = 1$ and $\beta = 2$, the zig-zag, straight, quick, and diagonal lacings have approximate lengths 48.72, 51.92, 54.55, and 103.19, respectively. Thus the zig-zag lacing comes out to be the winner. Indeed, it can be shown that,

Theorem 1 [431] *For $n \geq 3$, zig-zag lacing is the unique optimal solution to the shoelace problem $SP(\alpha, \beta)$.*

For $n = 2$, all four lacing patterns discussed earlier have the same length. So far we have assumed that $A_0, A_1, A_2, \dots, A_n$ are arranged

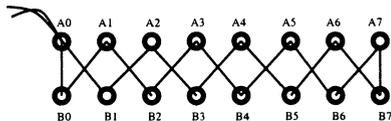


Figure 1.2. Zig-zag Lacing

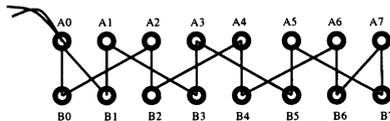


Figure 1.3. Straight Lacing

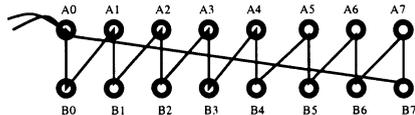


Figure 1.4. quick Lacing

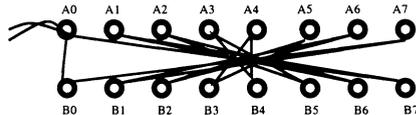


Figure 1.5. diagonal Lacing

on a straight line and $B_0, B_1, B_2, \dots, B_n$ are arranged on a parallel line. However, this assumption can be relaxed. If $A_i A_j B_k B_l$ for all $i < j$ and $k > l$ forms a convex quadrilateral, then also the zig-zag lacing can be shown to be optimal [600, 789]. What can we say about the diagonal lacing? Is it the longest? We leave this question for the reader.

Choosing the eyelets A_0, A_1, \dots, A_n and B_0, B_1, \dots, B_n as nodes of a complete bipartite graph where the edge costs correspond to the Euclidean distances between these points except for the edge (A_0, B_0) the cost of which is $-M$ where M is a very large number, the shoelace problem can be considered as a TSP restricted to a complete bipartite graph which in turn can be viewed as traveling salesman problem on a complete graph by adding edges of large cost.

Although we have a closed form solution for this special case, TSP in general is NP-hard. (For a summary of computational complexity results on TSP, we refer to Appendix B.) This inherent difficulty of the problem and its various practical applications inspired researchers and practitioners to investigate various aspects of the problem. In particular, significant developments have taken place in the recent past in the study of structural properties of the problem, development of efficient exact and approximation algorithms, and identification of efficiently solvable special cases. In the following chapters we study each of these aspects of TSP and its variations. In the rest of this chapter, we will concentrate on applications, variations and mathematical formulations of the problem.

2. Simple Variations of the TSP

Several variations of the TSP that are studied in the literature have been originated from various real life or potential applications. Let us first consider some of these variations that can be reformulated as a TSP using relatively simple transformations.

The MAX TSP: Unlike the TSP, the objective in this case is to find a tour in G where the total cost of edges of the tour is maximum. MAX TSP can be solved as a TSP by replacing each edge cost by its additive inverse. If we require that the edge costs are to be non-negative, a large constant could be added to each of the edge-costs without changing the optimal solutions of the problem. For details on MAX TSP, we refer to Chapter 12.

The bottleneck TSP: In the case of a bottleneck TSP, the objective is to find a tour in G such that the largest cost of edges in the tour is as small as possible. A bottleneck TSP can be formulated as a TSP with exponentially large edge costs. This problem is studied in detail in Chapter 15.

TSP with multiple visits (TSPM): Here we want to find a routing of a traveling salesman, who starts at a given node of G , visits each node *at least* once and comes back to the starting node in such a way that the total distance traveled is minimized. This problem can be transformed into a TSP by replacing the edge costs with the shortest path distances in G . In the absence of negative cycles, shortest path distances between all pairs of nodes of a graph can be computed using efficient algorithms [7]. If G contains a negative cycle, then TSPM is unbounded. Several shortest path algorithms are capable of detecting negative cycles in a graph [7].

Messenger problem: This problem[593] is also known as the wandering salesman problem [474]. Given two specified nodes u and v in G , we want to find a least cost Hamiltonian path in G from u to v . This can be solved as a TSP by choosing a cost of $-M$ for the edge (v, u) where M is a large number. If no nodes are specified, and one wishes to find a least cost Hamiltonian path in G , it can also be achieved by solving a TSP. Modify the graph G by creating a new node and connecting it with all the nodes of G by edges (for directed graphs, introduce two arcs, one in forward and other in backward directions) of cost $-M$. From an optimal solution to the TSP on this modified graph, an optimal solution

to the original problem can be recovered.

Clustered TSP: In this case, the node set of G is partitioned into clusters V_1, V_2, \dots, V_k . Then the clustered TSP [426, 469] is to find a least cost tour in G subject to the constraint that cities within the same cluster must be visited consecutively. This problem can be reduced to a TSP by adding a large cost M to the cost of each inter-cluster edge.

Generalized TSP (GTSP): As in the previous case, let V_1, V_2, \dots, V_k be a partition of the node set of G . In a GTSP, we want to find a shortest cycle in G which passes through exactly one node from each cluster V_i , $1 \leq i \leq k$. If $|V_i| = 1$ for all i , GTSP is the same as TSP. Using the reduction described in [629] we now show that GTSP can be reduced to a TSP for arbitrary $|V_i|$'s. Without loss of generality, assume that G is a digraph and the partitions are numbered in such a way that $|V_i| \geq 2$ for $1 \leq i \leq r$ and $|V_i| = 1$ for $r + 1 \leq i \leq k$. For any $i \leq r$ let $V_i = \{i_1, i_2, \dots, i_{n_i}\}$. For each arc $e \in E$, consider a new cost d_e defined as follows:

$$d_{i_j i_{j+1}} = -M, \quad j = 1, \dots, n_i \text{ with } n_i + 1 \equiv 1, i = 1, \dots, r$$

This ensures that if an optimal TSP tour in G enters a cluster V_i through node i_j , it visits vertices of V_i in the order $i_j, i_{j+1}, \dots, i_{n_i}, i_1, \dots, i_{j-1}$ and leaves the cluster V_i from the node i_{j-1} . We want to translate this outcome equivalent to a GTSP tour entering and leaving the cluster V_i by visiting just node i_j . To replicate this fact, we make the new cost of arcs going out of i_{j-1} corresponds to the original cost of arcs leaving i_j . More precisely,

$$d_{i_{j-1}p} = c_{i_j p}, \quad p \notin V_i, j = 1, 2, \dots, n_i \text{ with index } 0 \equiv n_i, i = 1, \dots, r$$

and $d_{uv} = c_{uv}$ for all other edges. From an optimal solution to the TSP on G with the modified costs d_e for $e \in E$, an optimal solution to GTSP can be recovered. GTSP is discussed in detail in Chapter 13.

The m -salesmen TSP: Assume that m salesmen are located at node 1 of G . Starting from node 1, each salesman visits a subset X_i of nodes of G exactly once and returns to node 1. We are interested in finding a partition X_1, X_2, \dots, X_m of $V - \{1\}$ and a routing of each of the m salesmen such that (i) $|X_i| \geq 1$ for each i , (ii) $\cup_{i=1}^m X_i = V - \{1\}$, (iii) $X_i \cap X_j = \emptyset$ for $i \neq j$, and (iv) total distance travelled by all the salesmen is minimized [474]. If G is undirected, the m -salesmen TSP can be formulated as a TSP on a graph $G' = (V', E')$ with $m - 1$ additional nodes, where $V' = V \cup \{n+2, n+3, \dots, n+m\}$, $E' = E \cup \{(i, n+k) : 2 \leq$

$k \leq m, 2 \leq i \leq n\}$. Let c'_{ij} be the cost of edges in G' , where $c'_{ij} = c_{ij}$ if $(i, j) \in E$, and $c'_{i,n+k} = c_{i1}$, for $2 \leq k \leq m, 2 \leq i \leq n$. From an optimal TSP solution on G' an optimal solution to the m -salesmen TSP can be recovered. The case when G is a directed graph can be handled similarly with simple modifications in the above transformation. For variations of the m -salesmen TSP that can be formulated as a TSP we refer to [96, 695].

3. Applications of TSP

Applications of the TSP and its variations go way beyond the route planning problem of a traveling salesman and spans over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering, and electronics. In this section, we discuss some selected applications of TSP.

3.1. Machine Scheduling Problems

Perhaps the most well studied application area of the TSP is machine sequencing and scheduling. A simple scheduling application can be described as follows. There are n jobs $\{1, 2, \dots, n\}$ to be processed sequentially on a machine. Let c_{ij} be the set up cost required for processing job j immediately after job i . When all the jobs are processed, the machine is reset to its initial state at a cost of c_{j1} , where j is the last job processed. The sequencing problem is to find an order in which the jobs are to be processed such that the total setup cost is minimized. Clearly, finding a permutation π of $\{1, 2, \dots, n\}$ that minimizes $c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$ solves the problem.

In many practical applications, the jobs are not arbitrary but often can be clustered together so that the setup time, if any, between jobs within a cluster is relatively small compared to setup time between jobs in two different clusters. This is a typical scenario in assembly line. Assembling similar products need minimal setup time in between; but if a different product needs to be assembled, the setup time may be larger as one may require new parts, tools etc. The sequencing problem with such a specialized cost matrix reduces to the clustered TSP. This cluster property can be effectively exploited in developing efficient algorithms. Ozgur and Brown [639] developed a two stage heuristic for such a problem originated from the mechanical push-pull cable control systems.

Let us now consider another scheduling problem - a no wait flow shop problem. There are n jobs each requiring processing on m machines in the order $1, 2, 3, \dots, m$. No job is allowed to have a waiting time between

processing on two consecutive machines. The objective is to find an optimum sequencing of jobs to be processed so that the makespan (total completion time) is minimum. Applications of this type of sequencing problems arise in a variety of situations [272]. The no-wait flow shop problem is strongly NP-hard, but solvable in polynomial time when $m = 2$ in which case it reduces to the well known Gilmore-Gomory problem. (See Chapter 11 for more details on this.)

From an instance of the no-wait flow shop problem, we construct an equivalent TSP instance using the reduction proposed in [272]. Create a complete directed graph G on $n + 1$ nodes, where node j corresponds to job j , $1 \leq j \leq n$ and node $n + 1$ represents both the start and end of processing. The cost c_{ij} of arc (i, j) in G represents the additional schedule length if job j is the immediate successor of job i in the schedule. Thus a minimum cost tour in G corresponds to a schedule with minimum makespan. To complete the reduction, we must identify the values of c_{ij} . Let p_{jk} be the processing time of job j on processor k , $1 \leq j \leq n$, $1 \leq k \leq m$. Then c_{ij} can be obtained [272] using the equations,

$$c_{n+1,i} = \sum_{r=1}^m p_{ir}, \quad i = 1, 2, \dots, n$$

$$c_{ij} = \max_{1 \leq k \leq m} \left\{ \sum_{r=1}^k p_{ir} + \sum_{r=k}^m p_{jr} \right\} - \sum_{r=1}^m p_{ir}, \quad 1 \leq i, j \leq n, i \neq j$$

$$c_{i,n+1} = c_{ii} = 0, \quad i = 1, 2, \dots, n.$$

The applications we considered so far reduce a given problem to the TSP. Let us now discuss another application of TSP where it is used as a subroutine to develop efficient heuristics for a more complex problem involved in the control of a pick and placement machine. This application is taken from Ahmadi and Mamer [4].

A pick-and-place machine includes a work platform, a placement arm, pick and place heads, component delivery carrier (feeder magazines), and tool magazines. Generally these types of machines are used in printed circuit board assembly [4, 83] and available in different architectures such as the placement arm is moving while work platform remains stationary or the placement arm remains stationary while the work platform is moving or a combination of the two. In our model the placement arm moves between two fixed locations - the pick up position and the delivery position. The work platform moves in the x and y directions and feeder-tool carrier moves in the x direction only. The head attached to the placement arm needs to be changed if a different component type is to be placed on the board. Normally tool change operations are more time

consuming in comparison to pick and placement operations. The cycle time - i.e. the time required to move the placement arm between the pickup and delivery positions, perform a pick up operation and perform a delivery operation- is assumed to be constant. Let p_i be the position of the i^{th} placement site, c_{ij} be the time required to move the placement board from site p_i to site p_j . When the placement head moves to the pickup position, simultaneously the work platform moves to adjust the next placement site. Let t_{ij} denote the time lost in switching between part i and part j and N_k denote the index set corresponding to placement sites requiring part type k , $k = 1, 2, \dots, m$. It is assumed that only one component is placed at a given placement site and the p_i 's are numbered consecutively with the part type. Thus $N_1 = \{p_1, p_2, \dots, p_{k_1}\}$, $N_2 = \{p_{k_1+1}, p_{k_1+2}, \dots, p_{k_1+k_2}\}$, \dots . Define $n = \sum_{i=1}^m k_i$. The calculation of c_{ij} and t_{ij} depends on the architecture of the machine. The pickup and placement problem is to find a tour through p_1, p_2, \dots, p_n which minimizes the total time to complete all placements, including the time required to start and return the machine to the resting position.

Although this problem is not exactly a TSP, its close relationship with TSP allows us to develop efficient heuristics by using TSP as a subroutine or by appropriate modifications of the TSP heuristics itself. Ahmadi and Mamer [4] developed a two stage heuristic for this problem using a TSP solver (exact or heuristic) as a subroutine. For details of other research works on printed circuit board assembly that are relevant to the TSP, we refer to [83].

3.2. Cellular Manufacturing

In a cellular manufacturing system families of parts (products) that require similar processing are grouped and processed together in a specialized cell to achieve efficiency and cost reductions. In such systems robots are often used in handling material to improve the efficacy of the cell. Aneja and Kamoun [23] considered the problem of sequencing the robot activities in a two machine robotic cell and showed that the problem can be formulated as a TSP. For some fundamental results on this problem, we refer to [758].

Let p_1, p_2, \dots, p_n be n part-types to be produced. Each part type p_i needs to be processed on two machines M1 and M2 with M1 followed by M2. Let a_i be the processing time of p_i on M1 and b_i be the processing time of p_i on M2. A robot picks up a part from an input storage buffer, loads it on to machine M1, collects from M1 and then loads to M2, collects from M2 and places the product in the output storage buffer. The storage buffers and machines are arranged on a straight line so that

the robot travels only along a straight line. The time taken by the robot for movement between input storage buffer and each machine, between each machine and output storage buffer, and between the machines is given by t . Let l be the time taken by the robot in loading or unloading.

Two robotic movement cycles $C1$ and $C2$ with a point of intersection are considered [758]. Let π be a permutation of $\{1, 2, \dots, n\}$ representing the sequence in which the parts are selected for processing. In cycle $C1$, starting from an initial state, the robot loads a part, say $p_{\pi(i)}$ on M2, waits for it to complete, picks up the processed part from M2 and places it in the output storage buffer, goes back to the input storage buffer, picks up part $p_{\pi(i+1)}$ and moves to M2. The cycle time in this case is given by

$$T_{\pi(i)\pi(i+1)}^1 = 6(l + t) + b_{\pi(i)} + a_{\pi(i+1)}.$$

For cycle $C2$, starting from an initial state, the robot loads $p_{\pi(i)}$ on M2, moves to the input storage buffer, picks up part $p_{\pi(i+1)}$ and loads on to M1, goes to M2 and waits (if necessary) to complete the processing of $p_{\pi(i)}$ on M2, picks up part $p_{\pi(i)}$ from M2 and moves to the output storage buffer to unload the part, moves back to M1 and wait (if necessary) to complete the processing of $p_{\pi(i+1)}$ on M1, picks up $p_{\pi(i+1)}$ and moves to M2. The cycle time in this case can be calculated as

$$T_{\pi(i)\pi(i+1)}^2 = 8t + 6l + w_1(i + 1) + w_2(i)$$

where $w_1(i + 1)$, the waiting time at M1, is given by $\max\{0, a_{\pi(i+1)} - (4t + 2l + w_2(i))\}$ and $w_2(i)$, the waiting time at M2, is given by $\max\{0, b_{\pi(i)} - (4t + 2l)\}$. Let $X_i = b_i + 2(t + l)$ and $Y_i = a_i + 2(t + l)$. Then T_{ij}^1 and T_{ij}^2 can be written as

$$T_{ij}^1 = 2(t + l) + X_i + Y_j$$

and

$$T_{ij}^2 = 2(t + l) + \max\{X_i + Y_i, 6t + 2l\}.$$

Note that each cycle $C1$ or $C2$ involves two parts $p_{\pi(k)}$ and $p_{\pi(k+1)}$ for some k , under the schedule π . For the cycles $C1$ and $C2$, the state in which the robot completed loading a part on M2 is common. Thus at such a state the robot needs to make a decision which cycle to be used. Clearly, if $T_{\pi(i)\pi(i+1)}^1 < T_{\pi(i)\pi(i+1)}^2$, cycle $C1$ is selected, otherwise $C2$ is selected. Now consider the cost matrix $C = (c_{ij})_{n \times n}$ where $c_{ij} = \min\{T_{ij}^1, T_{ij}^2\}$. The optimal TSP tour using cost matrix C provides an optimal part sequencing. This TSP can be solved in $O(n \log n)$ time exploiting the special structure the matrix C (see [23] and Chapter 11).

Instead of the straight line topology for robotic movements, other structures could be considered. For example, M1, M2, the input storage buffer, and the output storage buffer are placed at 4 corners of a rectangle and the robot moves along the perimeter of this rectangle choosing shortest path distances, the resulting part sequencing problem can also be formulated as a TSP.

3.3. Arc Routing Problems

A general arc routing problem, known as mixed windy rural postman problem (MWRPP), can be stated as follows. Let $G = (V, A \cup E)$ be a mixed graph where elements of A are arcs (directed) and elements of E are edges (undirected). Let $A' \subset A$ and $E' \subset E$. The arc and edge costs are assumed to be non-negative. The MWRPP is to find a minimum cost closed walk on G containing all arcs in A' and all edges in E' . Several problems such as mixed Chinese postman problem, windy Chinese postman problem, windy rural postman problem, stacker crane problem, etc. are special cases of MWRPP. Applications of MWRPP also include street sweeping, snow plowing, etc. and the problem is known to be NP-hard. We observe that MWRPP can be solved as an asymmetric TSP. This transformation is taken from Laporte [534].

In G , replace each undirected edge (i, j) by two arcs (i, j) and (j, i) with cost equal to that of (i, j) . Let $\bar{G} = (V, \bar{A})$ be the resulting digraph. Define $\bar{A}' = A' \cup \{(i, j), (j, i) \in \bar{A} : (i, j) \in E'\}$. We now formulate a generalized TSP on the digraph $D = (U, Z)$. For each arc $(i, j) \in \bar{A}'$ there is a node u_{ij} in U . The cost of arc $(u_{ij}, u_{pq}) \in Z$ is defined as the length of the shortest path from node i to node q in \bar{G} . Now the MWRPP is equivalent to a GTSP on D where the node set partition $\{U_1, U_2, \dots, U_m\}$ is given by $U_l = \{u_{ij}\}$ if $(i, j) \in A'$, $l = 1, 2, \dots, |A'|$ and $U_l = \{u_{ij}, u_{ji}\}$ if $(i, j) \in E'$, $l = |A'| + 1, \dots, |A'| + |E'|$. This GTSP can be reduced to a TSP as discussed earlier. Laporte [534] reports experimental results on several arc routing problems using such transformations.

3.4. Frequency Assignment Problem

In a communication network with a collection of transmitters, the frequency assignment problem is to assign a frequency to each transmitter from a given set of available frequencies satisfying some interference constraints. These constraints can be represented by a graph $G = (V, E)$ where each node i represents a transmitter. A non-negative integer weight c_{ij} is prescribed for each arc (i, j) representing the tolerance. Let $F = \{0, 1, 2, \dots, R\}$ be a collection of allowable frequencies.

A frequency assignment is to assign number $f(i) \in F$ to node $i \in V$ such that $|f(i) - f(j)| > c_{ij}$ for all $(i, j) \in E$. If such an assignment exists, it is called a feasible assignment. If R is sufficiently large a feasible assignment is always possible. The minimum value of R for which a feasible assignment for G exists is called the span of G and is denoted by $Span(G)$.

Let G^* be the complete graph obtained from G by adding (if necessary) edges of zero weight. Let c'_{ij} be the weight of edge (i, j) in G^* such that $c'_{ij} = c_{ij} + 1$. Let $C'(H^*)$ be the sum of the weights of edges in a minimum cost Hamiltonian path in G^* . Smith and Hurly [765] showed that $Span(G) \geq C'(H^*)$. Thus TSP can be used to compute a lower bound for the frequency assignment problem. Successful applications of the TSP based bounds and its variations in solving frequency assignment problem are discussed in [17].

3.5. Structuring of Matrices

Let A be a finite set and $f : A \times A \rightarrow \mathbb{R}$. Consider the matrix $X = (x_{ij})_{m \times n}$ where $x_{ij} \in A$. For each row i of X , let $L_i(X) = \sum_{j=1}^n f(x_{ij}, x_{i(j+1)})$ where $x_{i(n+1)} = x_{i1}$. Similarly for each column j , let $C_j(X) = \sum_{i=1}^m f(x_{ij}, x_{(i+1)j})$ where $x_{(m+1)j} = x_{1j}$. Define $L(X) = \sum_{i=1}^m L_i(X)$ and $C(X) = \sum_{j=1}^n C_j(X)$. Let $S(X)$ be the family of matrices obtained by permuting rows of X . For $Y, Z \in S(X)$, Y is preferable to X if $C(Y) < C(Z)$. The set $B = \{Y \in S(X) : C(Y) = \min_{Z \in S(X)} C(Z)\}$ is called matrices with best row structure with respect to X . Structuring of rows of a matrix X corresponds to identifying an element of B . This problem can be solved by solving a TSP on the complete graph $K_m = (V, E)$ where $V = \{1, 2, \dots, m\}$ with the weight of edge (i, j) is given by $c_{ik} = \sum_{j=1}^n f(x_{ij}, x_{kj})$ [516, 555]. In a similar way, structuring of columns of a matrix can be solved as a TSP. Applications structuring of rows/columns include clustering of data [555], phsyotgenology, and theory of group decisions [723].

3.6. Additional Applications

We now give some additional references to more applications of TSP. Korostensky et al [513] used the traveling salesman problem to compute a near optimal *multiple sequence alignment* with guaranteed performance bound. In a related work Korostensky et al [514] used TSP based methods in constructing an *evolutionary tree* of optimal 'score'. The traveling salesman model is applicable in a variety of other situations including data analysis in psychology [454], X-Ray crystallography [111], overhauling gas turbine engines [670], warehouse order-picking problems [699],

and wall paper cutting [348], to cite a few. Marchand et al [580] formulated an optimal control problem as a TSP. Potential application areas of their model include problems that arise in routing in telecommunication networks, test objective generation, and artificial intelligence [580].

4. Alternative representations of the TSP

We have introduced TSP as the problem of finding a least cost Hamiltonian cycle in a graph (graphical definition). We also discussed quadratic and linear permutation representations of TSP. In this section we study representations of the TSP as a mathematical programming problem.

4.1. Linear programming representation

This is perhaps the most well studied representation of the TSP. Let \mathbb{F} be the family of all tours in G represented as a collection of incidence vectors in $\mathbb{R}^{|E|}$ and $P(\mathbb{F})$ be the convex hull of \mathbb{F} . Then the traveling salesman problem can be stated as the following mathematical programming problem:

$$\begin{array}{ll} \text{Minimize} & \sum_{j=1}^m c_j x_j \\ \text{Subject to} & X \in P(\mathbb{F}), \end{array}$$

where $X = (x_1, x_2, \dots, x_m)$, m being the number of edges in G .

A complete linear inequality description of $P(\mathbb{F})$ is not known which somewhat limits application of linear programming techniques to solve TSP. However, it may be noted that linear programming based methods are among the best known approaches for solving TSP which exploits partial linear inequality description of $P(\mathbb{F})$. For the state of the art research in this area we refer to chapters 2 and 3.

4.2. Integer programming formulations

Let us first consider the case of the ATSP. It may be noted that the linear programming relaxation of the integer programming formulations discussed below could be strengthened by adding 'valid' inequalities corresponding to $P(\mathbb{F})$, especially those defining facets or higher dimensional faces of $P(\mathbb{F})$. For a discussion of such inequalities we refer to chapters 2 and 3. Most of the integer programming formulations of the ATSP are based on the assignment problem. A 0-1 programming

formulation of the assignment problem can be described as follows:

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \\ & \text{Subject to} \\ & \quad \sum_{i=1}^n x_{ij} = 1, \quad j = 1, \dots, n, \\ & \quad \sum_{j=1}^n x_{ij} = 1, \quad i = 1, \dots, n, \\ & \quad x_{ij} = 0 \text{ or } 1 \end{aligned}$$

For any solution $X = (x_{ij})_{n \times n}$ of the above assignment problem, consider the set $S = \{(ij) : x_{ij} = 1\}$. Clearly, S represents a tour or a collection of subtours in G . In order to model the ATSP as an integer program, we must include additional restrictions to make sure that S does not contain any subtours. These restrictions are called *subtour elimination constraints*.

4.2.1 Clique Packing Constraints. Introduced by Dantzig, Fulkerson, and Johnson [239], these are the most popular form of subtour elimination constraints. It states that from any clique Q of the subgraph G^* of G induced by the node set $V^* = \{2, 3, \dots, n\}$, select at most $|Q| - 1$ arcs. There are an exponential number of such constraints which are given by

$$\sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1 \text{ for all } \emptyset \neq Q \subset \{2, 3, \dots, n\}$$

The clique packing constraints can be written in an equivalent form as

$$\sum_{i \in Q} \sum_{j \in \bar{Q}} x_{ij} \geq 1 \text{ for all } \emptyset \neq Q \subset \{2, 3, \dots, n\}$$

where $\bar{Q} = V - Q$.

4.2.2 Circuit Packing Constraints. This class of subtour elimination constraints were introduced by Grotschel and Padberg [401]. (See also [392].) These constraints can be described by replacing cliques Q in the clique packing constraints by circuits in G^* . Again there are an exponential number of circuit packing constraints.

The assignment problem together with clique packing or circuit packing constraints leads to a valid 0-1 programming formulation of the

ATSP. These formulations use only variables that correspond to the arcs and self-loops of the complete graph G and are called *natural formulations* [676]. It may be noted that the self-loop variables x_{ii} could be discarded from the formulation to reduce the problem size. In this case one needs to consider only cliques (circuits) of size two or more in the clique (circuit) packing constraints, further reducing the problem size. We have used the variables x_{ii} in our formulation in order to be consistent with the definition of the classical assignment problem. One could also choose $c_{ii} = M$, where M is a large positive number, to insure they will not appear in any optimal solution to the assignment problem. In this case also it is sufficient to consider clique (circuits) of size two or more within the clique (circuit) packing constraints.

By using polynomial (in n) number of additional constraints and variables it is possible to ensure that S is cycle free. These are called *compact representations* of the subtour elimination constraints.

4.2.3 MTZ constraints. Miller, Tucker and Zemlin [595] showed that by using $(n - 1)^2$ additional constraints and $n - 1$ additional variables, it is possible to ensure that S is free of subtours. These constraints are given by

$$(n - 1)x_{ij} + u_i - u_j \leq (n - 2), \quad i, j = 2, 3, \dots, n \quad (1)$$

where u_i , $i = 2, 3, \dots, n$ are unrestricted real variables. If S contains any subtour (a cycle with less than n arcs) then S will contain a subtour \tilde{T} that does not contain node 1. If \tilde{T} is a single node, say $\{k\}$, then inequality (1) is violated for $i = j = k$. If \tilde{T} contains more than one node, then adding the above inequalities for arcs in \tilde{T} leads to a contradiction. This establishes that the MTZ constraints force S to be free of subtours.

4.2.4 Network flow constraints. Network flow based formulations can also be used to obtain compact representations of subtour elimination constraints. We first consider a general form of subtour elimination constraints and observe that most of the network flow based subtour elimination constraints studied in literature fall within this general framework. Consider the inequality system:

$$AY \leq b \quad (2)$$

$$Y \geq 0 \quad (3)$$

$$BX + DY \leq g, \quad (4)$$

where $X = (x_{11}, x_{12}, \dots, x_{1n}, x_{21}, \dots, x_{nn})$ is a solution to the assignment problem, $Y = (y_1, y_2, \dots, y_p) \in \mathbb{R}^p$, A, B, D are matrices, and b

and g are column vectors. Each y_k is associated with an arc (i, j) of G . It is possible that y_k and y_r are associated with the same arc (i, j) for $k \neq r$, but no y_k is associated with two arcs. The inequalities within the system (2) and (3) are called *flow constraints* and the inequalities within the system (4) are called *coupling constraints*. For any feasible solution Y^* of the flow constraints, consider the subgraph G_{Y^*} induced by the arcs of G associated with positive components of Y^* . Then G_{Y^*} is called the *flow graph* of Y^* .

Theorem 2 *The assignment problem together with the flow and coupling constraints gives a valid mixed 0-1 programming formulation of the TSP if the following conditions are satisfied.*

- (i) *If the assignment solution $X^0 = (x_{ij}^0)$ represents a Hamiltonian cycle, then there exists $Y^0 \geq 0$ such that (X^0, Y^0) satisfies (2) and (4).*
- (ii) *For any feasible solution (X^*, Y^*) , the flow graph G_{Y^*} is connected.*
- (iii) *For any feasible solution (X^*, Y^*) , if $x_{ij}^* = 0$ then all y_k^* 's associated with arc (i, j) are zeros.*

Proof. The assignment constraints insure that $S = \{(i, j) : x_{ij} = 1\}$ defines either a tour or a collection of subtours. In view of (iii), if (ii) is satisfied, then S is free of subtours. The result follows from (i). ■

Let us now consider a single commodity flow formulation suggested by Gavish and Graves [350] which satisfies the conditions of Theorem 2. Let y_{ij} be the 'flow' of commodity y along arc (i, j) in G . Consider the following constraints:

Flow constraints:

$$\sum_{j=1}^n y_{ij} - \sum_{j=1}^n y_{ji} = 1, \quad i = 2, 3, \dots, n \quad (5)$$

$$\sum_{j=1}^n y_{1j} = n - 1 \quad (6)$$

$$y_{ij} \geq 0 \quad (7)$$

Coupling constraints:

$$(n - 1)x_{ij} - y_{ij} \geq 0 \text{ for } i, j = 1, 2, \dots, n \quad (8)$$

It is easy to verify that these flow constraints and coupling constraints satisfy the conditions of Theorem 2 and hence are valid subtour elimination constraints. The coupling constraints (8) could be strengthened

as

$$(n - 1)x_{1j} - y_{1j} \geq 0 \text{ for } j = 2, \dots, n \quad (9)$$

$$(n - 2)x_{ij} - y_{ij} \geq 0 \text{ for } i, j = 2, 3, \dots, n \quad (10)$$

Another slightly strengthened variation of the coupling constraints can be described as follows:

$$y_{j1} = 0, \text{ for } j = 2, 3, \dots, n \quad (11)$$

$$y_{1j} = (n - 1)x_{1j} \text{ for } j = 2, 3, \dots, n \quad (12)$$

$$x_{ij} \leq y_{ij} \leq (n - 2)x_{ij} \text{ for } i, j = 2, 3, \dots, n \quad (13)$$

This variation is equivalent to the single commodity flow formulation proposed by Gouveia and Voß [392].

A two-commodity flow formulation of the subtour elimination constraints was proposed by Finke, Claus, and Gunn [291]. Let y and z be two commodities and y_{ij} be the flow of commodity y along arc (i, j) and z_{ij} be the flow of commodity z along arc (i, j) . The resulting flow constraints are given by:

$$\sum_{j=1}^n y_{1j} - \sum_{j=1}^n y_{j1} = n - 1 \quad (14)$$

$$\sum_{j=1}^n y_{ij} - \sum_{j=1}^n y_{ji} = -1, \quad i = 2, 3, \dots, n \quad (15)$$

$$\sum_{j=1}^n z_{1j} - \sum_{j=1}^n z_{j1} = -(n - 1) \quad (16)$$

$$\sum_{j=1}^n z_{ij} - \sum_{j=1}^n z_{ji} = 1, \quad i = 2, 3, \dots, n \quad (17)$$

$$\sum_{j=1}^n y_{ij} + \sum_{j=1}^n z_{ij} = n - 1, \quad i = 1, 2, \dots, n \quad (18)$$

$$y_{ij} \geq 0 \text{ for } i, j = 1, 2, \dots, n \quad (19)$$

$$z_{ij} \geq 0 \text{ for } i, j = 1, 2, \dots, n \quad (20)$$

with coupling constraints:

$$y_{ij} + z_{ij} = (n - 1)x_{ij} \text{ for all } (ij). \quad (21)$$

It is not difficult to verify that these constraints satisfy the conditions of Theorem 2 and hence are valid subtour elimination constraints.

Using multicommodity flows, Wong [827] presented another compact representation of the subtour elimination constraints. He used $2(n - 1)$ commodities y^k and z^k , $k = 2, \dots, n$. Let y_{ij}^k be the flow of commodity y^k along arc (i, j) and z_{ij}^k be the flow of commodity z^k along arc (i, j) , $i, j = 1, 2, \dots, n$ and $k = 2, 3, \dots, n$. The flow constraints of Wong can be stated as follows.

$$\sum_{j=1}^n (y_{ij}^k - y_{ji}^k) = \begin{cases} 1 & \text{if } i = 1, k = 2, 3, \dots, n, \\ -1 & \text{if } i = k, k = 2, 3, \dots, n, \\ 0 & \text{if } i \neq 1 \text{ and } k, k = 2, 3, \dots, n \end{cases} \quad (22)$$

$$\sum_{j=1}^n (z_{ij}^k - z_{ji}^k) = \begin{cases} 1 & \text{if } i = 1, k = 2, 3, \dots, n, \\ -1 & \text{if } i = k, k = 2, 3, \dots, n, \\ 0 & \text{if } i \neq 1 \text{ and } k, k = 2, 3, \dots, n \end{cases} \quad (23)$$

with coupling constraints:

$$0 \leq y_{ij}^k \leq x_{ij} \quad \text{for all } i, j, k \quad (24)$$

$$0 \leq z_{ij}^k \leq x_{ij} \quad \text{for all } i, j, k. \quad (25)$$

Note that the source node for commodity y^k is 1 for all k and for commodity z^k , it is node k . Similarly, the sink node of commodity y^k is node k and for commodity z^k it is node 1 for all k . Constraints (22) and (23) insures that one unit of commodity y^k travels from node 1 to node k while one unit of commodity z^k travels from node k to node 1. Thus the corresponding flow graph is strongly connected, establishing condition (i) of Theorem 2. Conditions (ii) and (iii) of Theorem 2 are easy to verify. Thus constraints (22) to (25) represent valid subtour elimination constraints.

Eliminating the z_{ij} variables from the above formulation, Claus [202] presented a multicommodity flow formulation of the subtour elimination constraints as described below:

Flow constraints:

$$\sum_{i=1}^n y_{1i}^k = 1 \text{ for } k = 2, 3, \dots, n \quad (26)$$

$$\sum_{i=1}^n y_{i1}^k = 0 \text{ for } k = 2, 3, \dots, n \quad (27)$$

$$\sum_{i=1}^n y_{ik}^k = 1 \text{ for } k = 2, 3, \dots, n \quad (28)$$

$$\sum_{i=1}^n y_{ki}^k = 0 \text{ for } k = 2, 3, \dots, n \quad (29)$$

$$\sum_{i=1}^n y_{ij}^k - \sum_{i=1}^n y_{ji}^k = 0 \text{ for } j, k = 2, 3, \dots, n; j \neq k \quad (30)$$

$$y_{ij}^k \geq 0 \quad (31)$$

Coupling constraints:

$$y_{ij}^k \leq x_{ij} \text{ for } i, j = 1, 2, \dots, n; k = 2, 3, \dots, n \quad (32)$$

Other related multicommodity flow formulations of the TSP which are variations of Wong [827] are proposed by Langevin [531] and Loulou [567]. The linear programming (LP) relaxations of all these multicommodity formulations yield the same objective function value as that of the LP relaxation of Dantzig et al [239]. However, LP relaxations of the single and two commodity formulations yield weaker bounds. Other well studied formulations of TSP include time-stage dependent formulations [637]. For a comparative study of LP relaxations of various (mixed) integer programming formulations of ATSP we refer to [392, 532, 637, 643].

4.3. Symmetric TSP

Since the symmetric traveling salesman problem is a special case of ATSP (as discussed in Section 1), the above formulations are valid for STSP also. However, exploiting symmetry, we could replace the assignment constraints by 2-matching constraints. Let $\delta(v)$ be the set of edges incident on node v of G . Then a generic integer programming formulation of STSP is given by

$$\begin{aligned}
& \text{Minimize } \sum_{e \in E} c_e x_e \\
& \text{Subject to} \\
& \quad \sum_{e \in \delta(v)} x_e = 2, \quad v \in V \\
& \quad x_e = 0 \text{ or } 1, \quad e \in E \\
& \quad \{e \in E : x_e = 1\} \text{ contains no subtours.}
\end{aligned}$$

As discussed earlier, additional constraints can be used to make sure that $\{e \in E : x_e = 1\}$ is free of subtours. For other integer programming formulations of the STSP we refer to [39, 41, 169, 42, 40]

4.4. Binary quadratic programming formulation

We have seen that by adding new constraints to the assignment problem (2-matching problem), various mixed integer linear programming formulation for the ATSP (STSP) can be obtained. While the assignment constraints represent all permutations of $\{1, 2, \dots, n\}$, the additional constraints (subtour elimination constraints) make sure that only cyclic permutations are selected. We have seen in Section 1 that under the quadratic permutation representation of TSP, any permutation is feasible and hence the assignment constraints are good enough in this case. However the objective function now is no longer linear. The quadratic permutation representation of TSP yields the following binary quadratic programming representation of TSP:

$$\text{Minimize } \sum_{i,j,k=1}^n d_{ij} x_{ik} x_{j(k+1)}$$

subject to the assignment constraints, where $x_{ij} = 1$ is interpreted as ‘the j^{th} city visited is i ’ and the index $n + 1 \equiv 1$. This problem is represented sometimes using a cyclic permutation matrix. Let T be a cyclic permutation matrix of order n , i.e. its $(i, j)^{\text{th}}$ element t_{ij} is given by $t_{i,i+1} = 1$, for $i = 1, \dots, n - 1$, $t_{n1} = 1$, and all other entries zero. Now define $d_{ijkl} = c_{ij} t_{kl}$. Then TSP is equivalent to the 0-1 quadratic programming problem [545]

$$\text{Minimize } \sum_{i,j,k,l=1}^n d_{ijkl} x_{ik} x_{jl}$$

subject to the assignment constraints.

4.5. Three Matroid Intersection

Let E be a finite set and F be a family of subsets of E . $M = (E, F)$ is a *matroid* if (1) $Y \in F$, $X \subset Y$ implies $X \in F$. (2) if $|X| < |Y|$ and $X, Y \in F$ implies there exists an $e \in Y - X$ such that $X \cup \{e\} \in F$.

Let S_1, S_2, \dots, S_p be a partition of E and F be the collection of all subsets of E with at most one element from each S_i , $i = 1, 2, \dots, p$. Then $M = (E, F)$ is called a *partition matroid*. If E is the arc set of a directed graph G and S_i contains all arcs coming into node i of G generates a partition matroid. Similarly if S_i is the collection of all arcs going out of node i in G we have another partition matroid. If F contains the collection of all 1-forests (a forest with one additional arc) in G , then $M = (E, F)$ is a matroid called the *1-graphic* matroid. Any collection of n edges of G that is common to the two partition matroids defined above and the 1-graphic matroid defines a Hamiltonian cycle in G . Thus the ATSP can be formulated as a minimum cost three matroid intersection problem. Since the STSP is a special case of the ATSP, it is also a three matroid intersection problem.

5. Matrix Transformations

Let us now discuss some useful transformations of distance matrices associated with a TSP that leaves an optimal solution invariant. Let $C = (c_{ij})$ and $D = (d_{ij})$ be two $n \times n$ matrices associated with a TSP on G . For any Hamiltonian cycle H of G , for $\alpha > 0$ and $\beta \in \mathbb{R}$, if

$$\sum_{ij \in H} d_{ij} = \alpha \left(\sum_{ij \in H} c_{ij} \right) + \beta$$

then we say that C and D are *equivalent* with respect to TSP. Let $\phi : \mathbb{R} \rightarrow \mathbb{R}$ such that $\phi(c_{ij}) = d_{ij}$. If C and D are equivalent with respect to TSP, then ϕ is called an *equivalent matrix transformation* (EMT). Thus under an equivalent matrix transformation, the optimal solution of the TSP remains unchanged. For any $\gamma > 0$ and $\delta \in \mathbb{R}$, $d_{ij} = \gamma c_{ij} + \delta$ is an EMT. Similarly for $a_i, b_i \in \mathbb{R}$ and $\gamma > 0$, $d_{ij} = \gamma c_{ij} + a_i + b_j$ is an EMT which is perhaps the most popular EMT studied for TSP. However, under this EMT, the matrix D need not be symmetric even if C is symmetric. By choosing $a_i = b_i$ the EMT leaves symmetry of a matrix invariant. Another important structural property of a matrix useful in the study of TSP is the *parameterized triangle inequality* (τ -triangle inequality) [22, 117]. We say that the matrix C satisfies the τ -triangle inequality [22, 117] if

$$c_{ij} \leq \tau(c_{ik} + c_{kj}), \text{ for all } i \neq j \neq k$$

If $\tau = 1$, then we say that the matrix C satisfies the *triangle inequality*. If $\tau = 1/2$ all elements of C are equal. Thus for $1/2 \leq \tau < 1$, τ -triangle inequality is a restriction on the triangle inequality while for $\tau > 1$ it is a relaxation on the triangle inequality. Under the EMT $d_{ij} = c_{ij} + a_i + b_j$ the matrix D need not satisfy τ -triangle inequality even if C satisfies the τ -triangle inequality for moderate values of τ . However, for any $1/2 < \tau$ it is possible to choose values of a_i and b_i such that D satisfies the τ -triangle inequality even if C does not satisfy the τ -triangle inequality. To achieve this choose a_i 's and b_i 's such that

$$(\tau - 1)(a_i + b_j) + \tau(a_k + b_k) \geq c_{ij} - \tau(c_{ik} + c_{kj}) \quad \forall i \neq j \neq k \quad (33)$$

For $\tau > 1/2$ the above inequality is always feasible since $a_i = b_i = M$ is a solution for large enough M . If one prefers smaller values of a_i 's and b_i 's, a minimization linear program with objective function $\sum(a_i + b_i)$ subject to (33) as constraints, can be solved. Thus using an EMT, any cost matrix can be reduced to an equivalent cost matrix satisfying τ -triangle inequality for any $\tau > 1/2$.

A cost matrix C for a TSP is said to be in *standard reduced form* if $c_{nj} = c_{jn} = 0$, $j = 1, 2, \dots, n - 1$ and $c_{12} = 0$

Theorem 3 [489] *For any cost matrix C , there exists an EMT which produces a standard reduced matrix.*

Proof. For $i = 1, 2, \dots, n - 1$ define $a_i = c_{in} + x/2$ and $b_i = c_{ni} + x/2$ where $x = c_{12} - c_{1n} - c_{n2}$. Let $a_n = b_n = -\frac{x}{2}$. Now the matrix D obtained by the EMT $d_{ij} = c_{ij} - a_i - b_j$ will be a standard reduced matrix. If C is symmetric, it can be verified that D is also symmetric. ■

Using cost matrix of a TSP in standard reduced form allows simplification of some proofs [489]. (See also Chapter 11). As a consequence of the above theorem, we could reduce the cost of all arcs (incoming and outgoing) incident on a given node i , to zero as well as the cost of another arc not incident on i leaving the optimal solution invariant.

6. More Variations of the TSP

We now discuss additional variations of TSP studied in the literature. Our discussion on these variations are confined to their definitions only. For more details on these problems, see the corresponding references cited. Further, the references we cite need not be the paper in which the problem was originally introduced and the reference list is not exhaustive. Through out this section G is a complete graph (digraph) with c_e as the cost of edge e in G .

The time dependent TSP: For each arc (i, j) of G , n different costs c_{ij}^t , $t = 1, 2, \dots, n$ are given. The cost c_{ij}^t corresponds to the 'cost' of going from city i to city j in time period t . The objective is to find a tour $(\pi(1), \pi(2), \dots, \pi(n), \pi(1))$, where $\pi(1) = 1$ corresponds to the home location which is in time period zero, in G such that $\sum_{i=1}^n c_{\pi(i)\pi(i+1)}^i$ is minimized. The index $n + 1$ is equivalent to 1. For all (i, j) , if $c_{ij}^1 = c_{ij}^2 = \dots = c_{ij}^n$, then the time dependent TSP reduces to the traveling salesman problem. For details on this problem, we refer to [319, 392, 669].

Period TSP: This generalization of TSP considers a k -day planning period, for given k . We want to find k cycles in G such that each node of G is visited a prescribed number of times during the k -day period and the total travel cost for the entire k -day period is minimized [177].

The delivery man problem: This problem is also known as the *minimum latency problem* and the *traveling repairman problem*. Let H be a tour in G and v_1 be a starting node. For each vertex v_i of G , define the *latency* of v_i with respect to H , denoted by $L_i(H)$, is the total distance in H from v_1 to v_i . The delivery man problem is to find a tour H^* in G such that $\sum_{i=1}^n L_i(H^*)$ is as small as possible. The DMP is strongly NP-complete. For details on this problem, we refer to [1, 385, 599, 293, 572]. It can be verified that this problem is a special case of the time dependent TSP.

Black and White TSP: This problem is a generalization of the TSP. Here, the node set of G is partitioned into two sets, B and W . The elements of B are called *black nodes* and the elements of W are called *white nodes*. A tour in G is said to be feasible if the following two conditions are satisfied. (i) The number of white nodes between any two consecutive black nodes should not exceed a positive integer I and the distance between any two consecutive black nodes should not exceed a positive real number R . The black and white TSP is to find a minimum cost feasible tour in G . Applications of this problem include design of ring networks in the telecommunication industry [133]. A variation of the black and white TSP with applications in the air-line industry, known as TSP with replenishment arcs, has been discussed in [577].

Angle TSP: Let v_1, v_2, \dots, v_n be n points of the Euclidean plane. Consider a two edge path from v_i to v_k passing through v_j . The 'cost' of passing through v_j is defined as the angle between the vectors $\overrightarrow{v_i v_j}$ and $\overrightarrow{v_j v_k}$. For a tour H through v_1, v_2, \dots, v_n , let $v_i(H)$ be the cost of pass-

ing through v_i . Then the angle TSP is to find a tour H that minimizes $\sum_{i=1}^n v_i(H)$. The angle TSP is NP-hard.

Film-copy Deliverer Problem: For each node i of G , a non-negative integer p_i is prescribed. We want to find a routing of a film-copy deliverer, starting at node 1, visit each node exactly p_i times and come back to node 1 in such a way that the total distance travelled is minimized [182]. This problem generalizes TSP and some of its variations. A generalization of the Film-Copy Deliverer problem has been studied in [549]. In this case, a minimum cost cycle in G is sought that passes through each node i of G at least p_i times and at most q_i times.

The selective TSP: [355] This problem is also known as the orienteering problem (see Chapter 13). For each node i of G , a weight w_i is given. Then the selective traveling salesman problem is to find a cycle Y in G such that the sum of the weights of nodes in Y is maximized while keeping $\sum_{ij \in Y} c_{ij} \leq L$, where L is a given number. This problem is closely related to the prize collecting traveling salesman problem studied in Chapter 14.

Resource constrained TSP: For each edge e of G a requirement r_{ij} is also given in addition to its cost. Then the resource constrained traveling salesman problem is to find a minimum cost tour H in G such that $\sum_{ij \in H} c_{ij} \leq K$ where K is a given constant [663].

Serdyukov TSP: Let X_1, X_2, \dots, X_n be a collection of subsets of the node set of G such that $|X_i| \leq k$ for a given constant k . Starting from a city in X_1 , we want to find a minimum cost tour in G such that the i^{th} city visited must be from X_i [752]. For this problem we assume that G need not be complete. If d is the maximum vertex degree in G , then we call the resulting problem a (k, d) -problem. Serdyukov [752] showed that the (k, d) -problem is NP-hard for all $k \geq 2, d \geq 3$.

Ordered Cluster TSP: [25] This problem is a simplified version of the clustered TSP and is defined as follows. The node set of G is partitioned into k clusters X_1, X_2, \dots, X_k where X_1 is singleton representing the home location. The salesman starting and ending at the home location must visit all nodes in cluster X_2 , followed by all nodes in cluster X_3 , and so on such that the sum of the edge costs in the resulting tour is minimized.

Precedence Constrained TSP: Let B be a subset of $V \times V$ defined by $B = \{(i, j) : i \in V, j \in V, i, j \neq 1, \text{ and } i \text{ must be visited before } j\}$ represents the precedence constraints. Then the precedence constrained TSP is find a least cost TSP tour starting at node 1, visits each node of G exactly once and returns back to node 1 in such a way that the precedence constraints B are satisfied [71]. A special case of this problem is the dial-a-ride problem [674].

k -Peripatetic Salesman Problem: [240] A k -peripatetic salesman tour in G is the collection of k edge disjoint Hamiltonian cycles in G . The objective is to find a least cost k -peripatetic salesman tour in G . This problem has applications in network design.

Covering Salesman Problem: [233] In this case, we want to find a minimum cost cycle L in G such that the distance between L and any node i of G not on L , denoted by $d(L, i)$, is within a prescribed limit, where $d(L, i) = \min\{c_{ij} : j \text{ is a node of } L\}$.

TSP with time windows: [44, 45] For each arc (i, j) of G , let c_{ij} be the cost and t_{ij} be the traversal time. For each node i of G a triplet $\{a_i, b_i, s_i\}$ is given where $a_i \leq b_i$ and $0 \leq s_i \leq b_i - a_i$. The quantity s_i represents the service time at node i and the job at node i must be completed in the time interval $[a_i, b_i]$. If the salesman arrives at node i before a_i , he will have to wait until a_i . Then TSP with time windows is to find a least cost tour in G satisfying the restriction discussed above.

Moving Target TSP: A set $X = \{x_1, x_2, \dots, x_n\}$ of n objects placed at points $\{p_1, p_2, \dots, p_n\}$. Each object x_i is moving from $p_i \in \mathbb{R}^n$ at velocity v_i . A pursuer starting at origin moving with speed v wants to intercept all points x_1, x_2, \dots, x_n in the fastest possible time [447]. This problem is related to the time dependent TSP.

Remote TSP: For a given integer k , we are interested in finding a subset S of V with $|S| = k$ such that the minimum cost of a tour in the subgraph of G induced by S is maximized [430].

There are several other variations that one could discuss depending on various application scenarios. These variations include traveling salesman location problem, traveling salesman problem with backhauls [353], k -best TSP [800], minmax TSP [767], multi-criteria TSP [271], stochastic TSP (different variations), k -MST [113, 601], minimum cost biconnected spanning subgraph problem [604], graph searching problem [519],

k -delivery TSP [175], quadratic assignment problem [545] etc., and combinations of one or more of the variations discussed here. Vehicle routing problems form another very important class of generalizations of TSP which are not discussed here.

Acknowledgement: I am thankful to G.Gutin and K.P.K. Nair for their comments on an earlier version of this chapter. Special thanks are due to S.N. Kabadi and H. Marchand for several discussions. This work was partially supported by the NSERC grant OPG0170381.

Chapter 2

POLYHEDRAL THEORY AND BRANCH-AND-CUT ALGORITHMS FOR THE SYMMETRIC TSP

Denis Naddef

Laboratoire Informatique et Distribution, Institut National Polytechnique de Grenoble, France

Denis.Naddef@imag.fr

1. Introduction

In this chapter we deal with the problem of solving symmetric TSP (STSP) instances to optimality. Of course, STSP instances are particular cases of asymmetric TSP (ATSP) instances, those for which the distance between any two cities is irrelevant of the direction. Therefore we could transform any instance of the STSP to an asymmetric one and use the results of Chapter 4 to solve it. In fact the techniques of Chapter 4 do not perform well when the costs of the arcs (i, j) and (j, i) only slightly differ. Progress in the solution techniques for the STSP is such that it is common to transform an ATSP into a symmetric one to solve it to optimality (see [474]).

Various methods have been proposed, among which we can cite, for historical reasons, a Lagrangian relaxation algorithm based on *1-trees* by Held and Karp (see [444] and [445]) and a dynamic programming approach that can be found in any text book dealing with dynamic programming. Both methods are very limited, even today, in the size of problems they can solve to optimality. Belloni and Lucena have revisited the Lagrangian approach to the TSP in [98] and report promising results at least for instances that are not too large. The only method that has given good, not to say impressive, results is the Branch-and-Cut method using the double index formulation of the problem. This method finds its foundations in the seminal work of Dantzig, Fulkerson and Johnson all the way back in 1954 [239].

The next section contains the double index integer linear programming model we will use. We will also mention other possible formulations of the problem as integer linear programs. We will explain why a substantial amount of theoretical work has to be carried out in order to solve large instances of the problem. Those theoretical developments will be the subject of sections 3 to 12. The following three sections will be devoted to the use of these developments to solve TSP instances to optimality. From now on, in this chapter, *solve* will be understood as *solve to optimality*, which includes a proof of optimality, even if this proof is not easily checkable. Numerical results will be given to show that the method indeed works very well on a large set of instances.

We assume the reader knows the basic concepts of linear algebra dealing with polyhedra.

Except when specified, we assume that we deal with a complete graph $K_n = (V, E)$, with $n = |V|$ representing the number of vertices. Let $S \subset V$, then $\delta(S)$ (resp. $\gamma(S)$) represents the set of edges with exactly one endnode in S (resp. both endnodes in S), i.e. $\delta(S) = \{(u, v) \in E : u \in S, v \notin S\}$ (resp. $\gamma(S) = \{(u, v) \in E : u \in S, v \in S\}$). The edge set $\delta(S)$ is in general called the *coboundary* of S (some authors say *cocycle* of S or *cut* defined by S). We write $\delta(v)$ instead of $\delta(\{v\})$ for $v \in V$. For $S \subset V$ and $T \subset V \setminus S$ we denote by $(S : T) = (T : S)$ the set of edges with one endnode in S and the other in T . We denote by \mathbb{R}^E the set of vectors indexed by E , that is the components of a vector of \mathbb{R}^E are in one to one correspondence with the elements of E . For $E^* \subset E$ and $x \in \mathbb{R}^E$ we let $x(E^*)$ represent $\sum_{e \in E^*} x_e$. Let $G(S)$ denote the induced subgraph on S , i.e. $G(S) = (S, \gamma(S))$. Let $x^* \in \mathbb{R}^E$, with x_e^* viewed as the *capacity* of edge e ; for $S \subset V$ we call $x^*(\delta(S))$, the *value* of the cut defined by S . In the following we will abusively say cut S for cut defined by S , that is for $\delta(S)$.

The STSP instances we will mention are taken from the TSPLIB [709]. The number in the name of the instance represents the number of cities.

2. Integer linear programming models

There are various ways of modelling the STSP as an integer linear program. Some contain a polynomial (in n) number of variables and constraints, they are said to be *compact*, some others only a polynomial number of variables but an exponential number of constraints.

Since a STSP instance can be transformed into an asymmetric one by replacing each edge by two oppositely directed arcs of same cost, all the integer formulations for the ATSP yield formulations for the STSP. There are various known compact formulations for the ATSP, see for

example [650] and [291]. We will mention two compact formulations for the STSP that do not come from the ATSP.

For computational reasons, which will become clear in this chapter, among two integer models, the best is the one whose linear relaxation has the highest objective function value. In order for this to make sense one must assume that the integer linear programming model is minimal in terms of inequalities, since any integer linear programming model can be strengthened by the use of valid inequalities for the underlying polyhedron, which will be the subject of most of this chapter. This strengthening, at least theoretically, could achieve a value equal to the optimal integral value.

Among all the integer linear programming models known, in this respect, three models emerge and attain the same value for their linear relaxations. None is known that attains a higher value. These are the *multistage insertion* formulation of Arthanari [39] (see also [41]), the *cycle shrink* of Carr [169], and finally what is known as the *double index* or *subtour elimination* formulation. This latter formulation will be extensively described here since it is the only one that has been used so far in computational studies. The multistage-insertion is inspired by dynamic programming recursion, that is building up a tour step by step. Cycle shrink does the opposite, that is going from a tour to a node. It is not surprising then that these two formulations are equivalent, see Arthanari and Usha [42] (see also [40] for the equivalence with the double index formulation). For the ATSP, Padberg and Sung [650] show that the double index formulation dominates all the other known ones.

We now describe the *double index* formulation. To every edge $e \in E$ we associate a variable x_e which will take value 1 if e is in the resulting optimal tour and 0 else. The STSP can then be formulated as the following integer linear program:

$$(\text{IP}(\text{STSP})) \quad \min \sum_{e \in E} c_e x_e \quad (1)$$

subject to

$$x(\delta(v)) = 2 \quad \text{for } v \in V \quad (2)$$

$$x(\delta(S)) \geq 2 \quad \text{for } 3 \leq |S| \leq |V|/2 \quad (3)$$

$$0 \leq x_e \leq 1 \quad \text{for } e \in E \quad (4)$$

$$x_e \text{ integer} \quad \text{for } e \in E \quad (5)$$

Equations (2) just say that two edges have to be chosen incident to any vertex. Inequalities (3) forbid cycles which do not contain all the vertices and are called *subtour elimination inequalities*. If $S = \{v\}$, i.e. $|S| = 1$, the corresponding Inequality (3) is implied by the Equation (2)

corresponding to v . Since $\delta(S) = \delta(V \setminus S)$, we can restrict the subtour elimination inequalities to sets S with at most half the vertices. The number of subtour elimination inequalities is still in $\Omega(2^n)$, that is, exponential in the size of the problem. This may seem to be a serious problem, we will discuss this point later on.

The name *double index* formulation comes from the ATSP in which both extremities of the arcs play different roles and therefore x_{ij} is used instead of x_e , creating two indices. When dealing with the STSP this is not the case, so we stick to variables x_e and will avoid using the misleading terminology “double index” and rather use the other usual name of *subtour elimination* formulation which comes from the Inequalities (3).

Why deal with an exponentially large integer model while the other two equivalent, in terms of strength, formulations only have a polynomial number of them? This has to do with the fact that separating these inequalities is not a problem. We will make this concept of separation more precise shortly.

Why do we prefer formulations with high optimal linear relaxation value? This has to do with the solution technique which goes as follows.

The linear relaxation $LP(STSP)$ of $IP(STSP)$ is the linear program obtained by dropping the integrality conditions (5). The most obvious way to solve $IP(STSP)$ is via Branch-and-Bound using as lower bound the value of the linear relaxation of the subproblems (see for example [209], [826]). The method goes as follows (assuming we are minimizing, which is our case) and that the term feasible relates to the *integer* linear program:

A Branch-and-Bound algorithm for integer linear programming

- **Initialization:** The set of subproblems to solve is initialized to the linear program obtained by dropping the integrality conditions on the variables. This problem is referred to as the root subproblem.
- **Choose a subproblem:** If the list of open problems is empty, the best known feasible solution is optimal. Else choose an open subproblem and delete it from the list.
- **Treat subproblem:** Solve the associated linear program. If the solution is integer, go back to *Choose a subproblem* after eventually updating the best known integer solution and the best known solution value.
- If the value of the objective function exceeds that of the best known feasible solution, go back to *Choose a subproblem*.

- Else, using some linear inequality, partition the current subproblem into two new subproblems. The union of the feasible (integer) solutions to each of these two subproblems contains all the feasible solutions of the problem that has been partitioned. This is commonly done by choosing a variable with a current fractional value \bar{x}_e , and by creating two subproblems which are added to the list of open problems, one in which we impose $x_e \geq 1$ and another in which we impose $x_e \leq 0$. (There are other ways of doing this, which will be exposed in due time.) These two subproblems are called the *sons* of the current subproblem. Go to *Choose a subproblem*.

The reader may realize that, as described here, the algorithm is not of any use for our problem since none of the linear programs could fit on any computer for even relatively small size instances. This is due to the number of constraints which is exponential in the number of cities of the TSP instance. One way of getting around this is known as *constraint* or *row generation*. One call to the solution of a linear program, in the previous algorithm, will now be replaced by a series of calls. In our case it goes as follows.

Assume we have an algorithm that for a given vector $\bar{x} : E \rightarrow \mathbb{R}$, seen as capacities on the edges of G , returns a *minimum capacity cut* of G . Such an algorithm returns a set $S \subset V$ such that $\bar{x}(\delta(S))$ is minimum. The Branch-and-Bound algorithm described above is modified in the initialization and the treatment of each subproblem as follows:

- **Initialization:** The set of subproblems to solve is initialized to the linear program consisting of Constraints (2) and (4).
- **Treat subproblem:** Until the subproblem is not declared examined, repeat: Solve the current linear program which yields the optimal solution \bar{x} . Search for $S \subset V$ such that $\bar{x}(\delta(S)) < 2$. If no such S exists, the subproblem is **examined**, else, add $x(\delta(S)) \geq 2$ to the set of constraints of the current linear program.

When one has finished treating the first subproblem, which we refer to as the *rootnode problem*, one has *optimized over the STSP subtour elimination polytope*.

We are now concerned with two questions:

- Is there an efficient algorithm to find a minimum capacity cut in a weighted undirected graph?
- Does the number of calls to the minimum capacity cut algorithm remain reasonable in practice?

To answer the first question, Nagamochi and Ibaraki in [622] and Nagamochi, Ono and Ibaraki in [623] give efficient minimum cut algorithms in the case of undirected graphs. Padberg and Rinaldi in [646] give very efficient preprocessing procedures that significantly accelerate any minimum cut algorithm. One can find a study of performances of several minimum cut algorithms in Jünger, Rinaldi and Thienel [476] and in Chandra, Chekuri, Goldberg, Karger, Levine and Stein [181].

As for the second question, since finding a minimum capacity cut is polynomial, using the ellipsoid algorithm, one can, theoretically, finish in a polynomial number of iterations (see Grötschel, Lovász and Schrijver [399]). If we use the procedure we just described, almost all problems of the TSPLIB require very few iterations.

The value of the linear relaxation obtained from $IP(STSP)$ by dropping the integrality conditions on the variables (i.e. the root problem), is known as the *Held and Karp bound* or *value on the subtour elimination polytope*.

Although the Held and Karp bound is very good, it is not good enough to be able to solve even average-size STSP instances by the procedure just described since the number of subproblems created is far too high. We mention here a conjecture, which seems difficult to trace back to its origin, but that can be found in Goemans [383].

Conjecture 1 *If the edge costs satisfy the triangular inequality, then the optimal value of a tour is at most $4/3$ the value of the Held and Karp bound.*

Consider the partition of the vertex set V into V_i^j , $i = 0, \dots, k$, $j = 1, \dots, 3$, with $V_0^j = A$ and $V_k^j = Z$ for $j = 1, \dots, 3$, and $k \geq 3$. In Section 6 this will be called a $(k-1)$ -path configuration. Let $c_e = 0$ if $e \in \gamma(V_i^j)$ for all i and j , $c_{uv} = 1$ if $u \in V_i^j$ and $v \in V_{i+1}^j$ for all i and j , $c_{uv} = k-2$ if $u \in A$ and $v \in Z$. For all the other edges $e = (u, v)$, let c_{uv} equal the length of a shortest path linking u to v using only edges for which the cost has been defined in the preceding lines. For such an instance of STSP, the optimal value of a tour is $4k-2$ and the Held and Karp bound is $3k$, therefore the ratio can be made as close as one desires from the bound of the conjecture.

For all the examples of the TSPLIB, the gap is much less than the one we would expect from the conjecture.

The more general procedure known as Branch-and-Cut differs from the algorithm described to optimize on the subtour elimination polytope in the way a subproblem is treated. One not only tries to generate constraints coming from the integer linear formulation but also any linear inequality that separates the current fractional solution from the feasible

solutions. Such an inequality is called a *cutting plane* or just a *cut*. Using the term *cut* would be confusing since it is also a set of edges that disconnects an undirected graph. In the following, except for the name Branch-and-Cut, we will avoid saying cut for an inequality. Remember that we will use cut S in place of cut $\delta(S)$.

The next sections will show the existence of a finite set of linear inequalities, which can be added to $IP(STSP)$ in such a way that the linear relaxation of the newly obtained integer program is always integral and therefore corresponds to an optimal tour. Unfortunately, as we will see, this is only an existence theorem and it is unlikely that we will discover that complete set of linear inequalities in a near future even for medium size instances. Table 3.1, which will be commented in the next section, gives the number of such inequalities for instances up to 10 nodes. Fortunately, even a partial knowledge of that description is of precious help in solving STSP instances, and that will be the topic of the next sections.

In view of all this the **Treat subproblem** phase of a Branch-and-Cut looks like this:

Treat subproblem. Repeat until a stopping criterion is attained:

- Solve the current lp, let \bar{x} be its fractional optimal solution \bar{x} .
- Find a linear inequality $fx \geq f_0$ satisfied by all incidence vectors (see definition in next section) of tours and such that $f\bar{x} < f_0$.

Two stopping criteria are given in Section 19.4.

3. Introducing the symmetric TSP polytope and its various relaxations

We let \mathcal{H}_n (resp. \mathcal{H}_G) represent the set of tours of the complete graph K_n (resp. of graph G). To every tour $\Gamma \in \mathcal{H}_n$ (resp. $\in \mathcal{H}_G$), we associate a vector $x^\Gamma \in \mathbb{R}^E$ such that $x_e^\Gamma = 1$ if $e \in \Gamma$ and $x_e^\Gamma = 0$, else. This vector is called indifferently *incidence* or *representative* vector of Γ . Some authors use *characteristic* vector of Γ .

3.1. The symmetric TSP polytope

The *symmetric TSP polytope* of G , $STSP(G)$, is the convex hull of all the vectors x^Γ when Γ ranges over all Hamiltonian cycles of G . When $G = K_n$, we let $STSP(n)$ stand for $STSP(K_n)$, i.e. $STSP(n) = \text{conv}\{x^\Gamma : \Gamma \in \mathcal{H}_n\}$. When not specified, we assume the underlying graph is K_n .

n	# tours	# different facets	# facet classes
3	1	0	0
4	3	3	1
5	12	20	2
6	60	100	4
7	360	3 437	6
8	2 520	194 187	24
9	20 160	42 104 442	192
10	181 440	≥ 51 043 900 866	≥ 15 379

Table 2.1. Some statistics on $STSP(n)$

It is well known from a theorem of Weyl [823], that $STSP(n)$ can be described by a *finite* set of linear equations and inequalities. The study of the symmetric TSP Polytope consists in finding such linear equations and inequalities. All the linear *equations* are known in the case of $STSP(n)$, this is the basis of the proof of Theorem 2. This is not the case for $STSP(G)$ if G is an arbitrary connected graph.

Before going further on we give an idea of the richness of the facial structure of $STSP(n)$. Table 3.1 is taken from Christof and Reinelt [186]. This table gives, for small values of n , the number of tours, the number of different facets and the number of classes in which they can be put. Two facet inducing inequalities for $STSP(n)$ are said to belong to the same class if a renumbering of the vertices transforms one into the other. One can observe that this polytope is highly degenerate, that is the number of facets incident to a vertex greatly exceeds the minimum number of facets needed to define that vertex. As can be seen from this table, the complexity of the polytope increases very fast and so does its degeneracy. The numbers in the last line are conjectured to be exact.

The first thing to do, when studying a polyhedron, is to determine its dimension. That dimension is not known for $STSP(G)$ when G is a connected arbitrary graph. Note that $STSP(G)$ is non empty if and only if G is Hamiltonian.

For complete graphs, the polytope dimension is given by the following theorem.

Theorem 2 *The dimension of $STSP(n)$ is $|E| - |V|$.*

Proof: (sketch): For $n = 3$ there is only one tour, so the theorem is true in that case. Equations (2) are linearly independent, so the dimension cannot be more than what is announced in the theorem. It is therefore enough to exhibit $|E| - |V| + 1$ affinely independent tours. For this we use a theorem on the partition of the edges of K_{n-1} . If $n = 2k + 1$, then the edges of K_{n-1} can be partitioned into k edge disjoint Hamiltonian

cycles. If $n = 2k$, then the edges of K_{n-1} can be partitioned into $k - 1$ edge disjoint Hamiltonian cycles and a perfect matching. For n odd, for each of the $(n - 1)/2$ disjoint Hamiltonian cycles that partition the edges of K_{n-1} and each edge of that cycle, we create one Hamiltonian cycle of K_n by inserting node n between the endnodes of that edge, i.e. if $e = (i, j)$, we remove edge e and add the two edges (i, n) and (j, n) . In the other case, for the Hamiltonian cycles of the partition, we do the same thing. As for the perfect matching, we complete it arbitrarily to a Hamiltonian cycle, and do as previously only with the edges of the perfect matching. We leave it to the reader to check that in both case we have the right number of cycles and that the corresponding vectors are affinely independent. ■

Remark 3 *Another proof not using the decomposition of the edges of K_n is given by Queyranne and Wang in [687].*

Corollary 4 *Any Equation in the description of $STSP(n)$ is a linear combination of the Equations (2). Therefore Equations (2) are the only necessary equations in a minimal description.*

Remark 5 *In Chapter 11, the problem of characterizing the cost functions such that all Hamiltonian cycles have the same cost (constant TSP) is raised. If one knows a maximal linearly independent set of linear equations $c^i x = c_0^i$ for $i = 1, \dots, p$, satisfied by all the incidence vectors of Hamiltonian cycles, one has the dimension of $STSP(G)$ and also a solution to the constant STSP problem. Let c be a cost function on the edges such that all Hamiltonian cycles of G have the same cost c_0 , then $cx = c_0$ is a linear equations satisfied by all Hamiltonian cycles of G and is therefore a linear combination of the equations $c^i x = c_0^i$ for $i = 1, \dots, p$.*

In the case of complete graphs this yields:

Corollary 6 *(see also Chapter 11) The only cost functions on the edges that have the property that all Hamiltonian cycles of the complete graph have the same cost, are those obtained from any function $\pi : V \rightarrow \mathbb{R}$ and by setting $c_e = \pi(i) + \pi(j)$ for all $e = (i, j)$.*

Proof: Let $c : E \rightarrow \mathbb{R}$ be such that all the tours have a cost of c_0 . Then $cx = c_0$ is an equation satisfied by all tours, and therefore, by Theorem 2, must be a linear combination of the equations (2). The coefficients of that linear combination are the $\pi(i)$'s. ■

Finding the dimension for general graphs is much more complex and only very few results are known. In fact deciding whether $STSP(G) \neq$

\emptyset is NP-complete. The reason is that the set of equations is difficult to find, since some constraints written as inequalities turn out to be equations. Take for example the case of Figure 2.1. There is an implicit equation, linearly independent from the degree constraints, namely that all tours contain exactly two of the three horizontal edges, therefore the subtour inequality defined by one of the triangles is in fact an equation.

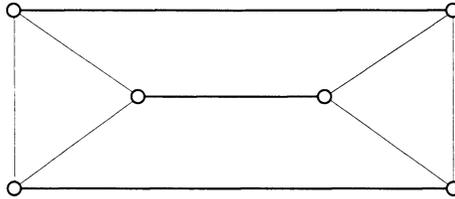


Figure 2.1. The diamond graph

For G a Halin graphs (see Cornuéjols, Naddef and Pulleyblank [220]), a complete description of $STSPP(G)$ is known. This case was generalized to classes of graphs which have the property to “decompose”, in some way, using three-edge cutsets (see Cornuéjols, Naddef and Pulleyblank [221]). It would be a nice result to characterize those graphs for which Constraints (2) to (4) of the linear relaxation of $IP(STSP)$ describe $STSPP(G)$. We will mention, later on, such a result but for the convex hull of *spanning closed walks*, a *spanning closed walk* being a cycle visiting each vertex **at least** once (any edge can be used an arbitrary number of times).

From now on we restrict to the case of complete graphs. Non full dimensional polyhedra have the undesirable property that an infinite number of linear inequalities define the same facet, unlike the case of full dimensional ones where a facet is described by a unique inequality up to scaling by a strictly positive factor. The following theorem, which in fact is a corollary of Theorem 2, deals with two representations of the same facet of $STSPP(n)$, where A is the node-edge incidence matrix of K_n .

Theorem 7 *Let $f x \geq f_0$ be a facet inducing inequality of $STSPP(n)$. The valid inequality $g x \geq g_0$ defines the same facet if and only if there exists $\pi : V \rightarrow \mathbb{R}$ and $\pi_0 \in \mathbb{R}$, $\pi_0 > 0$, such that: $g = \pi_0 f + \pi A$ and $g_0 = \pi_0 f_0 + 2 \sum_{i \in V} \pi_i$*

Testing whether two facet inducing linear inequalities define or not the same facet can be solved in $O(n^2)$ using the following algorithm of F.Margot [582]. Choose a spanning tree T and add an edge e^* that

creates an odd cycle when added to the edges of T . Solve for the variables π_i , $i \geq 1$ as a function of π_0 using the set of n equations relative to each edge of $T + \{e^*\}$. This can be performed in $O(n)$ time. Solve for π_0 using the right hand sides. In $O(n^2)$ time check whether the solution satisfies the equations relative to all the edges not in $T + \{e^*\}$.

The usual way to tackle the problem of multiple linear descriptions of the same facet, is to embed the non full dimensional polyhedron in a full dimensional one. In general we require the original polyhedron to be a face of the full dimensional one. The larger polyhedron is called a *relaxation* of the smaller one. A first step is to study a minimal linear description of the relaxation, which is unique (up to scaling by a positive integer), and then try to see which facets of the relaxation define facets of the original polyhedron.

We now turn to the various relaxations that have been used to study STSPP(n).

3.2. The monotone Relaxation

Historically this has been the first relaxation used in the study of STSPP(n) [195], [402], [401], [403], [404], [642]. The *monotone STSP polytope* is the convex hull of the incidence vectors of the tours and all edge subsets of tours of K_n . It is trivially of full dimension since the empty set is a subset of a tour, and the sets consisting of a single edge are also subsets of tours, since, in K_n , every edge belongs to some tour (in fact each edge appears in $(n-2)!$ different tours). Balas and Fischetti [74] give some very simple conditions under which facet inducing inequalities of the monotone STSP polytope yield facets of *STSPP*(n).

We will not detail this relaxation since it seems to have attained its limits and is nowadays only very seldom used. We will pass over relaxations of *STSPP*(n) using the *2-matching polytope* which has received very little attention (see Cornuéjols and Pulleyblank [224], [225], [223]) and turn to two much more interesting relaxations.

3.3. The Hamiltonian path relaxation

This relaxation is due to Queyranne and Wang [687]. They observed that there is a bijection between the tours of K_{n+1} and the Hamiltonian paths of K_n . Let *HP*(n) denote the convex hull of the representative vectors of all the Hamiltonian paths (no fixed extremities) of K_n . The following theorem states that this polytope is near full dimensional.

Theorem 8 $Dim(HP(n)) = \frac{n(n+1)}{2} - 1$

Proof: Note that $x(E_n) = n - 1$ holds for all representative vectors of Hamiltonian paths of K_n . Hence the dimension cannot be more than stated. Let $fx = f_0$ be an equation satisfied by all Hamiltonian paths of K_n . Let e_1 and e_2 be any two different edges of K_n and let Γ be a Hamiltonian cycle of K_n containing these two edges, which always exists. Let $P_i = \Gamma \setminus \{e_i\}$, for $i = 1, 2$, be two Hamiltonian paths obtained from Γ by removing one of the edges e_1 and e_2 respectively. We have $fx^{P_1} = f_0$ and $fx^{P_2} = f_0$ and therefore $fx^{P_1} - fx^{P_2} = 0 = f_{e_1} - f_{e_2}$. Therefore $f_{e_1} = f_{e_2}$, and since e_1 and e_2 are arbitrary, f_e is a constant for $e \in E_n$, that is $fx = f_0$ is a multiple of $x(E_n) = n - 1$. ■

The main interest of this approach is that it provides a normalized form for the linear inequalities defining facets of $STSP(n)$. A facet defining inequality $fx \leq f_0$ for $STSP(n)$ is in *normalized form* if:

- $f_e = 0$ for all $e \in \delta(\{1\})$
- $f_e \geq 0$ for all $e \in E \setminus \delta(\{1\})$
- $\exists e \in E \setminus \delta(\{1\})$ such that $f_e = 0$
- $\min\{f_e : e \in E \setminus \delta(\{1\}) \text{ and } f_e > 0\} = 1$

Theorem 9 *Every facet defining inequality of $STSP(n)$ has a unique normalized form*

Proof: See [687]. ■

How difficult is it to find the normalized form of the facet defined by a facet inducing inequality $fx \leq f_0$? Queyranne and Wang give a $O(n^2)$ algorithm to do so. This yields another $O(n^2)$ algorithm to check whether or not two facet defining inequalities define the same facet of $STSP(n)$. To do so, find the normalized form of the facet defined by each. If they are identical, the facets are the same, else they are different. We will give later on another standard form for $STSP(n)$ facet defining inequalities, which will only lead to a $O(n^3)$ algorithm for the problem of recognizing whether or not two inequalities define the same facet of $STSP(n)$.

3.4. The graphical relaxation

Let $G = (V, E)$ be a connected, not necessarily complete, graph. A *spanning closed walk* W of G is a family of edges of G (in a family we assume a same element can appear more than once) such that the graph (V, W^*) is *eulerian*, where W^* is a set of edges obtained from W by replicating an edge as many times as it appears in W . An *eulerian*

graph is a connected multigraph in which each vertex is incident to an even number of edges. Figure 2.2 shows a graph and one of its closed walks. We will **always** assume closed walks to be spanning and therefore omit the word spanning from now on.

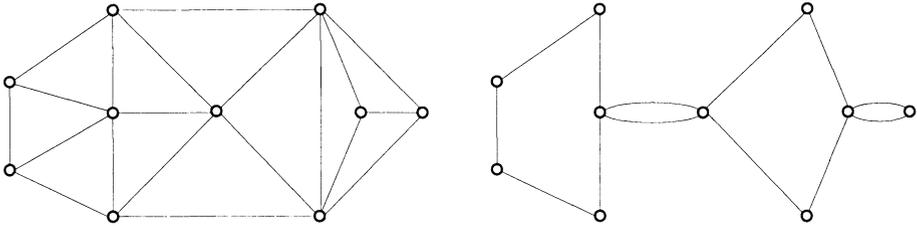


Figure 2.2. A graph and one of its closed walks

We can assign a vector $x^W \in \mathbb{R}^E$ to a closed walk W such that x_e^W represents the number of times e appears in W .

The *Graphical Traveling Salesman Polyhedron of G* , $GTSP(G)$, is the convex hull of all the representative vectors of closed walks of G . As we will see, $GTSP(n) = GTSP(K_n)$ has very strong ties with $STSP(n)$, which explains why today this is the most used relaxation to study the latter polytope. Note that $STSP(n)$ is the face of $GTSP(n)$ defined by $x(E) = n$.

If W is a closed walk of G , then so is $W + (2k)\{e\}$ (the walk obtained from W by going k times back and forth along edge e) for any positive integer k , and therefore $GTSP(G)$ is an *unbounded* polyhedron if G is connected and has at least two vertices.

The problem of finding a closed walk of minimum cost in G is called the *Graphical Traveling Salesman Problem on G* , $GTSP(G)$. The term “graphical” may not be the most suitable, but at the time it was defined in [219], the idea was to reflect the fact that this version of the traveling salesman problem has an optimal solution as long as the graph is connected. Note that if some edge e is such that $c_e < 0$, then the problem has no finite optimal solution since adding to any solution two copies of edge e strictly decreases its value. If all edge costs are non-negative, then there is an optimal solution in which no edge will appear more than twice. Finally note that if in addition the edge costs satisfy the triangular inequality and $G = K_n$, then there is an optimal solution to $GTSP(n)$ which is also a solution to $STSP(n)$.

Theorem 10 *If G is connected, then $GTSP(G)$ is of full dimension, otherwise it is empty.*

Proof: If G is not connected, then no closed walk exists. Else let W be any given closed walk. Consider the $|E|$ closed walks $W_e = W + 2\{e\}$ for all $e \in E$. The representative vectors of those $|E|$ closed walks and that of W are affinely independent. ■

The following theorem shows that if G has no *bridge*, that is an edge the removal of which disconnects the graph, then the convex hull of the extreme points of $GTSP(G)$ is a polytope of full dimension. If edge e is a bridge, then each closed walk which corresponds to an extreme point uses exactly twice that edge, and therefore $x_e = 2$ is an equation satisfied by all such walks.

Theorem 11 *If G is connected with k bridges, then the convex hull of the extreme points of $GTSP(G)$ is a polytope of dimension $|E| - k$.*

Proof: See [219]. ■

Theorem 12 *The inequality $x_e \geq 0$ defines a facet of $GTSP(G)$ if and only if the edge e is not a bridge.*

Proof: If e is a bridge, then $x_e \geq 2$ holds for every closed walk of G . If e is not a bridge, then there exists a closed walk W of $G \setminus \{e\}$. Consider the $|E| - 1$ closed walks $W_f = W + 2\{f\}$ for every $f \in E \setminus \{e\}$. The representative vectors of these closed walks together with that of W are affinely independent. ■

Theorem 13 *Let $S \subset V$, then $x(\delta(S)) \geq 2$ is a facet of $GTSP(G)$ if and only if the induced subgraphs $G(S)$ and $G(V \setminus S)$ are connected.*

Proof: If, say, $G(S)$ is not connected, then $x(\delta(S)) \geq 4$ holds for all closed walks and therefore the inequality $x(\delta(S)) \geq 2$ is not even supporting for $GTSP(G)$. Conversely, for each $e \in \delta(S)$, let W_e be a closed walk that contains twice e and no other edge of $\delta(S)$. Let W_{e^*} be one of the just defined closed walks. For all $e \notin \delta(S)$, let $W_e = W_{e^*} + 2\{e\}$. The $|E|$ representative vectors of these closed walks are affinely independent and all satisfy the inequality with equality. ■

The reader may be surprised at this point that we did not give a formulation of $GTSP(G)$ as an integer linear program. The problem is that no such formulation is known. All we can write is the following:

$$\min cx \tag{6}$$

subject to

$$x(\delta(S)) \geq 2 \text{ and even for } 1 \leq |S| \leq |V|/2 \tag{7}$$

$$x_e \geq 0 \text{ and integer for } e \in E \tag{8}$$

Research Question 14 Find an integer linear formulation for the graphical traveling salesman problem using only the variables of the double index formulation.

The previous question needs to be made more precise. It is known that there does not exist a set of linear inequalities with the property that all points with integral components inside the polyhedron defined by these inequalities correspond to closed walks. The question is to find a set of linear inequalities such that all *integral extreme points* of the polyhedron they define correspond to spanning closed walks.

As suggested by M. Fischetti, one could change the space of variables, associating an integer variable $y_u \in \mathbb{N}$ to each node u , and add the constraints $x(\delta(u)) = 2y_u$.

Fonlupt and Naddef [317] characterized those graphs G for which the following set of linear inequalities defines $GTSPP(G)$

$$x(\delta(S)) \geq 2 \quad \text{for } 1 \leq |S| \leq |V|/2 \quad (9)$$

$$x_e \geq 0 \quad \text{for } e \in E \quad (10)$$

A *minor* of a graph G is any graph H that can be obtained from G by recursively performing the following operations in some order.

- 1 (edge deletion) Remove an edge from the current graph
- 2 (edge contraction) Remove an edge from the current graph and identify its two extremities.

Theorem 15 Inequalities (9) and (10) describe $GTSPP(G)$ if and only if the graph G does not contain, as a minor, one of the three graphs shown in Figure 2.3.

Proof: The proof is very long and technical. See [317]. ■

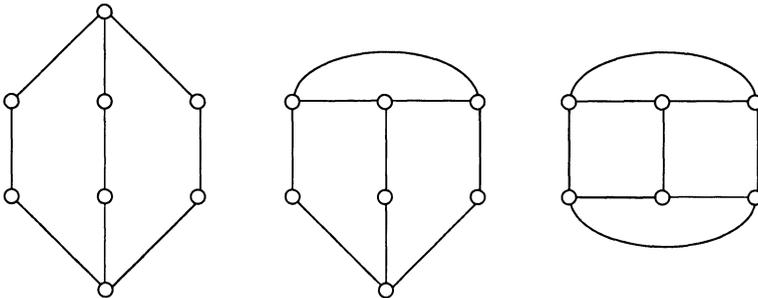


Figure 2.3. Excluded minors for Theorem 15

Graphs without such minors are well characterized, in the sense that they are easy to recognize. There is a small set of *minimal* graphs of the family called *bricks*. Given two graphs of the family, one can compose them in two ways to obtain a new graph of the family. One is by identifying a node of each, the second by identifying two nodes of each. Each non minimal graph of the family is obtained from two smaller ones of the family. Conversely, using disconnecting sets of one or two nodes, we can decompose the graph into smaller graphs recursively. At the end, if all obtained graphs belong to the list of possible starting bricks, then the graph belongs to the family. For more details see [317].

A similar characterization for the convex hull of Hamiltonian cycles is not known. Cornuéjols, Naddef and Pulleyblank, in [220] and [221], give families of graphs, which includes the Halin graphs, for which the convex hull of the Hamiltonian cycles is given by the degree constraints and the subtour elimination inequalities.

There are other known families of TSP instances that can be solved to optimality using only these inequalities in the formulation. But in these families, this is due to the objective function. Papadimitriou and Steiglitz [654] gave a family of TSP instances known as the *Papadimitriou and Steiglitz's traps* designed to defeat any attempt to solve them by a $k - OPT$ procedure, that is a procedure in which at each iteration one tries to replace a tour by a better one differing from it by at most k edges. Padberg and T.Y Sung [649] have shown that these graphs always yield an integral solution to the linear program obtained by Constraints (2), (3) and (4). The polytope on these constraints is in general called the *subtour elimination polytope*. The same has been observed by Althaus and Mehlhorn [20] for TSP problems arising in curve reconstruction. It yields a polynomial reconstruction algorithm. These two last examples are studied in Chapter 11. Moreover, in some experimental results, G. Rinaldi has observed that if the distance between cities is independently and uniformly randomly generated in some large interval, optimizing over the subtour elimination polytope almost always yields integer optimal solutions.

We now study more in depth the strong relationship between $GTSP(n)$ and $STSP(n)$.

4. The graphical relaxation Framework

Except when otherwise specified, graphs are assumed to be complete.

4.1. General theorems

We introduce first some notation. The same way that we represented the set of all tours of K_n by \mathcal{H}_n , we represent the set of all closed walks of K_n by \mathcal{W}_n^* . A *minimal closed walk* is a closed walk that does not contain a subfamily of edges which is also a closed walk. In particular, a minimal closed walk never contains an edge more than twice. The converse is of course false. The set of minimal closed walks is represented by \mathcal{W}_n .

From now on we do not differentiate sets of edges and the vectors that represent them. Therefore when we talk of a tour or a closed walk it is either a set of edges or its representative vector.

Given an inequality $fx \geq f_0$, valid for $STSP(n)$ (resp. $GTSP(n)$), we call *extremal* those tours (resp. minimal closed walks) which satisfy that inequality with equality. We let \mathcal{H}_f^- (resp. \mathcal{W}_f^-) represent the set of extremal tours (resp. minimal walks) with respect to $fx \geq f_0$. We will also often say that a tour of \mathcal{H}_f^- is *tight* for $fx \geq f_0$, the same for the closed walks.

A trivial remark is that for any valid inequality $fx \geq f_0$ of $GTSP(n)$, we have $f_e \geq 0$ for all $e \in E$. This comes from the fact that if $f_{e^*} < 0$ for some $e^* \in E$, then adding enough copies of e^* to any closed walk will bring its value below f_0 , contradicting that it is a valid inequality.

For any inequality $fx \geq f_0$ defined on \mathbb{R}^E and for each node $u \in V$, we define the set $\Delta_f(u)$ by :

$$\Delta_f(u) = \{(v, w) \in E : v \neq u, w \neq u, f_{vw} = f_{uv} + f_{uw}\}. \quad (11)$$

The set $\Delta_f(u)$ plays a central role in our study of $STSP(n)$ and $GTSP(n)$.

Another central notion in the study of $GTSP(n)$ is that of *tight triangular inequality*.

Definition 16 *An inequality $fx \geq f_0$ defined on \mathbb{R}^E is said to be tight triangular or in tight triangular form if the following conditions are satisfied:*

- (a) *The coefficients f_e satisfy the triangular inequality, i.e. $f_{uv} \leq f_{uw} + f_{wv}$ for each triplet $\langle u, v, w \rangle$ of distinct nodes of V .*
- (b) *$\Delta_f(u) \neq \emptyset$ for all $u \in V$*

We will abbreviate “tight triangular” by “TT”. The following theorem shows that almost all facet defining inequalities for $GTSP(n)$ are tight triangular.

Theorem 17 *A facet defining inequality $fx \geq f_0$ for $GTSP(n)$ falls in one of the following three categories:*

- (i) trivial inequalities $x_e \geq 0$ for all $e \in E$
- (ii) degree inequality $x(\delta(v)) \geq 2$ for all $v \in V$
- (iii) tight triangular inequalities

Proof: Let $fx \geq f_0$ be a facet defining inequality for $GTSP(n)$. Suppose it does not satisfy condition (a) of Definition 16. Then there is a triplet of vertices u, v, w such that $f_{uv} > f_{uw} + f_{wv}$. If there is a walk $W_{uv} \in \mathcal{W}_f^-$ containing the edge (u, v) , then the closed walk $W' = W_{uv} + \{(u, w)\} + \{(w, v)\} - \{(u, v)\}$ is such that $fx^{W'} < f_0$, which contradicts the fact that the inequality is valid. Therefore no closed walk of \mathcal{W}_f^- contains the edge (u, v) and the facet is that defined by $x_{uv} \geq 0$. Now assume that the inequality satisfies (a) but not (b) of Definition 16. Therefore there exists $u \in V$ such that for all pairs of distinct nodes v and w of $V \setminus \{u\}$, we have $f_{vw} < f_{uv} + f_{uw}$. If there is a closed walk $W \in \mathcal{W}_f^-$ such that the degree of u in W is at least 4, then there always exists two neighbors t and z (see Figure 2.4) of u on W such that $W' = W + \{(t, z)\} - \{(u, t), (u, z)\}$ is also a closed walk (if (u, t) or (u, z) appears more than once in W , we only remove one copy). We have by hypothesis, that $fx^{W'} < f_0$, contradicting the validity of the inequality. Therefore all closed walks $W \in \mathcal{W}_f^-$ which do satisfy condition (a) and not condition (b) satisfy $x(\delta(u)) = 2$ for some $u \in V$.

■

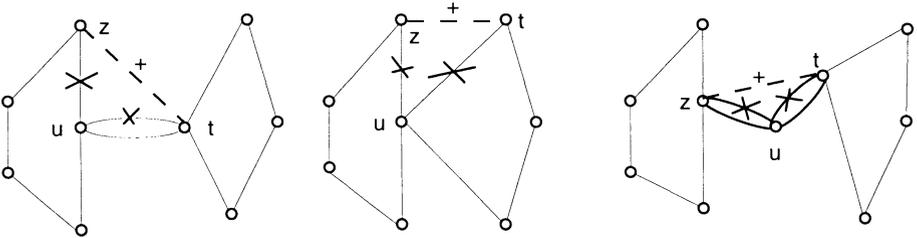


Figure 2.4. Examples of “shortcuts”

Corollary 18 Let $fx \geq f_0$ be a facet defining TT inequality for $GTSP(n)$. Then

- (a) for every edge $e \in E$ there exists a closed walk $W \in \mathcal{W}_f^-$ such that $e \in W$
- (b) for every node $v \in V$ there exists a closed walk $W \in \mathcal{W}_f^-$ such that v has degree at least 4 in W .

Remark 19 All facet defining TT inequalities $fx \geq f_0$ for GTSP(n) have the following structural property. There exists a unique partition $V_1, \dots, V_j, \dots, V_k$ of the vertex set V such that $f_e = f_{ij} > 0$ for all $e \in (V_i : V_j)$, $i \neq j$, and $f_e = 0$ if and only if $e \in E(V_j)$ for some j , $1 \leq j \leq k$.

Proof: Let $V_1, \dots, V_j, \dots, V_k$ be the vertex sets of the connected components of the graph $G^0 = (V, E^0)$, where $E^0 = \{e \in E : f_e = 0\}$. Let $e \in E(V_j)$ and $W_e \in \mathcal{W}_f^-$ such that $e \in W_e$. Assume $f_e > 0$. There is a path in V_j linking the extremities of e and for which all the edges have a coefficient of 0. The closed walk W^* obtained from W_e by removing e and by adding the edges of that path is such that $fx^{W^*} < f_0$, which contradicts the validity of $fx \geq f_0$. Now let e and e' be two edges of $(V_i : V_j)$ and assume $f_{e'} < f_e$. Let $W_e \in \mathcal{W}_f^-$ such that $e \in W_e$. The closed walk W^* obtained from W_e by removing e and adding the edges of a path in V_i from the extremity of e to the one of e' it contains, the edge e' and the edges of a path in V_j from the extremity of e' to the one of e it contains, does not satisfy the inequality $fx \geq f_0$. Note that the existence of W_e in all cases is guaranteed since otherwise the inequality would be $x_e \geq 0$, which is not tight triangular. ■

If, in the partition of Remark 19, all the sets have cardinality 1, we say that the inequality $fx \geq f_0$ is *simple*. As we will see, it is often convenient to study, first, simple inequalities and then to obtain the more general inequalities via the *node lifting* procedures described in Section 4.3.

Definition 20 For every ordered triplet $\langle u, t, z \rangle$ of distinct nodes in V , we call shortcut on $\langle u, t, z \rangle$ the vector $s_{utz} \in \mathbb{R}^E$ defined by:

$$s_{utz}(e) = \begin{cases} 1 & \text{if } e = (t, z) \\ -1 & \text{if } e \in \{(u, t), (u, z)\} \\ 0 & \text{otherwise} \end{cases}$$

Let $W \in \mathcal{W}_f^-$ with strictly more than n edges. The following lemma will be very useful:

Lemma 21 Let $fx \geq f_0$ be a TT inequality supporting for GTSP(n) and let $W \in \mathcal{W}_f^-$ be a closed walk with $t > n$ edges containing a certain edge e^* . For every vertex u with degree $k \geq 4$ in W , there exists a shortcut s_{uvw} such that $x^W + s_{uvw} \in \mathcal{W}_f^-$, contains e^* and has $t - 1$ edges.

Proof: Easy, see [618]. ■

Corollary 22 *Let $fx \geq f_0$ be a facet defining TT inequality for $GTSP(n)$. Then, for each edge $e \in E$, there exists $H \in \mathcal{H}_f^-$ that contains e .*

The following lemma shows how to transform a facet inducing inequality for $STSP(n)$ into an equivalent one in TT form.

Lemma 23 *Let $hx \geq h_0$ be a facet defining inequality for $STSP(n)$. An inequality $fx \geq f_0$ equivalent to $hx \geq h_0$ is tight triangular if and only if $f = \pi A + \pi_0 h$ and $f_0 = 2 \sum_{u \in W} \pi_u + \pi_0 h_0$ where (π, π_0) satisfy*

$$\pi_u = \frac{1}{2} \pi_0 \max\{h(v, w) - h(u, v) - h(u, w) : u, v, w \in V, u \neq v \neq w\} \quad (12)$$

Proof: It is straightforward to check that if (π, π_0) satisfies Equation (12), the inequality $fx \geq f_0$ is tight triangular. Suppose that $fx \geq f_0$ is tight triangular. Condition (a) of the definition of tight triangularity imposes that

$$\pi_u \geq \frac{1}{2} \pi_0 \max\{h(v, w) - h(u, v) - h(u, w) : u, v, w \in V, u \neq v \neq w\} \quad (13)$$

The second condition of that definition implies the existence of a triple of distinct vertices u, v, w for which (13) holds with equality. ■

A *basis* of a facet defining inequality $fx \geq f_0$ for $GTSP(n)$ is a set \mathcal{B}_f of $|E|$ closed walks in \mathcal{W}_f^- whose incidence vectors are linearly independent. A closed walk W is *almost Hamiltonian* in u if u has degree 4 in W and all other vertices have degree 2.

Definition 24 *A basis \mathcal{B}_f of a facet defining inequality $fx \geq f_0$ of $GTSP(n)$ is called canonical if it contains $|E| - n$ Hamiltonian cycles and n almost Hamiltonian walks.*

Theorem 25 *A non-trivial TT inequality $fx \geq f_0$ which is facet defining for $STSP(n)$ defines a facet of $GTSP(n)$.*

Proof: We just show how to build a basis \mathcal{B}_f . Since the inequality is facet defining for $STSP(n)$, there exists $|E| - n$ linearly independent Hamiltonian cycles in \mathcal{H}_f^- . We now construct n almost Hamiltonian walks, one for each $u \in V$. Let $e = (v, w) \in \Delta_f(u)$ and $\Gamma_e \in \mathcal{H}_f^-$ a Hamiltonian cycle containing e , which exists as the inequality is not equivalent to the trivial inequality $x_e \geq 0$. Let $W_u = \Gamma_e - \{e\} + \{(u, v), (u, w)\}$. All we have to prove is that the previously mentioned $|E| - n$ Hamiltonian cycles together with these n almost Hamiltonian walks are linearly independent. This can be found in [618]. ■

Note that the facet inducing inequality $x_e \leq 1$ for $STSPP(n)$ has a TT-form which is $x(\delta(\{u, v\})) \geq 2$ where $e = (u, v)$, which we already know is facet inducing for $GTSPP(n)$. The following theorem shows the strong relationship between $STSPP(n)$ and $GTSPP(n)$.

Theorem 26 *Every non-trivial facet of $STSPP(n)$ is contained in exactly $n + 1$ facets of $GTSPP(n)$, n of which are defined by the degree inequalities.*

Proof: Let F be a non-trivial facet of $STSPP(n)$. Then F belongs to the n facets of $GTSPP(n)$ defined by the degree inequalities $x(\delta(\{v\})) \geq 2$, for all $v \in V$. Every other facet of $GTSPP(n)$ containing F is defined by a TT-inequality. By Lemma 23, there is only one such facet and the theorem follows. ■

This theorem shows that the polyhedral structure of $STSPP(n)$ is very closely related to that of $GTSPP(n)$.

A corollary of this theorem is that the TT-form is a canonical way of expressing the facets of $STSPP(n)$. Therefore, together with Lemma 23, it leads to a $O(n^3)$ algorithm to recognize whether or not two facet inducing inequalities of $STSPP(n)$ define the same facet which, as already mentioned, it is not as good as Margot's or Queyranne and Wang's $O(n^2)$ algorithms.

We will study the polyhedral structure of $GTSPP(n)$ in more detail in the next sections. At this point, the reader may want to skip the rest of this section which addresses very technical aspects of the study of $STSPP(n)$. In particular, we address the problem of finding sufficient conditions for a facet defining inequality of $GTSPP(n)$ to define a facet of $STSPP(n)$. Note that, at the time of writing this chapter, no tight triangular facet inducing inequality for $GTSPP(n)$ is known that is not also one for $STSPP(n)$ (except of course the degree inequalities).

From Definition 24, a facet inducing TT inequality for $GTSPP(n)$ defines a facet for $STSPP(n)$ if and only if it has a canonical basis. In the following we investigate sufficient conditions for the existence of a canonical basis. The following remark is central in the approach to such conditions.

Remark 27 *Let $fx \geq f_0$ be a TT facet defining inequality for $GTSPP(n)$ and \mathcal{B}_f be one of its bases. Let $\{W_u : u \in V\}$ be a set of n almost Hamiltonian walks of $\mathcal{W}_{\bar{f}}$, where W_u is almost Hamiltonian in u . If every closed walk of \mathcal{B}_f can be reduced to a cycle of $\mathcal{H}_{\bar{f}}$ by using only shortcuts obtained by a linear combination of the incidence vectors of elements of $\mathcal{A}_f = \mathcal{H}_{\bar{f}} \cup \{W_u : u \in V\}$, then \mathcal{A}_f contains a canonical basis of $fx \geq f_0$. That is, all the incidence vectors of the closed walks T of \mathcal{B}_f can be expressed as a linear combination of the incidence vectors*

of elements in \mathcal{A}_f . In general, it is sufficient that every shortcut can be obtained as a linear combination of the incidence vectors of elements in \mathcal{A}_f with coefficients in \mathbb{R} . To obtain simple sufficient conditions we will restrict the coefficients to $\{-1, 0, 1\}$.

Let $e = (u, v)$ and $e' = (w, y)$ be two distinct edges of E . We say that e and e' are f -adjacent if there exists a Hamiltonian cycle in \mathcal{H}_f^- containing both e and e' . Let z be a vertex in V , we say that e and e' are f -adjacent in z if:

- (i) e and e' belong to $\Delta_f(z)$;
- (ii) There exists a closed walk $W_z \in \mathcal{W}_f^-$ almost Hamiltonian in z that contains (z, u) , (z, v) , (z, w) and (z, y) ;
- (iii) $W_z - \{(z, u), (z, v)\} + \{e\}$ is a Hamiltonian cycle (and therefore so is $W_z - \{(z, w), (z, y)\} + \{e'\}$).

A set of edges $J \subset E$ is said to be f -connected if for every pair of distinct edges $e_1, e_2 \in J$, there exists a sequence of edges $e'_1 = e_1, e'_2, \dots, e'_{k-1}, e'_k = e_2$ in J , such that for $i = 1, \dots, k-1$, e'_i is f -adjacent to e'_{i+1} . A set of edges $J \subset E$ is said to be f -connected in z if for every pair of distinct edges, e_1 and $e_2 \in J$, there exists a sequence of edges $e'_1 = e_1, e'_2, \dots, e'_{k-1}, e'_k = e_2$ in E (not necessarily in J), such that for $i = 1, \dots, k-1$, e'_i is f -adjacent in z to e'_{i+1} . Observe that the notion of f -connectivity in z is weaker than that of f -connectivity. Any subset of a f -connected set in z is also f -connected in z .

Lemma 28 *Let $fx \geq f_0$ be a TT facet defining inequality of GTSP(n). If $\Delta_f(u)$ is f -connected in u for every $u \in V$, then $fx \geq f_0$ has a canonical basis, hence it is facet defining for STSP(n).*

Proof: Let $u \in V$ and $\{W_u : u \in V\}$ be any set of n almost Hamiltonian walks of \mathcal{W}_f^- . This set always exists, by Corollary 22, since $fx \geq f_0$ is tight triangular it can be constructed as in the proof of Theorem 25. By Lemma 21 there exists a shortcut $s_{u\bar{y}\bar{z}}$ with $(\bar{y}, \bar{z}) \in \Delta_f(u)$ that can be used to reduce W_u to a Hamiltonian cycle $\Gamma \in \mathcal{H}_f^-$. We therefore have $s_{u\bar{y}\bar{z}} = x^\Gamma - x^{W_u}$, hence $s_{u\bar{y}\bar{z}}$ is a linear combination with coefficients in $\{-1, 1\}$ of incidence vectors of elements from $\mathcal{A}_f = \mathcal{H}_f^- \cup \{W_u : u \in V\}$. If $|\Delta_f(u)| = 1$, we are done. Else let $e = (v, w)$ and $e' = (y, z)$ be two distinct edges of $\Delta_f(u)$ which are f -adjacent in u . Let us assume that the shortcut s_{uyz} is a linear combination with coefficients in $\{-1, 1\}$ of incidence vectors of elements from \mathcal{A}_f . Since e and e' are f -adjacent in u , there exist a closed walk $W'_u \in \mathcal{W}_f^-$ almost Hamiltonian in u which contains (u, v) , (u, w) , (u, y) and (u, z) . We have $\Gamma_1 =$

$W'_u - \{(u, y), (u, z)\} + \{(y, z)\} \in \mathcal{H}_f^-$ and $\Gamma_2 = W'_u - \{(u, v), (u, w)\} + \{(v, w)\} \in \mathcal{H}_f^-$, by triangular inequality. The shortcut s_{uvw} can be expressed as: $s_{uvw} = x^{\Gamma_2} - x^{\Gamma_1} + s_{uyz}$, hence it is a linear combination with coefficients in $\{-1, 1\}$ of incidence vectors of closed walks from \mathcal{A}_f . By the assumption that $\Delta_f(u)$ is f -connected in u , it follows that for all $u \in V$ and all $(v, w) \in \Delta_f(u)$, the shortcuts s_{uvw} can be expressed as a linear combination of the incidence vectors of elements in \mathcal{A}_f , and by Remark 27 the lemma follows. ■

The conditions of this lemma are too restrictive in general. We give now a weaker version, which is the one that is in general used in proving facet inducing results for $STSP(n)$ from results for $GTSP(n)$.

Lemma 29 *Let $fx \geq f_0$ be a TT facet defining inequality of $GTSP(n)$. If there exists a basis \mathcal{B}_f of $fx \geq f_0$ and for every $u \in V$ there exists a nonempty set of edges $J_u \subseteq \Delta_f(u)$, f -connected in u , such that every closed walk $W \in \mathcal{B}_f$ can be reduced to a tour of \mathcal{H}_f^- by using only shortcuts from the set $\{s_{uvw} : (v, w) \in J_u, u \in V\}$, then $fx \geq f_0$ has a canonical basis, hence it is facet defining for $STSP(n)$.*

Proof: For every $u \in V$, let (v, w) be any edge in J_u . By Corollary 22 there exists $\Gamma \in \mathcal{H}_f^-$ containing the edge (v, w) . Let W_u be the almost Hamiltonian walk defined by $W_u = \Gamma - (v, w) + (u, v) + (u, w)$. By Remark 27 and a process analogous to that of the proof of the preceding lemma, it follows that the set $\mathcal{A}_f = \mathcal{H}_f^- \cup \{W_u : u \in V\}$ contains a canonical basis for $fx \geq f_0$ and the lemma follows. ■

We now turn to the problem of deriving facet inducing inequalities for $GTSP(n)$ and for $STSP(n)$ from other such inequalities. The first way we will study is via *composition* of two facet inducing inequalities, the second will be via what we will call *node lifting*.

4.2. Composition of inequalities

In [616], a composition of valid inequalities known as the *s-sum* is defined. We will focus here on the case of *2-sums*. For sake of simplicity we will assume that the inequalities $fx \geq f_0$ we deal with are *simple*, that is $f_e > 0$ for all $e \in E$.

For the reader with some knowledge of the traveling salesman polytope, the *2-sum* composition we describe here, can be used to build clique tree inequalities from comb inequalities.

We say that two edge weighted graphs $G^1 = (V^1, E^1, f^1)$ and $G^2 = (V^2, E^2, f^2)$ are *isomorphic* if there exists a one to one correspondence between their vertex sets that preserves the weights on the edges.

Definition 30 Let $f^1x \geq f_0^1$ and $f^2x \geq f_0^2$ be two TT inequalities valid for $GTSP(n_1) = (V^1, E^1)$ and $GTSP(n_2) = (V^2, E^2)$, respectively. Let $e_1 = (u_1, v_1) \in E_1$ and $e_2 = (u_2, v_2) \in E_2$ be two edges such that $f_{e_1}^1 = f_{e_2}^2 = \epsilon > 0$. Then a 2 – sum inequality obtained by identifying u_1 with u_2 (called now u) and v_1 with v_2 (called now v) is the inequality $fx \geq f_0^1 + f_0^2 - 2\epsilon$ defined on $G = K_n$ with $n = n_1 + n_2 - 2$ as follows:

- 1 $V_n = (V^1 \setminus \{u_1, v_1\}) \cup (V^2 \setminus \{u_2, v_2\}) \cup \{u, v\}$
- 2 The weighted subgraph $G(V^i)$ is isomorphic to the weighted graph G^i , with u and v corresponding to u_i and v_i respectively, for $i = 1, 2$.
- 3 The coefficients of the edges with one endpoint in V^1 and the other in V^2 , that we call the crossing edges of the 2 – sum, are computed in the following way
 - (a) Order the crossing edges $e_1, \dots, e_j, \dots, e_k$
 - (b) For $j = 1$ to k , let T^j be a minimum f -length closed walk among all closed walks of G containing e_j that uses only edges from $E_1 \cup E_2 \cup \{e_1, \dots, e_j\}$. Let f_{e_j} be such that the f -length of T^j is $f_0^1 + f_0^2 - 2\epsilon$.

The condition $f_{e_1}^1 = f_{e_2}^2$ is not restrictive since we can always scale one of the inequalities in order to obtain this condition.

The procedure described at point 3 of Definition 30 is called *sequential lifting* in the literature (see Padberg [640]). In general the coefficients of the crossing edges depend on the lifting sequence. If it is not the case we say that the 2 – sum is *stable*. Most of the known inequalities that come from 2 – sums are stable.

If, in the sequential lifting, the coefficients of the crossing edges are such that the minimum over all closed walks is the same as the minimum over all Hamiltonian cycles, then the 2 – sum inequality is called *h-liftable*.

For the sake of simplicity, when obvious, we do not mention the correspondence between the two composing graphs and the corresponding elements of the isomorphic subgraphs of the 2 – sum.

Let $fx \geq f_0$ be an inequality supporting for $GTSP(n)$. A vertex $v \in V$ is said to be *k-critical* for that inequality if the f -length of a minimum f -length closed walk of $K_n \setminus \{v\}$ is $f_0 - k$.

Remark 31 In the previous definition k must satisfy $k \leq 2 \cdot \min\{f_e : e \in \delta(v)\}$. We will only consider two types of vertices, the 0-critical and the \bar{k} -critical with $\bar{k} = 2 \cdot \min\{f_e : e \in \delta(v)\}$

We assume the 2 – sum is performed as described in Definition 30 and the notation is the one used in that definition. The following theorem, the proof of which can be found in [618], states conditions under which two facet inducing inequalities yield, by 2 – sum, another facet inducing inequality.

Theorem 32 *Let $f^1x \geq f_0^1$ and $f^2x \geq f_0^2$ be two TT facet inducing inequalities for $GTSPP(n_1)$ and $GTSPP(n_2)$, respectively. The 2 – sum inequality $fx \geq f_0$ is facet defining for $GSTPP(n)$ if v_1 is 2ϵ -critical for $f^1x \geq f_0^1$ and at least one of the two vertices u_2 and v_2 is 2ϵ -critical for $f^2x \geq f_0^2$.*

The next theorem is the corresponding theorem for $STSP(n)$.

Theorem 33 *Let $f^1x \geq f_0^1$ and $f^2x \geq f_0^2$ be two TT facet inducing inequalities of $STSP(n_1)$ and $STSP(n_2)$, respectively. The 2 – sum inequality $fx \geq f_0$ is facet defining for $STSP(n)$ if it is h-liftable and:*

(a) v_1 is 2ϵ -critical for $f^1x \geq f_0^1$,

(b) $\delta(u_2)$ is f^2 -connected,

and either (Case (A))

(c') u_2 is 2ϵ -critical for $f^2x \geq f_0^2$,

(d') $\delta(v_1)$ is f^1 -connected,

or (Case (B))

(c'') v_2 is 2ϵ -critical for $f^2x \geq f_0^2$,

(d'') $\delta(u_1)$ is f^1 -connected,

(e'') there exists a Hamiltonian cycle $H_1 \in \mathcal{H}_{f^1}^-$ containing edge (u_1, v_1) and any edge $e_1 \in \Delta_{f^1}(v_1)$,

(f'') there exists a Hamiltonian cycle $H_2 \in \mathcal{H}_{f^2}^-$ containing edge (u_2, v_2) and any edge $e_2 \in \Delta_{f^2}(v_2)$,

Proof: See [618] ■

The 2 – sum procedure can be carried out recursively. The only difficulty is to keep track of the “criticality” of the vertices after the composition. Various technical lemmas dealing with this aspect are given in [618].

4.3. Node lifting

As mentioned earlier, when studying a given family of facet inducing inequalities for STSPP(n), it is easier to first study the simple inequalities of the class. Then the result is generalized (*lifted*) to the non simple inequalities of that family. The tools to obtain these results are the aim of this section. Remark 19 states that a facet inducing inequality in TT form is such that the the components of the subgraph on the edges with zero coefficient are cliques. This may suggest that the only *lifting* operation is the replacement of nodes by cliques. This is not the case. A comb (see next section) with no node inside the handle not contained in any tooth yields a facet inducing inequality. The same comb with a node in the handle not in any tooth also does.

Let $fx \geq f_0$ be a simple valid inequality for $GTSP(n)$. An inequality $f^*x \geq f_0$ (same right hand side) is obtained by *clique lifting* from a simple one if each node $u \in V$ is replaced by a clique $K_{k_u}^u = (V^u, E^u)$ of $k_u \geq 1$ vertices and:

- 1 $f_e^* = 0$ for all $e \in E^u$ for some $u \in V$,
- 2 for all $e \in (V^u : V^v)$, $f_e^* = f_{uv}$, for all $u, v \in V$.

That is all the edges of the newly formed cliques get a coefficient of zero, the edges between two newly formed cliques inherit the coefficient of the corresponding edge in the original graph.

Theorem 34 *Let $fx \geq f_0$ be a TT facet inducing inequality of $GTSP(n)$. Any inequality obtained from $fx \geq f_0$ by clique lifting is also facet inducing for $GTSP(n^*)$, where $n^* > n$ is the number of vertices of the resulting graph.*

Proof: The inequality is obviously valid since any closed walk on the “extended” graph induces a closed walk on K_n . It is supporting since any closed walk T of K_n can be extended to a closed walk T' on the extended graph with $f(T) = f^*(T')$. Assume $f^*x \geq f_0$ is not facet inducing. Since it is valid and supporting, there is a facet that contains the face it induces. Let $gx \geq f_0$ be a linear representation of that facet (by scaling we can always assume the right hand side to be f_0). Note that we have $\mathcal{W}_{f^*}^- \subset \mathcal{W}_g^-$. Let $e \in \gamma(V_u)$ for some $u \in V_n$ and $T' \in \mathcal{W}_{f^*}^-$, so $T' \in \mathcal{W}_g^-$. The closed walk $T' + 2\{e\}$ is also in $\mathcal{W}_{f^*}^-$ and therefore in \mathcal{W}_g^- , which implies that $g_e = f_e = 0$. We are therefore left with the edges which correspond to some edge of the simple inequality, and proving that $g_e = f_e^* = f_e$ for these amounts to proving that the simple inequality is facet inducing. ■

Things are not that easy for $STSP(n)$. The clique lifting can be seen as a recursive application of an operation we will call *zero-lifting of a node*. We define now what we mean by *node lifting*.

Let $K_n = (V_n, E_n)$ and $K_{n^*} = (V_{n^*}, E_{n^*})$, $n^* \geq n$ be two complete graphs. Let $V_n = \{u_1, u_2, \dots, u_n\}$ and $V_{n^*} = V_n \cup \{u_{n+1}, \dots, u_{n^*}\}$. We say that the inequality $f^*x \geq f_0^*$ defined on $\mathbb{R}^{E_{n^*}}$ has been obtained by *node-lifting* of $fx \geq f_0$ defined on \mathbb{R}^{E_n} if

$$f_{(u_i u_j)}^* = f_{(u_i u_j)} \text{ for all } 1 \leq i < j \leq n.$$

The special case where $f_{(uv)}^* = 0$ for some $u \in V_n$ and all $v \in \{u_{n+1}, \dots, u_{n^*}\}$ is called *zero-lifting of node u* . Note that by triangularity one then have $f_{(vw)}^* = 0$ for all $v, w \in \{u_{n+1}, \dots, u_{n^*}\}$. For a TT inequality obtained by zero-lifting, the following proposition is a direct consequence of Remark 19.

Proposition 35 *For any facet inducing TT inequality $f^*x \geq f_0$ obtained from $fx \geq f_0$ by zero-lifting of node u , the following holds:*

- 1 $f_{(vw)}^* = f_{(vu)}$ for all $v \in V_n - \{u\}$ and $w \in \{u_{n+1}, \dots, u_{n^*}\}$,
- 2 $\Delta_{f^*}(w) = \Delta_f(u) \cup \bigcup \{\delta(w') \setminus \{(w', w)\} : w' \in \{u, u_{n+1}, \dots, u_{n^*}\}, w' \neq w\}$ for all $w \in \{u_{n+1}, \dots, u_{n^*}\}$, where $\delta(w')$ is a taken in E_{n^*} .

There are node-liftings which are not zero-liftings. For example the inequality of Figure 2.5 (b) is what is called in [618] a *1-node lifting* of the inequality of Figure 2.5 (a), and that of Figure 2.5 (c) a *1-node-lifting* of that of Figure 2.5 (b). In that figure the coefficient of the shown edges is the one shown; for the others the coefficient is that of a shortest path, in term of coefficient of the shown edges, between their end point. For example, in Figure 2.5 (a) a non shown edge linking a node of the top cycle to one of the bottom cycle and not shown has a coefficient of 3, since a shortest path between its extremities consists of an horizontal edge and a vertical one. All right hand sides are 10. Note that these three inequalities are what we will call later on *comb inequalities*.

A *1-node lifting* of an inequality $fx \geq f_0$ defined on \mathbb{R}^{E_n} is an inequality $f^*x \geq f_0$ (same right hand side) defined on $\mathbb{R}^{E_{n+1}}$ and which is not a zero-lifting. The coefficients $f_{vu_{n+1}}^*$ for $v \in V_n$, u_{n+1} being the new vertex, are not defined by this definition. They must satisfy the following necessary conditions.

Theorem 36 *Let $f^*x \geq f_0^*$ be a facet defining inequality for $GTSP(n+1)$ which is obtained by 1-node lifting of a TT facet inducing inequality for $GTSP(n)$; then the following conditions hold:*

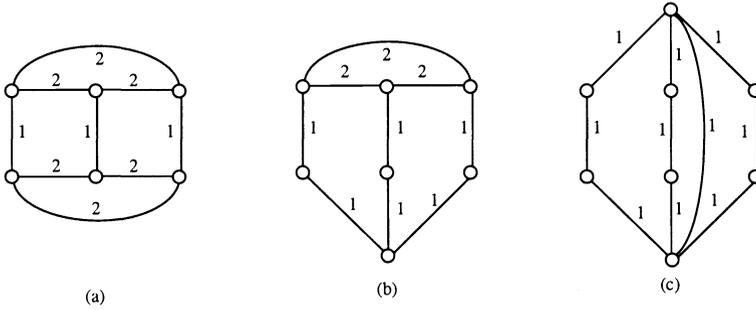


Figure 2.5. Examples of 1-node liftings

- 1 $f^*x \geq f_0^*$ is tight triangular,
- 2 $f_0^* = f_0$
- 3 for all $e \in \Delta_{f^*}(u_{n+1})$ there exists $e' \neq e, e' \in \Delta_{f^*}(u_{n+1})$ and $\Gamma \in \mathcal{H}_f^-$ such that both e and e' belong to Γ .
- 4 every connected component of the graph $(V_n, \Delta_{f^*}(u_{n+1}))$ contains at least one odd cycle.

Proof: By Theorem 17, the only non TT facet inducing inequalities for $GTSP(n + 1)$ are the non negativity constraints or the degree constraints, and none can be obtained by 1-node lifting, so Statement 1 follows. We can only have $f_0^* \geq f_0$. Since the inequality is TT, let $e \in \Delta_{f^*}(u_{n+1})$. There is a closed walk W of \mathcal{W}_f^- that contains $e = (u, v)$, and $W' = W - \{e\} + \{(u, u_{n+1}), (v, u_{n+1})\}$ is such that $f^*(W') = f_0$, which implies $f_0^* = f_0$. For the rest see [618]. ■

Facet inducing results on 1-node lifting for $GTSP(n)$ and $STSP(n)$ can be found in [618]. One can also find there some results on zero-liftings for $STSP(n)$ (we have settled here the case of $GTSP(n)$).

We now give a 2-node lifting procedure, known as *edge cloning*, which concerns two vertices and that cannot be obtained by successive 1-node liftings.

Let $fx \geq f_0$ be a TT inequality defined on \mathbb{R}_n^E and $e \in E_n$. We say that the inequality $f^*x \geq f_0^*$ defined on \mathbb{R}_{n+2h}^E , with $h \geq 1$, is obtained from $fx \geq f_0$ by cloning h times the edge e if it is obtained by node lifting of $fx \geq f_0$ and, assuming without loss of generality that $e = (u_{n-1}, u_n)$:

$$f_0^* = f_0 + 2hf_e,$$

$$f_{u_i u_{n+j}}^* = \begin{cases} f_{u_i u_{n-1}} & \text{for } 1 \leq i \leq n-2, 1 \leq j \leq 2h-1 \text{ and } j \text{ odd,} \\ f_{u_i u_n} & \text{for } 1 \leq i \leq n-2, 1 \leq j \leq 2h-1 \text{ and } j \text{ even,} \end{cases}$$

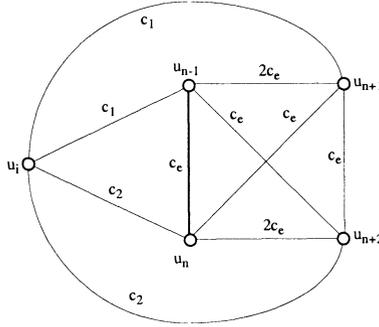


Figure 2.6. Examples of edge cloning

$$f_{u_{n+i}u_{n+j}}^* = \begin{cases} 2f_e & \text{for } -1 \leq i < j \leq 2h \text{ and } j - i \text{ even,} \\ f_e & \text{for } -1 \leq i < j \leq 2h \text{ and } j \text{ odd,} \end{cases}$$

Figure 2.6 shows the coefficients of the inequality with edge e cloned and $h = 1$. Let $fx \geq f_0$ be a TT inequality valid for $GTSP(n)$. We say that $e = (u, v)$ is f -clonable if:

- 1 u and v are $2f_e$ -critical for $fx \geq f_0$,
- 2 the f -length of any closed walk W of K_n is at least $f_0 + (d - 2)f_e$ where $d = \min(|(\delta(u) \cap T)|, |(\delta(v) \cap T)|)$.

This definition is more restrictive than that given in [618]. The first condition is not required there, but is required in all the subsequent theorems. In Figure 2.7 the coefficients of the edges are shown next to them, if not equal to 1. The edges not shown have a coefficient equal to that of a shortest path, with edges among those shown, between their extremities. The first inequality is a comb inequality, the right hand side is 10, only the three edges in bold are f -clonable. The other edges in that example are not f -clonable because the six vertices incident to the bold edges are 2-critical and the other two are 0-critical, therefore no other edge has its two extremities $2f_e$ -critical. An example of violation of the second condition is given in the second example (the right hand side is 14). The second inequality is what we will call, later on, a path inequality. None of the edges is f -clonable. All the vertices except the top and bottom one are 2-critical, the other two vertices being 0-critical. In particular, vertices u and v (in black) are 2-critical. The shown closed walk W has f -value $14 = f_0$ and both extremities of e have degree 4 in W , which shows that edge e is not clonable. The case of the other edges follows by symmetry.

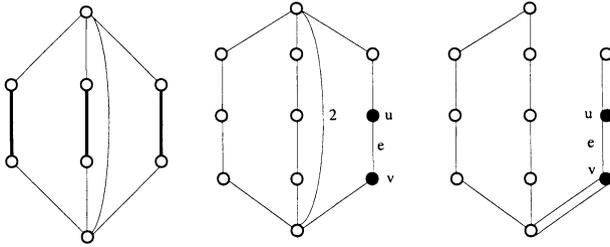


Figure 2.7. Examples of clonable and non clonable edges

Theorem 37 Let $fx \geq f_0$ be a TT facet inducing inequality for $GTSP(n)$ and $e = (u_{n-1}, u_n)$ be a f -clonable edge. Then the following holds:

- 1 the inequality $f^*x \geq f_0^*$, obtained from $fx \geq f_0$ by cloning $h > 1$ times edge e , is facet inducing for $GTSP(n + 2h)$,
- 2 the edges of $(\{u_{n-1}, u_{n+1}, \dots, u_{n+2h-1}\} : \{u_n, u_{n+2}, \dots, u_{n+2h}\})$ are f^* -clonable,
- 3 if $e' = (z_1, z_2) \neq e$ is an edge in E_n such that z_1 and z_2 are $2f_{e'}$ -critical for $fx \geq f_0$, then z_1 and z_2 are $2f_{e'}^*$ -critical for $f^*x \geq f_0^*$.

Proof: See the proofs of Lemma 4.11 and Theorem 4.12 in [618]. ■

Note that Statement 2 implies that the “new” vertices are $2f_e^*$ -critical. The purpose of Statements 2 and 3 is to fix the status of the various vertices and edges in view of further edge clonings. The corresponding theorem for $STSP(n)$ is:

Theorem 38 Let $fx \geq f_0$ be a non-trivial TT facet inducing inequality for $STSP(n)$ and let $e = (u_{n-1}, u_n)$ be a f -clonable edge. Then the inequality $f^*x \geq f_0^*$ obtained from $fx \geq f_0$ by cloning $h > 1$ times edge e is facet inducing for $STSP(n + 2h)$

Proof: See the proof of Theorem 4.13 in [618]. ■

The family of chain inequalities defined by Padberg and Hong [643] can be obtained from the comb inequalities by edge cloning.

We now turn to the study of specific classes of facet inducing inequalities of $GTSP(n)$ and of $STSP(n)$.

5. The Comb inequalities

Comb inequalities were first discovered, in a more restrictive form, by Chvátal [195] who derived them from the 2-matching constraints of

Edmonds [265]. They were generalized to the current form and shown to be facet inducing for $STSP(n)$ by Grötschel and Padberg [403].

A *comb inequality* is usually defined by a set $H \subset V$, called *handle*, and an odd number $t \geq 3$ of vertex subsets $\{T_1, T_2, \dots, T_t\}$, called *teeth*, such that:

$$H \cap T_i \neq \emptyset \quad \text{for } i = 1, \dots, t \quad (14)$$

$$T_i \setminus H \neq \emptyset \quad \text{for } i = 1, \dots, t \quad (15)$$

$$T_i \cap T_j = \emptyset \quad \text{for } 1 \leq i < j \leq t \quad (16)$$

Conditions (14) and (15) say that every tooth T_i intersects properly the handle H , Condition (16) states that no two teeth intersect.

The corresponding *comb inequality* is:

$$x(\delta(H)) + \sum_{j=1}^t x(\delta(T_j)) \geq 3t + 1 \quad (17)$$

Remark 39 *Comb inequalities are better known in the following form:*

$$x(\gamma(H)) + \sum_{i=1}^t x(\gamma(T_i)) \leq |H| + \sum_{i=1}^t (|T_i| - 1) - (t + 1)/2 \quad (18)$$

This form still has a few adepts because:

- *It is a rank type inequality for the independence system associated with the monotoneization of the polytope we mentioned earlier.*
- *It is a mod-2 cut, fact which is used in some separation routines (see Caprara, Fischetti and Letchford [161], Letchford [559]).*
- *It is sometimes sparser than the other form, fact which is useful when solving large linear programs.*

On the other hand it is not in TT form. We will never use this form.

Note that Inequality (17) is closed under taking complements of sets since $x(\delta(S)) = x(\delta(V \setminus S))$ for all proper subset S of V . The definition of a comb inequality is not very satisfactory since Condition (16) is not closed under taking complements. There is no real way around this problem as long as we do not deal directly with the edge sets. The most satisfactory one is to require that the given sets satisfy the conditions after eventually replacing some of them by their complements. Figure 2.8 gives two sets \mathcal{S} that give the same comb inequality, the first is a set as described in (14) to (16), the other with T_3 replaced by its complement.

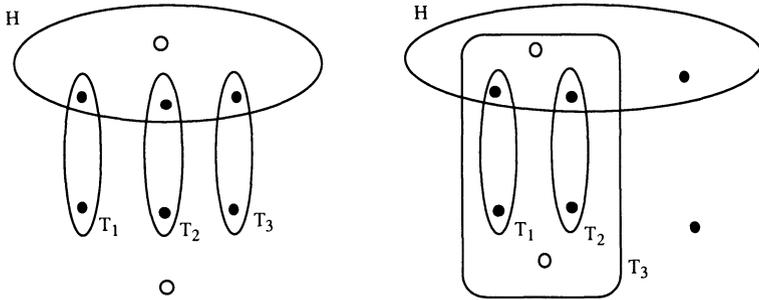


Figure 2.8. Example of a comb ($t = 3$)

Figure 2.8 enables us to define our convention for figures that will hold throughout this chapter. Sets with a black point have to be nonempty, those with a white filled point may or may not have nodes in them, those with no points must be empty. A point does not represent a unique node but a set of nodes in that position.

We will give three proofs of validity of comb inequalities, an algebraic (classical) one and two others that will give more insight into what these inequalities say in term of closed walks. We have so far used the term *tight* in various ways. We now use it for sets and will use it frequently. A subset $S \subset V$ is said to be *tight* for a Hamiltonian cycle or a closed walk Γ if $|\Gamma \cap \delta(S)| = 2$. In other terms, if x^Γ is the incidence vector of Γ , the corresponding subtour elimination inequality is tight, i.e. $x^\Gamma(\delta(S)) = 2$.

Theorem 40 *Comb inequalities are valid for GTSP(n) (and therefore for STSP(n)).*

Proof: (1) It is easy to check that we have:

$$x(\delta(H)) + \sum_{j=1}^t x(\delta(T_j)) \geq \quad (19)$$

$$1/2 \left[\sum_{j=1}^t x(\delta(H \cap T_j)) + \sum_{j=1}^t x(\delta(T_j \setminus H)) + \sum_{j=1}^t x(\delta(T_j)) \right] \geq 3t$$

Now the left part of (19), $x(\delta(H)) + \sum_{j=1}^t x(\delta(T_j))$, is an even integer as sum of even integers, when x represents a closed walk. Since $3t$ is odd, $3t$ can be replaced by $3t + 1$. ■

This easy proof does not really explain why any Hamiltonian cycle or any closed walk, intersects these $t+1$ coboundaries in at least $3t+1$ edges and not in $2t+2$ obtained by adding up the $t+1$ corresponding subtour elimination inequalities. The following proof, taken from Naddef and Pochet [615], will give more insights into the validity of comb inequalities.

Proof: (2) A non-algebraic and inductive proof of validity of comb inequalities. The comb inequality is assumed to be described by sets that satisfy Conditions (14) to (16). Assume first that $t=3$. Take a Hamiltonian cycle Γ , if $|\Gamma \cap \delta(T_j)| = 2$ for $j=1, 2$ and 3 , then $\Gamma \cap (T_i \setminus H : T_i \cap H) \neq \emptyset$, so $|\Gamma \cap \delta(H)| \geq 3$. Since $|\Gamma \cap \delta(H)|$ must be even we have $|\Gamma \cap \delta(H)| \geq 4$ and Inequality (17) is satisfied by all x representing such a cycle. If $|\Gamma \cap \delta(T_{j^*})| \neq 2$, for some j^* , $|\Gamma \cap \delta(T_{j^*})| \geq 4$, the 3 other sets having at least 2 edges of their coboundaries in Γ , again Inequality (17) is satisfied by all x representing such a cycle. Proving validity for the general case can be done by induction on the number of teeth $t \geq 5$. Assume validity for combs with $t-2$ teeth. Take a Hamiltonian cycle Γ . If $|\Gamma \cap \delta(T_j)| = 2$ for $j = 1, \dots, t$, then $|\Gamma \cap \delta(H)| \geq t$ and by parity one must have $|\Gamma \cap \delta(H)| \geq t+1$ and the result follows for those cycles. Else assume $|\Gamma \cap \delta(T_{j^*})| \geq 4$, for some j^* , let us assume for simplicity $j^* = t$. By induction hypothesis we have that: $|\Gamma \cap \delta(H)| + \sum_{j=1}^{t-2} |\Gamma \cap \delta(T_j)| \geq 3t - 6 + 1$, adding $|\Gamma \cap \delta(T_t)| \geq 4$ and $|\Gamma \cap \delta(T_{t-1})| \geq 2$ we prove again that the inequality is valid. ■

This second proof shows why the teeth and the handle play a different role. The teeth, in some sense, “force” a certain number of edges to be present in the coboundary of the handle.

The last proof is given because it may be adapted to prove validity of most of the inequalities of the coming sections. It is similar to a proof that can also be found in Applegate et al. [29].

Proof: (3) A non-algebraic and non-inductive proof of validity of comb inequalities. Let Γ be any Hamiltonian cycle, and x^Γ its associated characteristic vector. Inequality (19) is satisfied by x^Γ if all teeth are tight for Γ using exactly the same argument as in the previous proof. It is also trivially satisfied whenever $x^\Gamma(\delta(H)) \geq t+1$. In the other cases, i.e. $x^\Gamma(\delta(H)) < t+1$, define η by $x^\Gamma(\delta(H)) = t+1 - 2\eta$. Note that $\eta > 0$ is integer because t is odd, and $x^\Gamma(\delta(H))$ even. As $(t+1)/2$ represents the minimum number of paths traversing H when all teeth are tight, η represents the reduction in number of paths traversing H with respect to the tight teeth case. We prove basically, and the comb inequality tells us essentially, that the number of non tight teeth has to be at least as large as η .

First, we consider each tooth T_j separately, and define $\theta_j = 1$ if $x^\Gamma(T_j \cap H : T_j \setminus H) = 0$, $\theta_j = 0$ otherwise. It is readily seen that $x^\Gamma(\delta(T_j))/2 \geq 1 + \theta_j$ because $x^\Gamma(\delta(T_j))/2$ is the number of paths traversing T_j in the tour Γ , and the tooth T_j is traversed by at least two paths (i.e. is not tight) when $\theta_j = 1$. Thus, $x^\Gamma(\delta(T_j)) - 2 \geq 2\theta_j$.

Next, we consider the handle for which it is easy to check that $x^\Gamma(\delta(H)) \geq \sum_{j=1}^t x^\Gamma(T_j \cap H : T_j \setminus H) \geq \sum_{j=1}^t (1 - \theta_j) = t - \sum_{j=1}^t \theta_j$. Together with $x^\Gamma(\delta(H)) = t + 1 - 2\eta$, this gives $\sum_{j=1}^t \theta_j \geq 2\eta - 1 \geq \eta$ where the last inequality holds because η is a positive integer.

Hence, we have shown that the reduction η in the number of paths traversing H is at most $\sum_{j=1}^t \theta_j$, which is at most, by definition of θ_j , the number of non-tight teeth. This suffices to prove validity because $x^\Gamma(\delta(H)) = t + 1 - 2\eta \geq t + 1 - 2 \sum_{j=1}^t \theta_j \geq t + 1 - \sum_{j=1}^t [x^\Gamma(\delta(T_j)) - 2]$. ■

Theorem 41 (Grötschel and Padberg [403], [404]) *The comb inequalities are facet inducing for STSPP(n) and therefore for GTSP(n) ($n \geq 6$).*

Proof: The original proof is long and technical. It can be rewritten by proving that the simple comb inequalities are facet inducing for GTSP(n) (very easy), then for STSPP(n) by using Lemma 29, and finally using the node lifting theorems of [618] to obtain the result for general comb inequalities. ■

Comb inequalities together with subtour elimination inequalities and non negativity inequalities, completely describe STSPP(6), but not STSPP(7) (see end of next section). Comb inequalities, as we will see, are central in solving large TSP instances to optimality.

We now turn to more classes of inequalities. For most of them, comb inequalities can be seen as a subfamily. For the others, the comb inequalities can be seen as building blocks.

6. The Star and Path inequalities

Star inequalities (Fleischman [311]) are defined by two sets of subsets of V , $\mathcal{K} = \{H_1, \dots, H_h\}$, called *handles* and $\mathcal{T} = \{T_1, \dots, T_t\}$, called *teeth*, with $t \geq 3$ and odd, together with non-zero integers $\alpha_1, \dots, \alpha_h$ associated with the handles, and non-zero integers β_1, \dots, β_t associated with the teeth, such that:

$$H_1 \subset H_2 \subset \dots \subset H_i \subset \dots \subset H_h \tag{20}$$

$$T_i \cap T_j = \emptyset \quad \text{for } 1 \leq i < j \leq t \tag{21}$$

$$H_1 \cap T_j \neq \emptyset \quad \text{for } j = 1, \dots, t \tag{22}$$

$$T_j \setminus H_h \neq \emptyset \quad \text{for } j = 1, \dots, t \tag{23}$$

$$(H_{i+1} \setminus H_i) \setminus \bigcup_{j=1}^t T_j = \emptyset \quad \forall i, 1 \leq i \leq h - 1 \tag{24}$$

And a condition that relates the integers β_j to the integers α_i , which we call the *Interval Property*.

To define that property we will need a few definitions but we give first the corresponding Star inequality:

$$\sum_{i=1}^h \alpha_i x(\delta(H_i)) + \sum_{j=1}^t \beta_j x(\delta(T_j)) \geq (t + 1) \sum_{i=1}^h \alpha_i + 2 \sum_{j=1}^t \beta_j \quad (25)$$

Given a tooth T_j , a maximal index set of (successive) handles which have the same intersection with T_j is called *interval* relative to T_j . We call $\sum_{i=\ell}^{\ell+r} \alpha_i$ the *weight* of the interval $I = \{\ell, \ell + 1, \dots, \ell + r\}$.

Definition 42 *The Interval Property:* for each tooth T_j , we have β_j at least equal to the maximum weight of an interval relative to T_j

Figure 2.9 shows a tooth and the traces of the different handles on it. In the tooth, where a node is shown, there is at least one node, where none is shown there are no nodes. The handles, as suggested by the drawing, are numbered from top to bottom. There are 4 intervals, $\{1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}$, of weight 3, 4, 5 and 4, and therefore the β coefficient of that tooth must be at least 5.

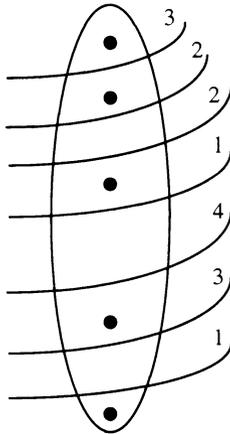


Figure 2.9. Example of intervals

Path inequalities (see [219]) are the special case where all the intervals relative to a same tooth have the same weight and the coefficient of that tooth is exactly that value. Figure 2.10 gives an example of a Path inequality. The coefficients associated with the handles and teeth are beside each set.

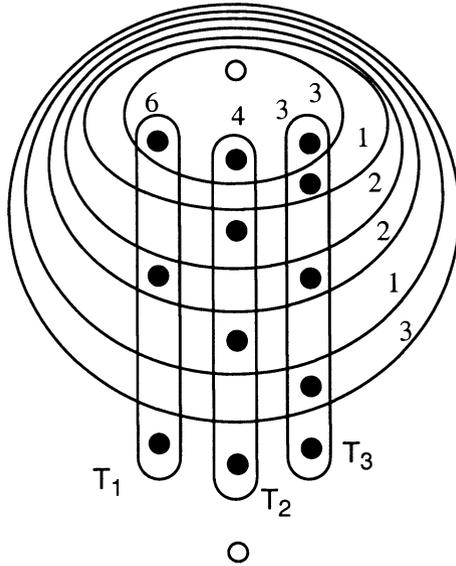


Figure 2.10. Example of a path ($h = 6, t = 3$)

In [219] a distinction is made between the case in which there is at least one vertex in the position of each of the two white filled points (path inequalities), only in one of these positions (wheelbarrow inequalities) or in none of these two positions (bicycle inequalities). We will here represent all three case under the same name: *path inequality*. Comb inequalities are exactly the path inequalities with a single handle.

The rationale behind the Interval Property is far from being obvious. We will try to explain it.

The RHS of Inequality (25) can easily be understood by the following argument. Consider Hamiltonian cycles for which all teeth are tight. Each handle must then have at least $t + 1$ edges of those cycles in its coboundary because t is odd (see comb inequalities), which leads to the first part of the RHS. The second part is due to the coboundaries of the teeth. This shows that for all such tours the inequality is valid. As one may suspect, the Interval Property is designed to make it valid for all the other tours. This property can be understood by looking at the tours Γ that are tight on all teeth except T_j and $|\Gamma \cap \delta(T_{j^*})| = 4$. Let $\{r, \dots, s - 1\}$ be an interval of T_{j^*} . Assume Γ uses edges of the coboundaries of the handles H_r to H_{s-1} only inside the teeth T_i for $i \neq j^*$. In other words, $x(T_{j^*} \cap H_i : T_{j^*} \setminus H_i) = 0$ for all $i \in \{r, \dots, s - 1\}$ (see Figure 2.11 where $j^* = 2$ from Figure 2.10). Then the coboundaries of the handles H_r to H_{s-1} may contain only $t - 1$ edges, the coboundaries

of the other handles only $t + 1$. Therefore a loss to the LHS, relatively to the former case, of $2 \sum_{i=r}^{s-1} \alpha_i$ is compensated by an increase of $2\beta_{j^*}$ due to the tooth T_{j^*} having now four edges in its coboundary. Therefore, to have validity of the inequality one must have $\beta_{j^*} \geq \sum_{i=r}^{s-1} \alpha_i$. This argument is valid for all intervals of T_{j^*} and therefore β_{j^*} must be greater or equal to the largest weight of an interval.

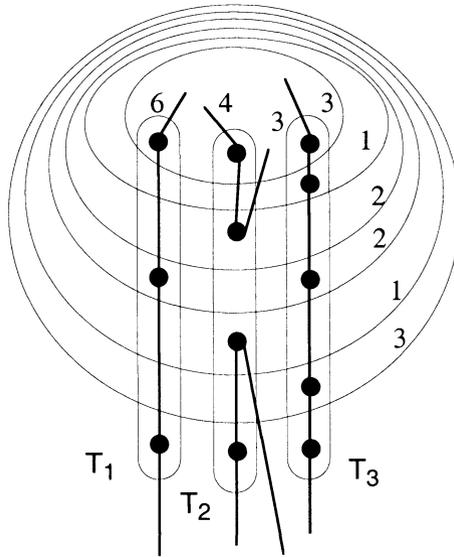


Figure 2.11. Example of minimal trace with T_2 not tight ($r = 3, s - 1 = 4$)

Of course, all this does not constitute a proof of validity. One can adapt the third proof of validity of combs to obtain one for the stars (see [615]). See [311] for a proof of:

Theorem 43 *Star inequalities are valid for $GTSP(n)$ and therefore for $STSP(n)$.*

Most star inequalities are not facet inducing even for $GTSP(n)$, but we have:

Theorem 44 *Path inequalities are facet inducing for $GTSP(n)$.*

Proof: See [219] ■

In Naddef and Rinaldi [619], and unpublished work of Queyranne and Wang the corresponding theorem for $STSP(n)$ can be found.

In the theoretical study of Goemans [383] on the comparison of valid inequalities for the TSP, the path inequalities turn out to be, at least in theory, the most powerful known inequalities in term of potential to

reduce the gap between the Held and Karp bound and the optimal value. We will see that they do turn out to be also useful in practice.

We end this section with an example of a facet inducing star inequality for $STSP(n)$ which is not a path inequality. It is shown in Figure 2.12.

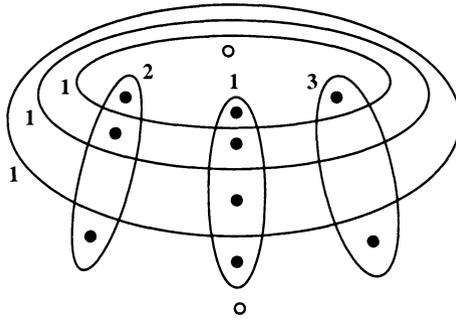


Figure 2.12. A facet inducing star (non-path) inequality; RHS=24

Research Question 45 Find a necessary and sufficient condition for a star inequality to be facet inducing for $GTSP(n)$.

Chances are that this necessary and sufficient condition will also hold for $STSP(n)$. It is easy to find necessary conditions. One is that the interval property must be satisfied with equality, else the inequality is the sum of another star inequality and of a subtour elimination inequality. Another necessary condition is that one cannot decompose the coefficients α_i and β_j in such a way that the star inequality is the sum of two other star inequalities.

To give an idea of the difficulty of the question, the first star inequality of Figure 2.13 is not facet inducing for $GTSP(n)$, $n \geq 8$, while the second is for $n \geq 10$. These two inequalities look pretty much like that of Figure 2.12. The RHS are 26 and 22, respectively.

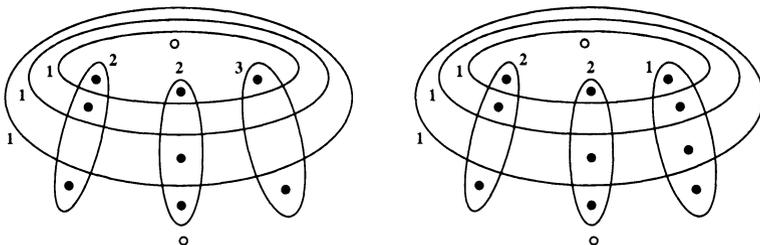


Figure 2.13. Examples of non facet and facet inducing star inequalities

For $n = 7$ path inequalities together with comb inequalities, subtour elimination inequalities and non-negativity constraints define entirely $STSP(7)$. For $n = 8$, to obtain such a description, one must add the crown inequalities ([617]) and four classes known as NEW1, NEW2, NEW3 and NEW4 that can be found in [473].

7. The Clique Tree and Bipartition inequalities

Clique tree inequalities were defined by Grötschel and Pulleyblank [406]. A *Clique tree* is defined by two sets of subsets of V , $\mathcal{K} = \{H_1, \dots, H_h\}$ and $\mathcal{T} = \{T_1, \dots, T_t\}$, such that:

$$H_i \cap H_j = \emptyset \quad \text{for } 1 \leq i < j \leq h \quad (26)$$

$$T_i \cap T_j = \emptyset \quad \text{for } 1 \leq i < j \leq t \quad (27)$$

$$T_j \setminus \bigcup_{i=1}^h H_i \neq \emptyset \quad \text{for } 1 \leq j \leq t \quad (28)$$

$$t_j = |\{i : T_j \cap H_i \neq \emptyset\}| \geq 1 \quad \text{for } 1 \leq j \leq t \quad (29)$$

$$h_i = |\{j : H_i \cap T_j \neq \emptyset\}| \geq 3 \text{ and odd} \quad \text{for } 1 \leq i \leq h \quad (30)$$

$$\text{If } T_j \cap H_i \neq \emptyset, \text{ it is a disconnecting} \quad (31)$$

set of the hypergraph $(V, \mathcal{K} \cup \mathcal{T})$.

In other words, handles are pairwise disjoint, and so are the teeth. No tooth is contained in a handle, every tooth has at least a vertex not in any handle and every handle intersects an odd number (≥ 3) of teeth. The *tree-like* structure is given by Condition (31). Further, in this section, we will drop this last condition together with Condition (28) to obtain a larger family of inequalities, the *bipartition inequalities* of Boyd and Cunningham [135]. When studying these inequalities, we will see the role of Condition (28) in the definition of a clique tree.

Figure 2.14 gives an example of a clique tree. Handles are in bold lines. If there are no vertices in position “Z”, then the clique tree spans the whole graph.

The corresponding *clique tree inequality* is:

$$\sum_{i=1}^h x(\delta(H_i)) + \sum_{j=1}^t x(\delta(T_j)) \geq \sum_{i=1}^h (h_i + 1) + 2t = 2h + 3t - 1 \quad (32)$$

in which h_i is defined by Condition (30).

As for all the inequalities that we will propose in this chapter, the RHS can easily be figured out by looking at tours for which all teeth are tight. For these tours, the handles must have at least $h_i + 1$ edges in their coboundaries, and this leads naturally to the RHS. To prove validity, one must prove that every time we augment the number of

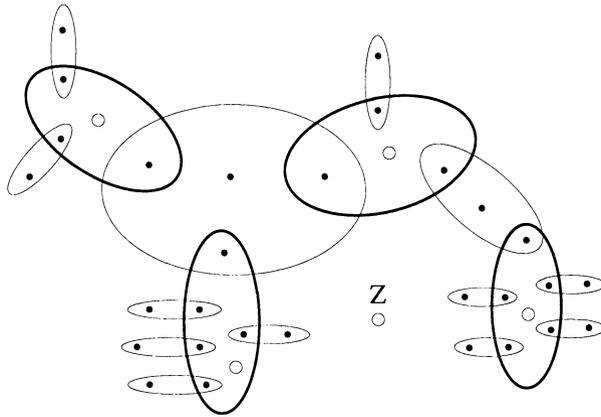


Figure 2.14. Example of a Clique Tree

edges in the coboundary of some teeth, we do not “save” more on the total number of edges in the coboundary of the handles. This is done in proofs of Applegate et al. [29] and in Naddef and Pochet [615]. An algebraic proof of validity is given in [406]. Moreover one can find there the following theorem:

Theorem 46 (Grötschel-Pulleyblank [406]) *Clique tree inequalities are facet inducing for STSPP(n).*

A clique tree with a single handle is a comb. Some authors admit $\mathcal{K} = \emptyset$ (which, by the way, does not satisfy Condition (29)), if so, subtour elimination inequalities are also clique tree inequalities. In [616], Naddef and Rinaldi show that the clique trees, which have at least a vertex in position “Z” of Figure 2.14, can be obtained by repeated 2-sums starting with two combs.

We now turn to the *bipartition inequalities* of Boyd and Cunningham [135]. A *bipartition* is defined just like a clique tree but without Conditions (28) and (31). There are coefficients $\beta_1, \dots, \beta_j, \dots, \beta_t$ associated with the teeth. Let D be the set of indices of teeth that contain no vertex outside any handle, i.e.

$$T_j \setminus \bigcup_{i=1}^n H_i = \emptyset \text{ if and only if } j \in D \tag{33}$$

A tooth in the set $\{T_j : j \in D\}$ is called *degenerate* by Boyd and Cunningham. We will define, later on, a more restrictive notion of degeneracy which will not consider as degenerate those teeth with index in D which intersect only two handles. The intuitive explanation of the co-

efficients, further down, will explain why we will not qualify these teeth as degenerate.

Let t_j be as defined in Condition (29), that is the number of handles that tooth T_j intersects, then the coefficients β_j are defined by:

$$\beta_j = 1 \quad \text{if } j \notin D \tag{34}$$

$$\beta_j = \frac{t_j}{t_j - 1} \quad \text{if } j \in D \tag{35}$$

The corresponding *bipartition inequality* is:

$$\sum_{i=1}^h x(\delta(H_i)) + \sum_{j=1}^t \beta_j x(\delta(T_j)) \geq \sum_{i=1}^h (h_i + 1) + 2 \sum_{j=1}^t \beta_j \tag{36}$$

In Figure 2.15, in which handles are in bold lines, the two teeth T^* and T are degenerate. The first one intersects three handles, therefore its coefficient is $3/2$, the second only two and therefore its coefficient is 2.

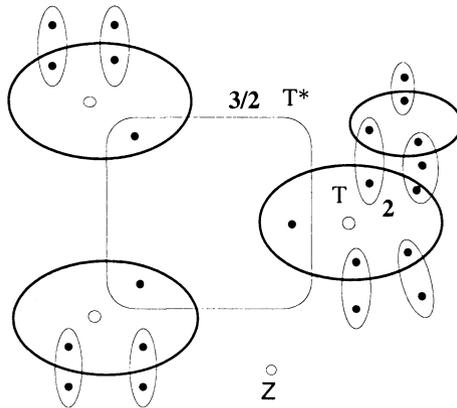


Figure 2.15. Example of a Bipartition

The right hand side is again understood by looking at the tours for which all teeth are tight. An example of the trace of such a tour on the various coboundaries, involved in the inequality, is given in the first drawing of Figure 2.16, which minimizes the total number of edges in these coboundaries. In such a tour each handle must have at least $h_i + 1$ edges, where h_i is the number of teeth that handle H_i intersects.

Assume that we look at tours which intersect tooth T in four edges. The second drawing of Figure 2.16 shows the trace of such a tour. Note that the increase of two edges in the coboundary of T is compensated for by a decrease of two edges in each of the coboundaries of the two only

handles it intersects. Enabling tours to use more edges in the coboundary of T does not yield any further gain in the minimum number of edges of the coboundaries of the handles. This explains the coefficient of 2, in order to maintain validity of the inequality.

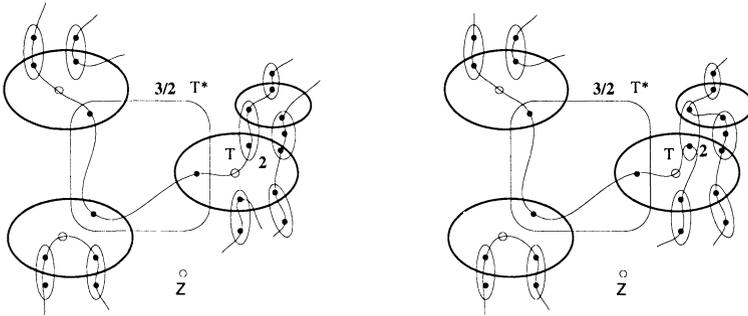


Figure 2.16. Traces of tight tours with 2 and 4 edges in $\delta(T)$

Now we turn to tooth T^* . The first drawing of Figure 2.17 shows again a minimal trace on the handles of a tour tight for all teeth. In the second, the trace corresponds to tours that intersect the coboundary of tooth T^* in four edges. We also gain two edges on the coboundaries of the handles. If we go one step further, and enable six edges in the coboundary of tooth T^* , that is four more than the minimum, this time we gain *four* edges in the coboundaries of the handles from the previous tour and *six* from the first one. If the tours, the traces of which are shown in the first and third drawing of Figure 2.17, are to be tight for the inequality, then the loss of 4 edges on the coboundary of T^* cannot be more than compensated by the gain of 6 edges on the coboundaries of the handles, therefore $\beta_{T^*} \geq 6/4 = 3/2$. Since enabling yet more edges in the coboundary of T^* does not decrease further the minimum number of edges in the coboundaries of the handles, if $\beta_{T^*} > 3/2$, then the only tight tours would be those which are tight on T^* , and therefore the inequality would be contained in the facet $x(\delta(T^*)) \geq 2$. This explains the coefficient $t_j/(t_j - 1)$ of some teeth.

In Section 9, the technique that we just used in order to define the coefficients of T and T^* will be developed into a general procedure. In some sense, the coefficient of teeth should *normally* be defined by the case of four edges in their coboundary, so we will call degenerate only those teeth for which the coefficient is defined by tours having six or more edges in their coboundaries. Therefore tooth T of our example will no longer be considered as degenerate. We make this more precise in Section 9.

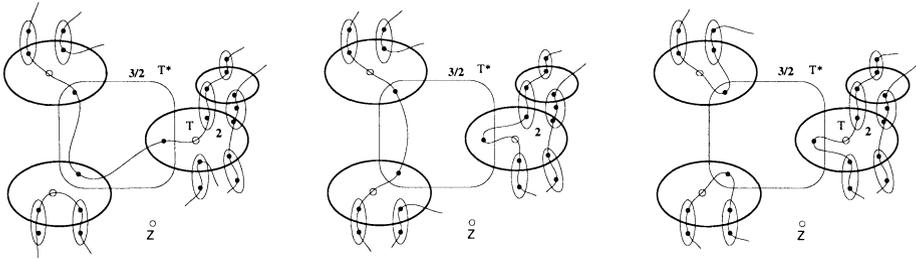


Figure 2.17. Traces of tight tours with 2, 4 and 6 edges in $\delta(T^*)$

Of course, since all we use is that a cycle intersects a coboundary in an even number of edges, so do closed walks, in all the preceding material, tours can be replaced by closed walks. In no case have we given a proof of validity of the bipartition inequality. This is done by:

Theorem 47 (Boyd-Cunningham [135]) *Bipartition inequalities are valid for STSPP(n).*

In [232] W. Cunningham and Y. Wang give two necessary conditions for such an inequality to be facet defining. The first is that there be no subset of degenerate teeth which is disconnecting for the underlying hypergraph (as a consequence, the bipartition of our example is not facet inducing). The second is that for each edge there exists a tight tour that contains it, which as already mentioned is necessary for any non-trivial inequality to be facet inducing. The next section gives an example of a bipartition which does not yield a facet inducing inequality, but for which there is a way around, via a correcting term, to get a facet inducing inequality. Cunningham and Wang conjecture that together these two conditions are sufficient.

8. The Ladder inequalities

A *ladder inequality* is defined by two handles H_1 and H_2 an even number $t \geq 4$ of teeth $\{T_1, \dots, T_j, \dots, T_t\}$, with integer coefficients $\beta_1, \dots, \beta_j, \dots, \beta_t$ associated with them, such that (see Figure 2.18 in

which the coefficients β are written next to the corresponding teeth):

$$H_1 \cap H_2 = \emptyset \tag{37}$$

$$T_j \cap T_k = \emptyset \quad 1 \leq j < k \leq t \tag{38}$$

$$H_1 \cap T_1 \neq \emptyset, H_2 \cap T_1 = \emptyset \tag{39}$$

$$H_2 \cap T_2 \neq \emptyset, H_1 \cap T_2 = \emptyset \tag{40}$$

$$T_1 \setminus H_1 \neq \emptyset, T_2 \setminus H_2 \neq \emptyset \tag{41}$$

$$T_j \cap H_i \neq \emptyset \quad \text{for } i = 1, 2 \text{ and } j \geq 3 \tag{42}$$

$$\text{if } T_j \setminus (H_1 \cup H_2) \neq \emptyset, \text{ then } \beta_j = 1, \quad \text{else } \beta_j = 2; \text{ for } j \geq 3 \tag{43}$$

$$\beta_1 = \beta_2 = 1 \tag{44}$$

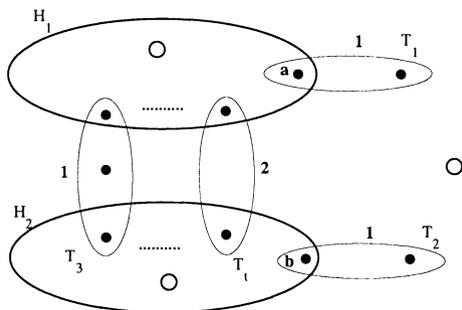


Figure 2.18. Example of a ladder

Teeth T_1 and T_2 play a special role. The first one only intersects H_1 and the second only H_2 , all other teeth intersect both handles. The ladder inequality is:

$$\sum_{i=1}^2 x(\delta(H_i)) + \sum_{j=1}^t \beta_j x(\delta(T_j)) - 2x(H_1 \cap T_1 : H_2 \cap T_2) \geq 2t + 2 \sum_{j=1}^t \beta_j \tag{45}$$

The coefficients on the teeth can be explained just as in the case of the bipartition inequalities. Note the special correcting (or lifting) term $-2x(H_1 \cap T_1 : H_2 \cap T_2)$ on the left hand side of Inequality (45) without which this would only be a particular case of bipartition inequalities.

Again we try to give the reader some insight into the validity of this inequality. Observe first that no tour that is tight for the corresponding bipartition (not ladder) inequality contains an edge of $(H_1 \cap T_1 : H_2 \cap T_2)$ (we leave it to the reader to convince himself/herself of this fact or consult [615]), and therefore the corresponding face is contained in the intersection of the facets $x_e \geq 0$ for $e \in (H_1 \cap T_1 : H_2 \cap T_2)$. Figure 2.19

gives the trace of a tight tour for the ladder inequality, that contains such an edge and with all teeth tight. Extension of this tour to the case $t \geq 6$ is straightforward.

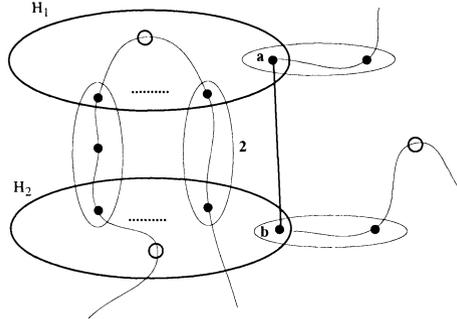


Figure 2.19. Trace of a tight tour containing $e \in (H_1 \cap T_1 : H_2 \cap T_2)$

Theorem 48 (Boyd, Cunningham, Queyranne and Wang [136]) *The ladder inequalities are facet defining for STSPP(n).*

9. A general approach to some TSP valid inequalities

Let $\mathcal{S} = \{S_1, \dots, S_i, \dots, S_p\}$ be a set of distinct (possibly overlapping) subsets of V . Let $\alpha_1, \dots, \alpha_i, \dots, \alpha_p$ be strictly positive integers. An inequality is said to be in *closed-set form* if it can be written as:

$$\sum_{i=1}^p \alpha_i x(\delta(S_i)) \geq r(\mathcal{S}) \tag{46}$$

where $r(\mathcal{S})$ is the minimum of the LHS on all tours.

Path, star, bipartition inequalities are inequalities in closed-set form, and therefore so are the comb and clique tree inequalities. Inequalities containing path, star and clique trees, and named *binested inequalities* in [613], are also inequalities in closed-set form. Ladder inequalities are not in closed-set form because of the correcting term on the LHS.

In most of the known closed-set form inequalities for $STSPP(n)$, the set \mathcal{S} of subsets of V is partitioned into two sets \mathcal{K} and \mathcal{T} where $\mathcal{K} = \{H_1, \dots, H_h\}$ is the set of so called *handles*, $\mathcal{T} = \{T_1, \dots, T_t\}$ is the set of so called *teeth*.

We also have non-zero integers $\alpha_1, \dots, \alpha_h$, associated with the handles, and (not necessarily integer) rationals β_1, \dots, β_t , associated with the teeth.

These inequalities in closed-set form read:

$$\sum_{i=1}^h \alpha_i x(\delta(H_i)) + \sum_{j=1}^t \beta_j x(\delta(T_j)) \geq A + 2 \sum_{j=1}^t \beta_j \quad (47)$$

or equivalently

$$\sum_{i=1}^h \alpha_i x(\delta(H_i)) \geq A - \sum_{j=1}^t \beta_j (x(\delta(T_j)) - 2). \quad (48)$$

Expression (48) suggests the following interpretation, which can be used in order to define a general procedure for computing the coefficients A and β_j , $j = 1, \dots, t$, of the valid inequality.

When all teeth are tight for a given Hamiltonian cycle Γ that is $x^\Gamma(\delta(T_j)) = 2$ for all $j = 1, \dots, t$, Inequality (48) then reduces to $\sum_{i=1}^h \alpha_i x(\delta(H_i)) \geq A$. Hence, for validity, the largest possible value of A is the minimum value of the left hand side of (48) over all Hamiltonian cycles Γ with all teeth tight. Formally,

$$A = \min_{\Gamma} \left\{ \sum_{i=1}^h \alpha_i |\Gamma \cap H_i| : |\Gamma \cap \delta(T_j)| = 2 \text{ for all } j = 1, \dots, t \right\} \quad (49)$$

and does not depend on the coefficients β_j .

Now we describe a general procedure to compute the coefficients β_j in the valid Inequality (48) when the coefficients of the handles are assumed to be fixed. First we order the teeth. We assume without loss of generality that the teeth have been renumbered in such a way that their order is $T_1, \dots, T_j, \dots, T_t$. Next, we compute the coefficients β_j from $j = 1$ to $j = t$ using a sequential lifting procedure. The lifted variables are the non-negative integer slack variables s_j associated to the subtour elimination inequalities defined on the teeth, namely $s_j = x(\delta(T_j)) - 2$ for $j = 1, \dots, t$. Initially, to compute the coefficient A in Expression (49), all slacks s_j are projected to zero and we obtain the inequality $\sum_{i=1}^h \alpha_i x(\delta(H_i)) \geq A$ which is valid for the subspace where $s_j = 0$ for $j = 1, \dots, t$, that is when all teeth are tight. Then, sequentially, from $j = 1$ to $j = t$, all slacks s_j are lifted.

For $k \in \{1, \dots, t\}$, assuming coefficients $\beta_1, \dots, \beta_{k-1}$ have been already computed, the lifting coefficient β_k is taken as the minimum value β for which the inequality

$$\sum_{i=1}^h \alpha_i x^\Gamma(\delta(H_i)) + \sum_{j=1}^{k-1} \beta_j [x^\Gamma(\delta(T_j)) - 2] + \beta [x^\Gamma(\delta(T_k)) - 2] \geq A \quad (50)$$

is satisfied by all Hamiltonian cycles Γ with $s_j = x^\Gamma(\delta(T_j)) - 2 = 0$, for all $j > k$.

We let \mathcal{B}_ℓ^k denote the set of Hamiltonian cycles for which all the teeth T_{k+1}, \dots, T_t are tight and intersect the coboundary of T_k in 2ℓ edges. Then the value of β_k is defined by

$$\beta_k = \max_{\ell > 1} \frac{b_k^1 - b_k^\ell}{2\ell - 2} \tag{51}$$

where

$$b_k^\ell = \min_{\Gamma \in \mathcal{B}_\ell^k} \left[\sum_{i=1}^h \alpha_i x^\Gamma(\delta(H_i)) + \sum_{j < k} \beta_j [x^\Gamma(\delta(T_j)) - 2] \right] \tag{52}$$

We call *degenerate* those teeth T_k for which the coefficient β_k cannot be obtained for $\ell = 2$ in Equation (51).

Note that by construction of β_k , we must have $b_k^1 = A$ for all $k \in \{1, \dots, t\}$. Also note that the β_k are not necessarily integer. The construction is summarized in the following theorem.

Theorem 49 *Let $\mathcal{K} = \{H_1, \dots, H_h\}$, let $\mathcal{T} = \{T_1, \dots, T_t\}$, be an ordered set of teeth, and let $\alpha_1, \dots, \alpha_h$ be nonzero integers associated with the handles. If A is defined by Expression (49), and, for $k \in \{1, \dots, t\}$, if β_k is the coefficient of tooth T_k determined by Expressions (52) and (51), then the following inequality is valid:*

$$\sum_{i=1}^h \alpha_i x(\delta(H_i)) \geq A - \sum_{j=1}^t \beta_j [x(\delta(T_j)) - 2] \tag{53}$$

Proof: Trivial by the way we constructed the coefficients of the teeth.
■

Remark 50 *The above procedure provides a way of defining valid inequalities for the TSP. The difficulty in using Theorem 49 to prove validity of inequalities is the exponential growth in the number of cases to study when the number of teeth increases.*

Remark 51 *In all the inequalities seen so far in this chapter, the coefficients of the teeth are order independent, that is they can be computed as if they are the first in the sequential lifting. This is not always the case. We give now an example.*

Figure 2.20, which has been provided by M. Queyranne, illustrates this. Here, the lifting coefficients are defined as in Theorem 49, but we

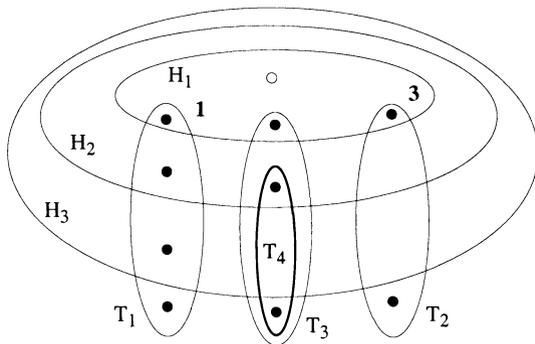


Figure 2.20. Example of a facet defining inequality

will see that their values are sequence dependent, and in fact only one order will lead to an inequality. Handles have all coefficients of 1.

The support in Figure 2.20 would give an inequality of the form

$$\sum_{i=1}^3 x(\delta(H_i)) \geq 12 - \sum_{j=1}^4 \beta_j [x(\delta(T_j)) - 2]. \tag{54}$$

If you authorize a Hamiltonian cycle to intersect the coboundary of T_4 in 4 or more edges restricting all the other teeth to be tight, we cannot compensate by “saving” anything on the coboundaries of the handles. Therefore, this procedure will always produce an inequality which is just the addition of the inequality obtained without tooth T_4 and the subtour elimination inequality associated to that tooth. In other words, lifting T_4 first gives $\beta_4 = 0$.

The coefficients of T_1 and T_2 can be defined in the manner used so far. This gives $\beta_1 = 1$ and $\beta_2 = 3$. Let us authorize a Hamiltonian cycle to intersect the coboundary of T_3 in 4 edges, imposing T_4 to be tight. We can only save 2 edges on the coboundaries of the handles (whether or not we impose T_1 and T_2 to be tight); see Figure 2.21(a). These 2 edges we save are 2 of the 4 that crossed the border of H_1 when we imposed all teeth to be tight. This yields a coefficient of $\beta_3 = 1$ for T_3 , instead of a coefficient of 2 in the star inequality made up of the same handles and teeth T_1, T_2 and T_3 . Now that the coefficients of T_1, T_2 and T_3 are known we can define the coefficient of T_4 .

The trace of a tour shown in Figure 2.21(b), shows that we can save 4 edges on the coboundaries of the handles by enabling a cycle to intersect the coboundary of T_4 in 4 edges. Two of these 4 edges are compensated for, by the extra 2 edges in the coboundary of T_3 , the 2 others by the

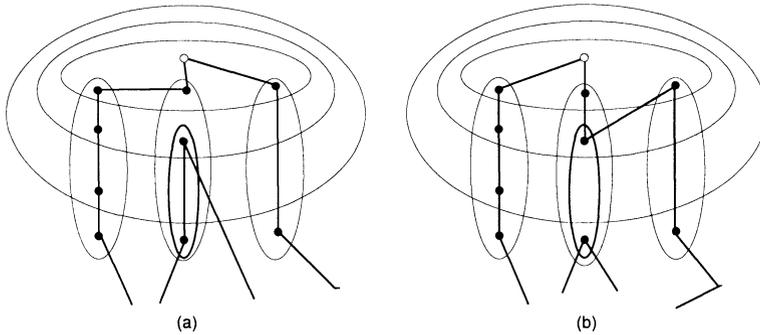


Figure 2.21. Some traces of Hamiltonian cycles

extra 2 edges in the coboundary of T_4 , which yields a coefficient of $\beta_4 = 1$ for tooth T_4 . This inequality is facet defining for $STSP(n)$, $n \geq 9$.

We use the developments of this section to describe a family of inequalities that contains the inequalities described in the previous sections.

10. A unifying family of inequalities

This section may be skipped by most readers. It would be too long to expose here the binested inequalities (see [613]). They contain the star (and therefore path), the clique tree, and the bipartition inequalities without degenerate teeth (in the sense of the previous section). They do not contain the ladder inequalities. One way of making them contain all remaining bipartition inequalities and the ladder inequalities would be as follows:

- 1 Recompute the coefficient of the teeth using the method of the previous section,
- 2 Arbitrarily order all edges which do not appear in a tight tour with the newly computed coefficients and compute sequentially the coefficient of the correcting term corresponding to them. The correcting coefficients will be subtracted to the LHS just like in the ladder inequalities.

We conjecture that the first calculation (the one for the teeth) is order independent. The second one is not. One can find an example in [615] in relation with generalizations of the ladder inequalities, for which this lifting is not order independent.

11. Domino inequalities

In the previous sections, teeth in some sense forced edges to cross the border of the handles or had to be no longer tight. This was obtained either because of parity or by the situation of nodes in key positions. There is another way of forcing edges in the border of the handles, that is by penalizing some edges, which must be taken in some circumstances if one does not cross the border of the handle. This is the rationale behind the *Domino Parity* inequalities defined by Letchford [559], although this is not explicit in his exposition. We describe here a large subclass of these inequalities which we will call *Domino* inequalities.

Let $H \subset V$, referred to as the *handle*, $\mathcal{T} = \{T_1, \dots, T_i, \dots, T_t\}$, $t \geq 3$ and odd, where $T_i = A_i \cup B_i \subset V$, referred to as the set of *teeth*, such that:

$$A_i \cap B_i = \emptyset \quad \text{for } i = 1, \dots, t \quad (55)$$

$$A_i \cup B_i \neq V \quad \text{for } i = 1, \dots, t \quad (56)$$

$$(A_i : B_i) \cap (A_j : B_j) = \emptyset \quad \text{for } 1 \leq i < j \leq t \quad (57)$$

We say that A_i and B_i are the *two half dominoes* of the *domino* (or *tooth*) T_i . Note that there are no further restrictions on the elements of \mathcal{T} , i.e. they can for example intersect.

The main difference with the Domino Parity configurations of Letchford is that we added the non crossing Condition 57. Naddef proves in [614] that the only facet inducing Domino Parity configurations that do not satisfy this condition have two teeth, say T_1 and T_2 such that $T_1 \cup T_2 = V$ with $T_1 \cap T_2 \neq \emptyset$, but in that case there exists a domino inequality satisfying condition 57, which includes the same facet. Let $C = (\bigcup_{i=1}^t (A_i : B_i)) \setminus \delta(H)$ be the set of edges inside the teeth and crossing from one half domino to the other without crossing the boundary of the handle. The *domino inequality* is defined by:

$$x(\delta(H)) + \sum_{j=1}^t x(\delta(T_j)) + 2x(C) \geq 3t + 1 \quad (58)$$

Theorem 52 *Domino inequalities are valid for STSP(n).*

Proof: Call *LHS* the left hand sides of these inequalities. it is easy to check that:

$$LHS \geq \frac{1}{2} \sum_{i=1}^t x(\delta(A_i)) + \frac{1}{2} \sum_{i=1}^t x(\delta(B_i)) + \frac{1}{2} \sum_{i=1}^t x(\delta(T_i)) = 3t \quad (59)$$

Since LHS is even for all tours, we can raise $3t$ to $3t + 1$. ■

Note the similarity with the algebraic proof of validity of comb inequalities. To understand the role of the partition $\{A_i, B_i\}$ of each tooth, consider the example of Figure 2.22, where, except for T_7 , $A_i = T_i \cap H$ and $B_i = T_i \setminus H$. For T_7 , we have $A_7 \supseteq T_3 \cup T_4$, $B_7 \supseteq T_5 \cup T_6$. All teeth T_1 to T_6 are pairwise node disjoint. A tour for which all teeth are tight does not necessarily force more than six edges in the coboundary of H as seen in Figure 2.23(a), but then it necessarily uses an edge of $(A_i : B_i) \setminus \delta(H)$ which is *penalized* by two units. So either such a tour goes from A_i to B_i by using an edge of the coboundary of H as in Figure 2.23(b), or it uses a penalized edge to do so. In this example none of the other domino partitions play a role. The domino inequality corresponding to this example is facet inducing since it is a twisted comb inequality (definition below, see also [161]).

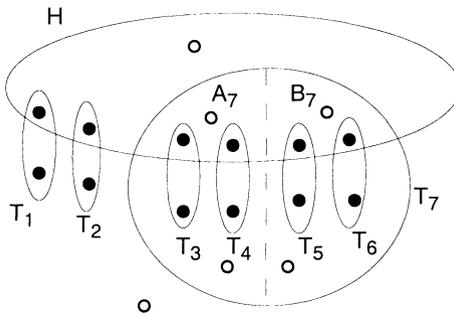


Figure 2.22. Example of domino inequality

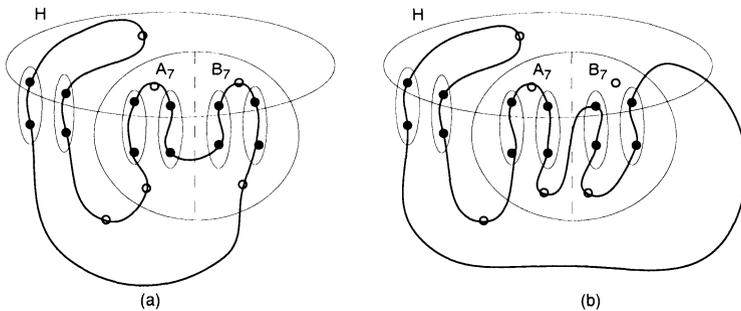


Figure 2.23. Tight tours for the example of Figure 2.22

We now turn to necessary conditions in order for a domino inequality to be facet inducing. A *minimal* domino configuration is one such that the number of non minimal (by inclusion) teeth is minimum, among all domino configurations that yield the same domino inequality. A comb

can be seen as a domino with only minimal teeth, but also as a domino configuration with one tooth containing all the others and such that one of its half dominoes contains all the teeth, the other half domino contains none, is non empty and has an empty intersection with the handle. Finally there is a node not in any tooth nor in the handle. Naddef in [614], proves the following propositions:

Proposition 53 *The teeth of a facet inducing minimal domino inequality are nested.*

Proposition 54 *If T_i is a minimal tooth in a facet defining domino inequality, then $\{A_i, B_i\} = \{T_i \cap H, T_i \setminus H\}$.*

Corollary 55 *Only the domino partition for non-minimal teeth has some interest. We will omit, from now on, giving the domino partition of minimal teeth.*

Minimal teeth are similar to comb teeth. For non minimal teeth, each tooth they contain must be contained in one of the half domino of that tooth. For minimal domino configurations it is shown in [614] that in a non minimal tooth each half domino must contain at least two *odd* teeth of immediate lower rank in the nesting. A tooth is odd if either it is minimal or it contains, including itself, an odd number of teeth.

Conjecture 56 *Minimal Domino inequalities with at least three odd maximal teeth and such that each half domino of a non minimal tooth contains at least two odd teeth, are facet inducing for STSPP(n).*

One can find in [614] the proof that a large subfamily of these inequalities is facet inducing for the graphical relaxation.

We end this section with a family of facet inducing domino inequalities. Boyd, Cockburn and Viella define in [134] the following *twisted combs*. A twisted comb inequality is a domino inequality satisfying all the previous necessary conditions and which has a unique maximal tooth which is not also minimal. Note that the example of Figure 2.22 refers to a twisted comb.

Theorem 57 *(Boyd, Cockburn, Vella [134]) The twisted comb inequalities are facet inducing for STSPP(n).*

For more on the domino inequalities, besides the papers already mentioned, one can also read Cockburn [204].

Research Question 58 *How can one generalize the domino partition for non minimal teeth to inequalities with more than one handle?*

A first answer to this question can be found in [614] where a generalization of path inequalities can be found.

12. Other inequalities

There are many other inequalities which do not fall in one of the classes defined so far. The best known example are the *hypohamiltonian inequalities* (see [195]). A graph is said to be *hypohamiltonian* if it is *not* Hamiltonian but the deletion of *any* vertex yields a Hamiltonian graph. A hypohamiltonian graph is said to be *edge maximal* if it is hypohamiltonian and the addition of *any* non existing edge yields a Hamiltonian graph. The Petersen graph is an edge maximal hypohamiltonian graph.

Let $G = (V, E^*)$ be an edge maximal hypohamiltonian graph and $K_n = (V, E)$ be the complete graph on the same vertex set. The *simple hypohamiltonian* inequality $fx \geq f_0$ is given by:

$$1 \quad f_e = 1 \text{ if } e \in E^*$$

$$2 \quad f_e = 2 \text{ if } e \in E \setminus E^*$$

$$3 \quad f_0 = |V| + 1$$

A general hypohamiltonian inequality is obtained by clique lifting of a simple hypohamiltonian inequality, that is replacing each vertex by a clique of some size, the coefficients of the inequality on the edges linking two nodes of the same clique is 0, the other coefficients are inherited from the ones of the simple inequality. In other words a hypohamiltonian inequality $fx \geq f_0$ is such that the partition of V into $V_1, \dots, V_i, \dots, V_p$, the vertex sets of the connected components of the graph $G_0 = (V, E_0)$ with $E_0 = \{e \in E : f_e = 0\}$ induces an edge maximal hypohamiltonian graph when one shrinks each V_i to a single node in $G_1 = (V, E_1)$, with $E_1 = \{e \in E : f_e = 1\}$.

Theorem 59 *Hypohamiltonian inequalities are facet inducing for GTSPP(n).*

Proof: Let $fx \geq f_0 = p + 1$ be a hypohamiltonian inequality. Let the sets $V_1, \dots, V_i, \dots, V_p$ be the the partition of V induced by the connected components of $G_0 = (V, E_0)$. Inequality $fx \geq f_0$ is trivially valid since any closed walk must either contain an edge with coefficient 2, or two edges of coefficient 1 in some $(V_i : V_j)$, $i \neq j$. Therefore the coefficients sum up to at least $p + 1$ on the closed walk. Assume $fx \geq f_0$ is not facet inducing, i.e. the face it defines is strictly contained in some facet. Let $hx \geq f_0$ (same RHS), be the corresponding linear inequality. Every closed walk $W \in \mathcal{W}_f^-$ is also in \mathcal{W}_h^- . Let $W \in \mathcal{W}_f^-$. For $e \in E(V_i)$ for some i , we have $W + 2k\{e\} \in \mathcal{W}_f^-$ for all integer k , and therefore also in \mathcal{W}_h^- , which proves that $h_e = 0$ for these edges. As mentioned earlier, necessarily $h_e = \gamma_{ij}$ for all $e \in (V_i : V_j)$. Let W_i be a Hamiltonian cycle

of $G \setminus V_i$ that contains only edges of f value 0 or 1. Such a cycle exists since picking a single vertex in each V_i and the edges of f -value 1 linking them yields an hypohamiltonian graph. Let $e \in \delta(V_i)$ be such that $f_e = 1$. We have that $W_i + 2\{e\} \in \mathcal{W}_f^-$, so also in \mathcal{W}_h^- . this proves that all these edges have the same value $h_e = \gamma_i$. Since a hypohamiltonian graph is necessarily connected, we have that for $h_e = \gamma$ for all $e \in E$ such that $f_e = 1$, and therefore $\gamma = 1$, since we chose the same RHS. Now for any edge $e \in (V_i : V_j)$ such that $f_e = 2$, there is a Hamiltonian cycle containing it and only edges of f -value 0 and 1, which proves that $h_e = 2$ for these edges, and we have proved that $h_e = f_e$ for all $e \in E$. ■

We have described most of the known facet inducing inequalities that can be found in the literature, with the exception of the *crown* inequalities of Naddef and Rinaldi [617]. Most of the other known inequalities come from clique lifting of the inequalities obtained from the complete description of polytopes for small instances by Christof and Reinelt and which are available at: www.informatik.uni-heidelberg.de/groups/comopt/software/SMAPO/tsp/tsp.html.

13. The separation problem

The separation problem for the STSP is the following: *Given a fractional solution x^* , find a valid inequality, preferably facet inducing for $STSP(n)$, say $fx \geq f_0$, such that $fx^* < f_0$ and such that $f_0 - fx^*$ is not too small.*

We may search for a violated inequality in two ways. Either by searching among known classes of inequalities, for example among those we described in the previous sections, or in a very general form. Applegate et al. in [31] refer to the first type of search as a template paradigm separation. This has been for a long time the only type of STSP separation framework (see Padberg and Rinaldi [645], [647] and [648], Grötschel [397], Padberg and Crowder [229]). More recently Christof, Reinelt and Wenger have tried to use the complete knowledge of the $STSP(n)$ for $n \leq 10$. We will give a brief description of their method in Section 15. Applegate, Bixby, Chvátal and Cook [31] uses a more elaborate method which we will sketch in Section 18.

We end this section with the few complexity results known so far. Separating subtour elimination constraints is polynomial because it amounts to finding a minimum cut in an undirected weighted graph— we already addressed this point in Section 2. There also exists a polynomial time separation algorithm for *2-matching* constraints i.e. simple comb inequalities (see Padberg and Rao [644]).

Another result deals with bipartition inequalities with a fixed number of handles and teeth. Carr [170] and [171] showed that there is a polynomial separation algorithm in this case. We give the nice and elegant argument in the particular case of a comb with t teeth.

- Choose $2t$ vertices and, among these, choose t vertices that will belong to the handle. Match each of these t vertices of the handle to one of the t others to form t disjoint pairs.
- Find a minimum cut that separates the t nodes chosen to be in the handle from the t others. This can be done in polynomial time.
- For each pair of matched vertices, find a minimum cut that separates the pair from the other $t - 2$ vertices. Choose one shore of the minimum cut to be a tooth.
- If the teeth intersect, there is an uncrossing argument that enables to obtain non intersecting teeth with the same coboundary values in x^* .
- Check for violation.

Since there is only a polynomial number of choices for the $2t$ vertices, and for each such choice a polynomial number of choices thereafter, the whole procedure is polynomial for any fixed t . Of course this does not lead to a practical separation routine, but some of the separation routines that we will describe can be seen as trying to guess clever choices of some of these $2t$ vertices.

Fleischer and Tardos [310], designed a polynomial algorithm to separate maximally violated comb inequalities as long as the support graph of x^* (the graph induced by the edges e with $x_e^* > 0$) is planar. A comb is said to be maximally violated if the difference between the RHS and the LHS is 1 ($.5$ if in the standard " \leq " form).

In case of planar support graphs, Letchford [559] gives a polynomial algorithm that separates combs without the maximal violation requirement. This same algorithm also separates the domino inequalities under the same conditions.

The following has never been implemented and is so far of only a theoretical interest. Caprara, Fischetti and Letchford [161] address the separation of maximally violated cuts in the general context of ILP's of the form $\min\{cx : Ax \leq b, x \text{ integer}\}$, where A is an $m \times n$ integer matrix and b an m -dimensional integer vector. For any given integer k they study *mod-k* cuts of the form $\lambda Ax \leq \lfloor \lambda b \rfloor$ for any $\lambda \in \{0, 1/k, \dots, (k-1)/k\}^m$ such that λA is integer. A *mod-k* cut is called maximally violated if it is violated by $(k-1)/k$ by the given fractional point x^* . It is

shown that, for any given k , separation of maximally violated mod- k cuts requires $O(mn \min\{m, n\})$ time. This result has applications to both the symmetric and asymmetric TSP. Indeed, for any given k , Caprara et al. propose an $O(|V|^2|E^*|)$ -time exact separation algorithm for mod- k cuts which are maximally violated by a given fractional (symmetric or asymmetric) TSP solution with support graph $G^* = (V, E^*)$. This implies that one can identify in $O(|V|^2|E^*|)$ time a maximally violated mod-2 cut for the symmetric TSP whenever a maximally violated comb inequality exists. The reader is referred to [161] for more details.

We now turn to the heuristic separation of specific classes of inequalities.

14. Greedy heuristic for minimum cut

Finding a minimum cut that contains a given set S_0 is polynomial. In some of the following separation heuristics this search has to be performed so many times that we will instead use the following *greedy* heuristic – also known as *max-back*.

Consider a graph $G = (V, E)$ and a weight function x^* on the edges such that $x^*(\delta(v)) = 2$ for all $v \in V$. Let $S \subset V$, we say that $v \notin S$ sees S by an amount of $b(v) = \sum_{u \in S} x_{uv}^*$. We call $b(v)$ the *max-back* value of v relative to S . A vertex not linked to S by any edge has a max-back value of zero. Note that this definition still makes sense if $v \in S$, and we will sometimes use it also in this case.

The max-back heuristic: Input $S_0 \subset V$ and x^* .

- $Cutmin = \sum_{e \in \delta(S_0)} x_e^*$, for all $v \notin S_0$, $b(v) = \text{max-back value of } v \text{ relative to } S_0$. Set $S = Smin = S_0$, $Cutval = Cutmin$
- While $S \neq V$: Choose $v \notin S$ of maximum max-back value. $S = S + \{v\}$, $Cutval = Cutval + 2 - 2 * b(v)$. For all $t \notin S$, set $b(t) = b(t) + x_{vt}^*$. If $Cutval < Cutmin$ then $Cutmin = Cutval$ and $Smin = S$.

The idea behind this heuristic is that if one wants to keep the cut small then one should take the vertices that see the current set by the largest amount. Note that the max-back idea is the key notion of the Nagamoshi-Ibaraki minimum cut algorithm (see [622],[623]). The terminology comes from A. Frank.

In general we will not explore the whole graph and will stop the previous loop after a certain number of iterations. We will also store the nodes not in S with a non zero max-back value in three lists:

- A list `supto1` which contains the nodes that see S by more than 1

- A list `inferto1` which contains the nodes that see S by strictly less than 1.
- A list `equalto1` which contains the nodes that see S by exactly 1

These definitions may vary depending on the use we make of them. At this point we will just note that the insertion into S of nodes of the list `supto1` strictly decreases the value of the cut, and if not empty, the next chosen vertex is in that list. Those of `equalto1` will leave the value of the cut unchanged, whereas those of the last list will increase the value of the cut.

Some separation procedures use another greedy heuristic to find minimum cuts which we will describe later. In that heuristic a path is added at each iteration.

15. Graph associated to a vector $x^* \in \mathbb{R}^E$

The *support graph* of $x^* \in \mathbb{R}^E$ is the graph $G^* = (V, E^*)$ where $E^* = \{e \in E : x_e^* > 0\}$. Before performing separation it is useful to do some transformations on this graph.

By shrinking a set $S \subset V$, we mean replacing all vertices in S by a single one s , called a *pseudo-node*, and deleting all edges with both extremities in S . All the edges having exactly one extremity in S are replaced by edges whose extremity in S is replaced by s , the other is unchanged. This may lead to a multigraph, that is there may be more than one edge between a pair of vertices. It is convenient in that case to only deal with a single edge e whose associated weight x_e^* is the sum of all the weights of these edges. As the reader has observed, we will use the same notation, that is x^* , for solution induced by x^* on the shrunk graph.

If one desires the degree inequality to be satisfied on the new node s only sets S with $x^*(\delta(S)) = 2$ can be shrunk.

We say that the shrinking of a set $S \subset V$ is *legal* if every violated inequality in the original graph yields such a inequality in the shrunk graph. This concept is very restrictive. Checking whether there is a violated inequality amounts to checking whether or not vector x^* is a convex combination of representative vectors of tours. We will come back to this later. Finally we define a shrinking *legal* for a class of inequalities, as a shrinking such that if the original graph contains a violated inequality of that class, then so does the shrunk graph.

Some easy recognizable cases of shrinkings are given in [647], two of them are given below. It is very important to perform these shrinkings before starting any separation heuristic, since they very often consid-

erably simplify the solution and reduce the size of the graph we work on.

A *path of 1-edges* is a maximal path P in the support graph G^* such that $x_e^* = 1$ for all $e \in P$.

Most authors advocate replacing a path of 1-edges by a single edge, that is shrinking all the nodes except one extremity of the path. This shrinking is not legal in general for a given class of inequalities, but it is for comb separation. Naddef and Thienel [620] found useful to reduce only those paths with more than 4 edges and then shrink all nodes except the 4 vertices of the two extreme edges. This will leave a path of 4 edges, the middle vertex being the one to which all nodes have been shrunk into. The advantage is that this is legal for most classes of known inequalities and that if one uses the two extremities of one of the extreme edges as a tooth, then we know exactly what vertices we are dealing with.

The second shrinking concerns a path of 1-edges, the extremities u and v of which are adjacent to a common node w with $x_{uw}^* + x_{vw}^* = 1$. In this case we can shrink the node set S of the path. Note that this yields a new edge (s, w) with $x_{sw}^* = 1$, which could yield another shrinking, therefore these shrinking operations have to be done recursively. Padberg and Rinaldi [647] show that this is a legal shrinking.

From now on we assume that these operations have been performed. An example of a graph obtained by such shrinkings is given in Figure 2.25 from the solution of Figure 2.24.

Christof, Reinelt and Wenger use the list of all minimum cuts to shrink the graph down to a graph on $k \leq 10$ nodes, in all possible ways, and look for a violated inequality in the list of inequalities describing $STSP(n)$ for $n \leq 10$, see [184], [185], [187], [822] for more details.

16. Heuristics for Comb Separation

Comb separation has received a lot of attention. The reason is that comb inequalities are the first discovered and the simplest inequalities that do not appear in the integer formulation of the traveling salesman problem. We have already mentioned that the exact separation of the special case of 2-matching inequalities is polynomial.

It is useful to understand how a comb inequality can be violated by x^* . Assuming that all subtour inequalities are satisfied by x^* , each tooth has a coboundary of value at least 2. If each tooth has coboundary value 2, the previously mentioned proof of validity, shows that the coboundary of the handle cannot be less than $2k + 1$ and therefore maximum violation is 1 if all subtour elimination inequalities are satisfied. Therefore if the handle has a coboundary of $2k + 2$ or more, no violated comb with $2k + 1$

teeth can be violated. In general to have violation we must have that the sum of the excesses of the values of the coboundaries over their minimum values be less than 1. That is:

$$x^*(\delta(H)) - (2k + 1) + \sum_{i=1}^{2k+1} (x^*(\delta(T_i)) - 2) < 1 \quad (60)$$

16.1. The biconnected component heuristic

A *cutnode* of a graph is a vertex whose removal disconnects the graph. If a graph does not contain any cutnode it is said to be *biconnected*. A *block* of a graph is a maximal biconnected induced subgraph. Finding all blocks is easy and can be done in linear time (see [787]).

Let G_{-1}^* be obtained from G^* by deleting all edges with $x_e^* = 1$. Padberg and Rinaldi [647] use a heuristic in which each block or union of adjacent blocks of G_{-1}^* is considered as a potential handle H of a comb. The sets of extremities of an edge $e = (u, v)$ with $v \in H, u \notin H, x_e^* = 1$ are used as teeth. The other teeth are chosen among the adjacent blocks, that is blocks which share a vertex with H . We advocate to try to grow by the max-back procedure a tooth from each cutnode of G_{-1}^* contained in H . This procedure is quite effective in the beginning of the Branch-and-Cut procedure. Moreover, it is very fast.

Note that if the paths of edges of value 1 have not been contracted to a unique edge, then one must add to each block the nodes of such paths linking two of its nodes.

Figure 2.24 gives an example of a fractional point of kroA100, the dotted edges correspond to value .5, the others to value 1. Figure 2.25 gives that same graph after contraction following the rules described earlier. The blocks happen here to be exactly the connected components of the graph from which the edges e with $x_e^* = 1$ have been removed and correspond to the nodes of the 6 triangles. Note that in the original graph there was one more block. Several violated 2-matching inequalities are easily found, some in the original graph are comb inequalities which are not 2-matching inequalities. Figure 2.26 shows two of these inequalities and one of the four 2-matching inequalities.

16.2. The max-back Heuristic

In this heuristic, starting from a given node chosen in a certain subset B , we grow a handle using a modified max-back procedure. Once in a while we stop growing the handle and try to grow a number of teeth using again a max-back procedure. If a violated comb has been obtained we stop, else we go back to grow the handle, and so on as long as no

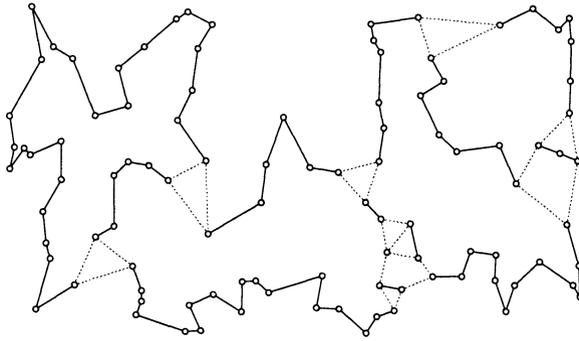


Figure 2.24. Example of a fractional point

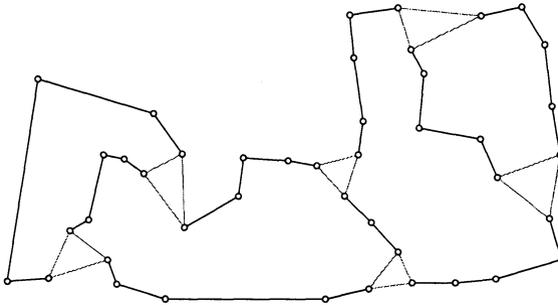


Figure 2.25. The graph of Figure 2.24 after shrinking

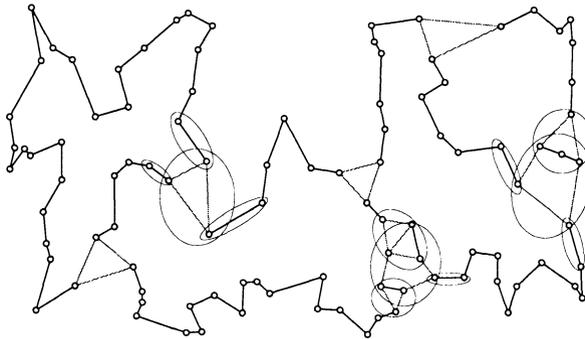


Figure 2.26. Examples of violated combs from Figure 2.24

violated comb has been found and the maximum number of iterations

has not been exceeded. We then choose another node from B until we have tried with all nodes in B . We make this more precise now.

In our case we chose B to be all extremities of paths of 1-edges. Let $u \in B$, we grow S by max-back starting with $S_0 = \{u\}$. Let v be such that $x_{uv}^* = 1$, the idea is that $\{u, v\}$ will be the first tooth. One therefore forbids the max-back procedure to pick up any other node of the path of 1-edges one extremity of which is u . Nodes of max-back value at least one and not incident to a 1-edge linking it to the current growing handle are included in the handle under construction. Those nodes incident to a 1-edge are chosen if their max-back value is greater than a prescribed value (say 1.7). The idea is that we lose a potential “nice” tooth, but on the other hand the coboundary of the handle will decrease in value. Whenever we introduce a node v in the handle, we also update the max-back value of all the nodes adjacent to it, including those which are already in the handle under construction.

Every time the previous procedure would choose a node of max-back value strictly less than 1 and the coboundary value is not too close to an even number, we first try to find a violated comb by searching for the right number of teeth. This number is easily computed knowing the value of the coboundary of the current handle. If that value is strictly between $2k$ and $2k + 2$, we need $2k + 1$ teeth. One tooth is already known, as already mentioned. Nodes of the handle which are incident to a 1-edge the other extremity of which is not in the handle, give us additional teeth. For the remaining teeth, we greedily choose the vertex in the handle with minimum max-back value among those not already in a tooth as the starting node for the next tooth. The rationale behind this is that if a node is poorly linked to the other nodes of the handle, it is likely that we will find a set with small coboundary value containing it that will yield a tooth. The tooth is grown by pure max-back from that node, leaving it grow freely inside or outside the handle, except that it cannot intersect already existing teeth. We do so until either we reach a value very close to 2, or a maximum number of iteration has been exceeded. We then return the best set found. In fact, in view of the search for other inequalities, we return more sets than just the best one, especially if that set contains only one node outside the handle. For other inequalities we will need the information on the best tooth having at least two vertices outside the handle. The weakness of this procedure is that we never change a previously found tooth, hence depending on the order in which we consider the starting nodes, we may or may not succeed in finding a violated comb. This is however unavoidable due to the nature of the greedy heuristics. In the code of Naddef and Thienel [620], the minimum max-back value for the choice of the starting node

of a tooth suffers many exceptions, the main one is as follows. If a path of 1-edges linking say u and v is entirely inside the handle, it is very often the case that a useful tooth will contain the whole path. In this case we consider that u and v as having a max-back value equal to the sum of their real max-back value minus two.

This heuristic is fast, pretty effective and easy to implement. It suffers from the fact that in many instances many max-back values are identical and one has to be lucky to choose the right node. This is especially true at the beginning of the cutting phase procedure. This is why Naddef and Thienel prefer the much slower but much more successful heuristic described in the next section.

Note that, in some way, this relates to Carr's comb separation algorithm. Once the handle is chosen, we choose the nodes in the handle which will be the seeds, as Carr calls them, of the teeth. The difference is that Carr's method also chooses one seed outside the handle for each tooth.

16.3. The Ear Heuristic

We now give another greedy way of growing sets. Given $S \subset V$, an *ear* relative to S is a (possibly closed) path in G with intermediate nodes not in S . In most graph theoretic domains where ears are used, i.e. connectivity or matching theory, a single edge between two nodes of S is considered as an ear, and this path need not be minimal in terms of vertices outside S . Here we will assume that there is at least one node not in S in the path and that, except for the first and last node of the path outside S , none of the nodes of that path are incident to nodes of S .

Given $S \subset V$, by "increasing S by an ear", we mean adding to S the ear that yields a set of minimum coboundary value among all sets that can be obtained this way. Of course, like in the case of the max-back procedure, when growing handles we are faced with the dilemma of choosing or not nodes linked to S by a 1-edge. We use the same rule as in the previous case. In fact the only difference with the previous section is the way sets are grown. In some way we control better the way the set grows and therefore this method typically gives much better results. The only problem is that one has not yet found an efficient method to find the best ear: if we could improve this part substantially, the running times reported below using the Naddef and Thienel separation heuristics would be drastically decreased.

16.4. About minimum cuts

The two comb separation heuristics we will describe later on, as well as the exact separation algorithms for violated combs and domino inequalities in planar graphs we have mentioned, make extensive use of the structure of minimum cuts. Let us assume that all subtour elimination are satisfied by x^* , that is $x^*(\delta(S)) \geq 2$ for all $S \subset V$. Therefore the value of a minimum cut is 2, since each node defines such a cut. Finding all cuts of value 2 is relatively easy. There are several ways of storing all these cuts in a compact form, either by a *cactus* representation of Lemonosov, Karzanov and Timofeev (see for example [310], [309], [241], [822]), by a PQ-tree of Booth and Lueker [131] or by a poset representation (see [342]).

The main property of minimum cuts which is used is that if the sets S_1 and S_2 define minimum *crossing cuts*, that is $S_1 \cup S_2 \neq V$, $S_1 \cap S_2 \neq \emptyset$, $S_1 \setminus S_2 \neq \emptyset$ and $S_2 \setminus S_1 \neq \emptyset$, then $S_1 \cup S_2$, $S_1 \cap S_2$, $S_1 \setminus S_2$ and $S_2 \setminus S_1$ all define minimum cuts. In particular, x^* representing the capacities of the edges, we have $x^*(S_1 : V \setminus (S_1 \cup S_2)) = x^*(S_1 \setminus S_2 : S_1 \cap S_2) = x^*(S_2 : V \setminus (S_1 \cup S_2)) = x^*(S_2 \setminus S_1 : S_1 \cap S_2) = 1$, $x^*(S_1 \cap S_2 : V \setminus (S_1 \cup S_2)) = 0$ and $x^*(S_1 \setminus S_2 : S_2 \setminus S_1) = 0$. Figure 2.27 summarizes this.

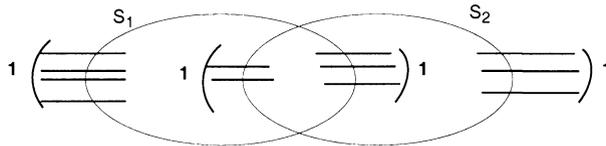


Figure 2.27. Two crossing minimum cuts

We only briefly describe here the PQ-tree structure to store minimum cuts. For each cut the shore that does not contain a prescribed node a is stored. A *PQ-tree* is a rooted tree with each internal node having at least two children and labelled either *P*-node or *Q*-node. Following [28], for a node u of a *PQ*-tree, we let $D(u)$ be the set of leaves of that tree that are descendants of u .

A *PQ*-tree represents all the shores of the minimum cuts not containing node a if and only if each shore is either:

- (i) $D(u)$ for some node u of the *PQ*-tree
- (ii) The union of $D(u)$ s for *consecutive* sons of a *Q*-node
- (iii) The union of $D(u)$ s for any subset of sons of a *P*-node

and each such set is the shore of a minimum cut. Note that the leaves are exactly the nodes of $G \setminus \{a\}$. Without going into the details of the

PQ -tree structure, from the preceding lines it is obvious that the sons of a Q -node are not placed in an arbitrary order.

16.5. The Domino Heuristic

Assuming all subtour elimination inequalities are satisfied, a $2k + 1$ tooth comb is maximally violated if the handle has a coboundary value of $2k + 1$ and each tooth is tight. That is, all teeth define minimum cuts, but also so do the intersection of each one with the handle and its intersection with the complement of the handle. Moreover for each tooth T_i we have: $x^*(T_i \cap H : H) = x^*(T_i \cap H : T_i \setminus H) = x^*(T_i \setminus H : V \setminus H) = 1$ and therefore $x^*(H : V \setminus (H \cup_{i=1}^{2k+1} T_i)) = 0$; see Figure 2.28 where the only edges that can cross the boundary of one of the sets that define the comb are shown.

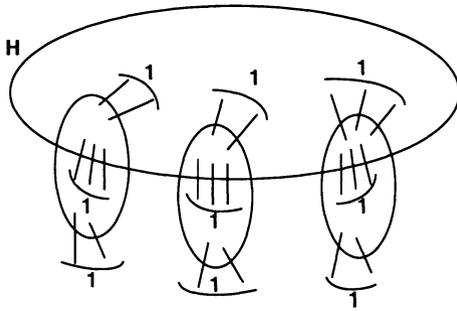


Figure 2.28. Edges in maximally violated comb

Applegate et al. [28] call a set $T = A \cup B$, such that $A \cap B = \emptyset$ and $x^*(\delta(T)) = x^*(\delta(A)) = x^*(\delta(B)) = 2$ a *domino*. We call an edge $e \in (A : B)$ such that $x_e^* > 0$ a *crossing edge* of the domino $T = A \cup B$.

If we find a set of $2k + 1$ pairwise node disjoint dominoes such that the union of their crossing edges form a cut of the support graph of x^* , then we have found a maximally violated comb. The handle is one of the shores of that cut. Such a set is called a *cutter* in [28]. This is the idea behind the heuristic proposed by Applegate et al. [28].

A *necklace* is a partition $V_0, \dots, V_i, \dots, V_k$ of V into tight sets such that $S = (V_i \cup V_{i+1})$ forms a domino for all i , where subscripts are taken modulo k . Each Q -node v of a PQ -tree compatible with x^* defines a necklace with $V_0 = V \setminus D(v)$ and $V_i = D(v_i)$ where $v_1, \dots, v_i, \dots, v_k$ are the sons of v in that order.

Given a PQ -tree compatible with x^* , let \mathcal{D} be the set of all dominoes of all the necklaces obtained from the Q -node of that tree. Applegate et al. in [28] search \mathcal{D} in a clever way in order to find a set of dominoes

which defines a cutter and therefore yields a violated comb. We refer the reader to [28] for more details. It very often happens that the support graph of a fractional solution is planar and therefore, one may now rather try to use the newly available algorithms of Fleischer and Tardos [310], of Letchford [559] or of Caprara, Fischetti and Letchford [161], although this latter has, so far, not yet been implemented.

16.6. The Consecutive Ones Heuristic

We just give here a flavor of the consecutive ones heuristic of Aplegate et al [28]. It is easier to understand if one assumes that the minimum cut has a value of two, and that we have a PQ -tree that represents the set of all minimum cuts. This is not really a restriction since some TSP codes as the one of Naddef and Thienel [620] and [621], only separate on other constraints when all subtour elimination inequalities are satisfied.

A set of subsets of a given ground set is said to have the *consecutive ones property* if one can number the elements of the ground set in such a way that all the elements of any subset have consecutive numbers. Given $S_i \subset V \setminus \{a\}$, $i = 1, \dots, k$, where a is as prescribed node as in Section 16.4, if all these subsets are tight for a certain tour Γ , then they satisfy the consecutive ones property since numbering consecutively the nodes in the order induced by Γ yields the desired property.

Let H be a vertex set such that $x^*(\delta(H)) < 4$. Any tour Γ either intersects the coboundary of H in two or at least four edges. Let's try and see if it is possible in two edges, that is we want to test whether H can be tight, when all the currently tight sets remain tight. There is an algorithm that enables to refine the current PQ -tree to incorporate the cut defined by H or returns a failure message together with a certificate explaining why this is not possible. This certificate has the form of three tight sets T_1, T_2, T_3 which are incompatible with the fact that H could be added to our set of subsets preserving the consecutive ones property. The sets H, T_1, T_2, T_3 define a violated comb inequality since the three sets T_1, T_2, T_3 are either pairwise disjoint or one contains the two others, which do not intersect.

The problem is the choice of the set H . It could be the handle of a former violated comb, plus or minus some nodes. That is, one starts with the handle of a former comb, and possibly increase it by the max-back procedure described earlier.

Research Question 60 *Assume the set of minimum cuts is stored in a cactus instead of a PQ -tree. Can we obtain a certificate that a given*

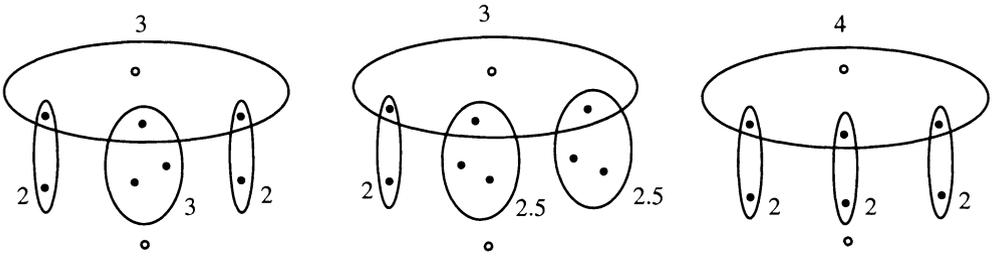


Figure 2.29. Examples of non violated comb inequalities

set cannot define a minimum cut if all the current minimum cuts remain minimum? Same question if one uses a poset representation.

To make the question more precise: Take a set S which does not define a minimum cut. Assume the solution x^* has changed in such a way that all the minimum cuts remain minimum. Can S now also define a minimum cut? In case of a NO answer a good certificate is asked for.

17. Separation of multi-handle inequalities

17.1. How do multiple handles help?

Most of the non-comb inequalities for $STSP(n)$ we have described have combs as building blocks and therefore have more than one handle. If one wants to design specific separation heuristics (in a template paradigm) it is important to understand how the addition of a handle can yield a violation when none of the constituting comb inequalities is violated. This is explained in details in Naddef and Thienel [621]. Figure 2.29 shows non violated (in fact, tight) comb inequalities. These could have been found in our search for violated combs.

All the figures of this section will follow the following convention as far as numbers are concerned: cut values will be given in floating point format, i.e. 2.0, 3.2, to distinguish them from set coefficients which are integers. In some of the forthcoming figures we also have drawn in bold the non violated comb inequality from which we have started the search for a multi-handle violated inequality.

The key remark in understanding how one of these inequalities can be violated, is to see that if a tooth with large cut value intersects *properly* enough handles with cut value close to an odd integer, then it is likely to be part of a violated inequality. We illustrate this in the following figures.

Figure 2.30 shows how violated path inequalities can be found on the two first cases of Figure 2.29, as long as one can find a second handle that

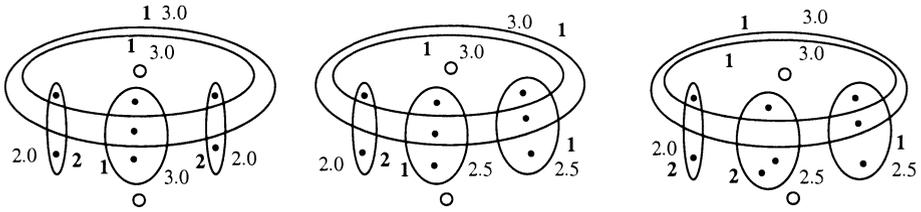


Figure 2.30. Examples of violated path inequalities with 2 handles

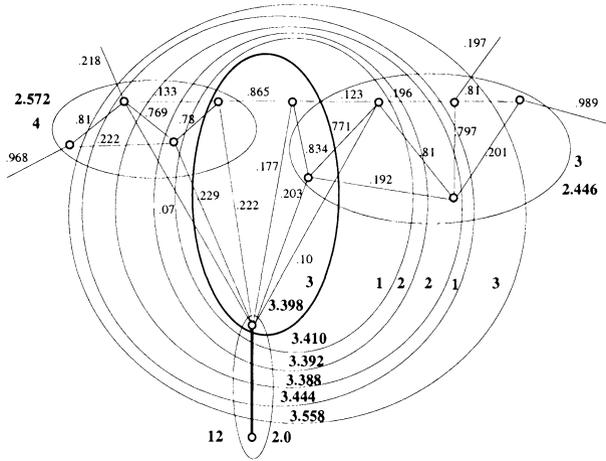


Figure 2.31. Example of a violated path inequality in ts225

intersects the teeth as shown (this is what we meant by *properly*). The last of these drawings shows a case in which the second handle intersects one of the teeth in such a way that the tooth gets a higher coefficient, yielding a weaker inequality. In the two first drawings, the inequalities are violated by 1, it is violated by only .5 in the last one. Note that teeth with cut value close to 2.0 are penalized only marginally if they receive higher coefficients. Figure 2.31 shows an example on the instance ts225, of a non violated comb inequality (central handle in bold) and a violated path inequality with the same set of teeth. The RHS is 76 and the LHS 70.3.

The same argument on teeth with high cut value holds if the handles are disjoint instead of being nested. This yields either bipartition or clique tree inequalities. Figure 2.32 refers to the case of clique trees if only one tooth has a large cut value, and Figure 2.33 to the case in which more than a tooth has such a cut value.

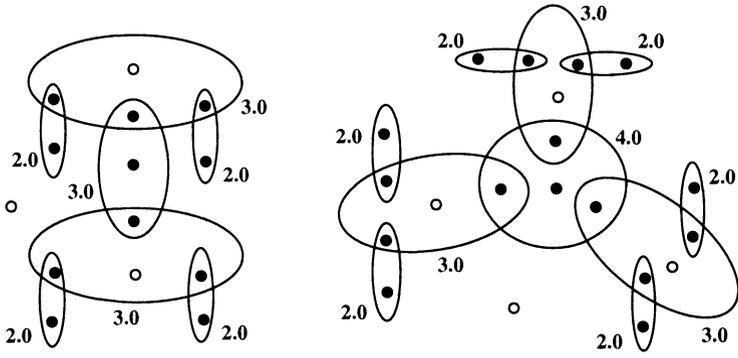


Figure 2.32. Examples of violated clique trees

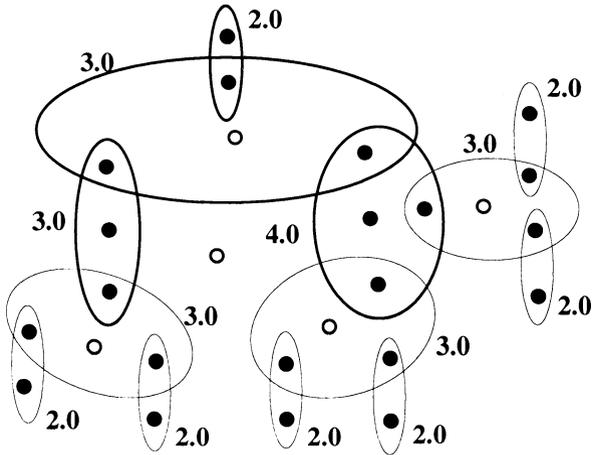


Figure 2.33. Example of violated clique tree

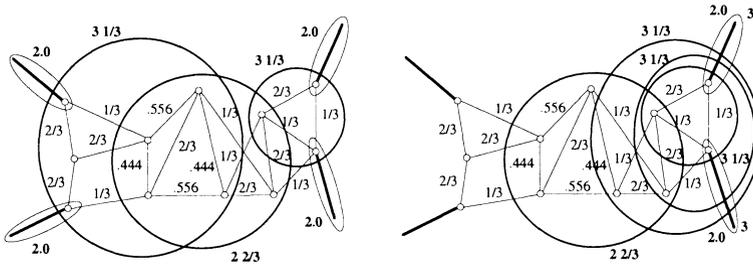


Figure 2.34. Violated clique tree and path inequalities from pr299

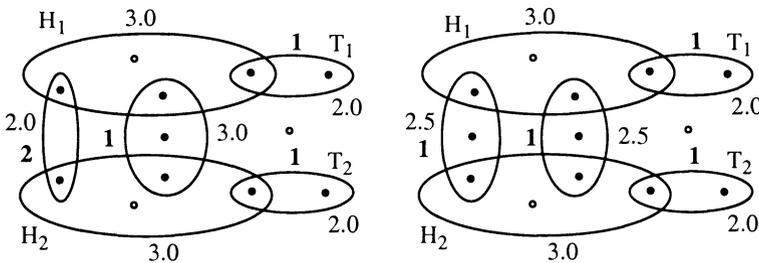


Figure 2.35. Example how ladders can be violated

Figure 2.34 gives an example of a clique tree inequality that can be found this way. It is taken from pr299. The RHS is 18 and the LHS 17.333. In this case there is also a violated path inequality, as shown in the second part of the figure; the two teeth made up by an edge have coefficient 3, the RHS 26 is and the LHS is 24.667. The path inequalities defined by any two of these three handles is also violated.

Ladder inequalities may also help in the same way, but since there can only have two handles, one cannot expect them to accept too high cut values for the teeth. Using the handles to compensate for high cut values for the teeth adds up in using a ladder inequality as a bipartition inequality on two handles (see Figure 2.35). We will see later on how ladders can help to compensate for bad cut values of handles, that is values too close to an even number. The correcting term will then be the key factor. Figure 2.36 gives an example of ladder violation in gil262, the RHS is 16, the LHS is 15. Figure 2.37 gives two violated ladders in a fractional solution to pr439, one with disjoint handles, the other with nested handles, in both cases the RHS is 18, the LHS is 17.667.

Note that so far we have shown how to deal with cases related to combs of the two first types of Figure 2.29, that is in the case in which non-violation is due to teeth of too high cut value. In the last case of that figure, the handle has a too high cut value and finding extra handles

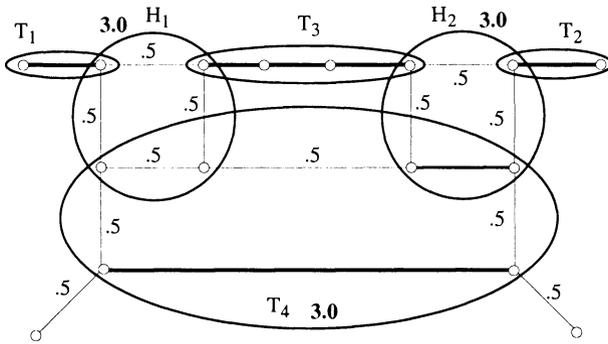


Figure 2.36. Example of ladder violation in gil262

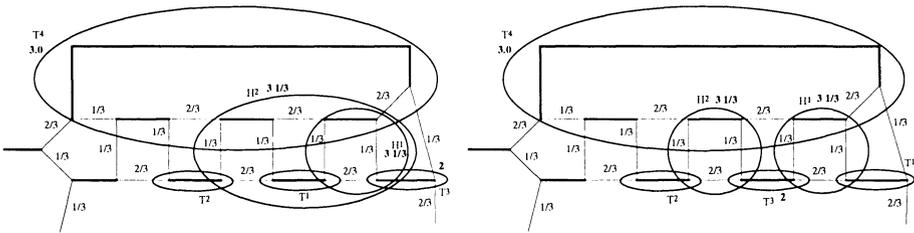


Figure 2.37. Example of violated ladders from pr439

in a way or another would not help. The only known possibility is to use the correcting term of the ladder inequalities. Figure 2.38 shows how this can happen on an example from a fractional solution to pr136. The first drawing shows a tight comb with a handle of cut value 4, the second shows a violated ladder in which the violation comes from the correcting term induced by the edge with black filled extremities. Figure 2.39 gives two examples taken from two fractional solutions to pr439 in which the handles are nested. In both cases the violation comes from the correcting term given by the edge, the two extremities of which are filled in black. All combs that can be built using one of the handles and the three teeth it intersects are not violated. The violation in the first case is $2/3$ and 0.518 in the second.

How can we hope to find such violated multi-handle inequalities? Naddef and Thienel [621] suggest the following strategy.

When searching for teeth in combs, if the best found tooth only has one node outside the handle and its cut value is not close to 2, also return the best tooth with at least two nodes outside the handle. For example, in Figure 2.31 the edge of value .78 in the coboundary of the inner handle yields a tooth of cut value 2.44 which is better than the 2.57

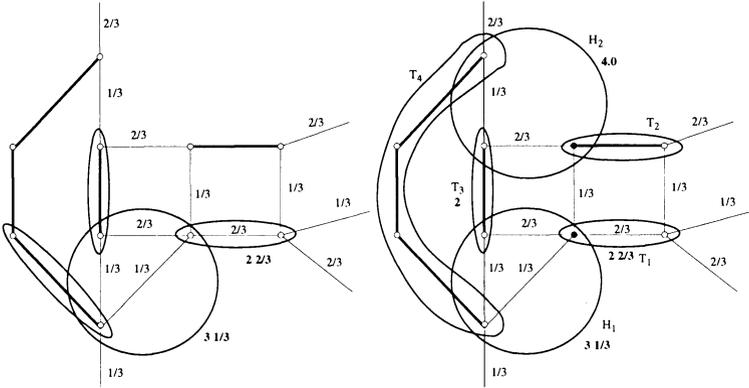


Figure 2.38. Violated ladder from pr136 with correcting term

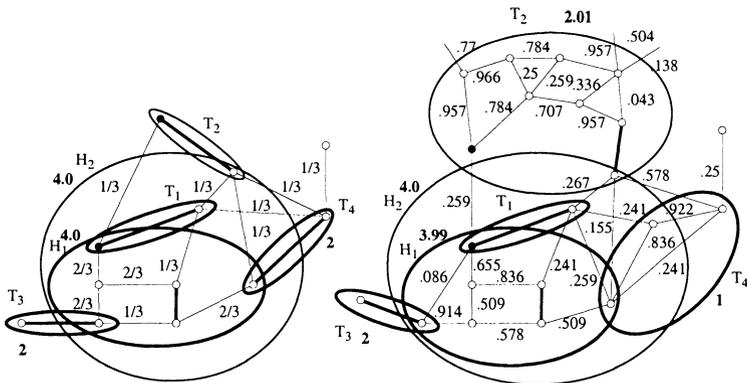


Figure 2.39. Violated ladders from pr439 with correcting term

cut value of the chosen tooth. This is not computationally expensive, since in the search for a tooth one goes anyway beyond the set which is finally returned. The idea is that a tooth with large cut value having only one node outside the handle will be a handicap towards violation that cannot be compensated by finding extra handles as shown in the previous section.

The general strategy is based on a non-violated comb. If we have large enough teeth, in terms of number of nodes outside the handle, we search for a violated path inequality. We try to grow handles, starting with the current one such that their cut values are low enough and that they yield small coefficients for the teeth with large cut value. We can also try to find handles included in the previous one. All this is done by the max-back or the ear procedure described earlier.

If one does not succeed, then one tries to find another type of violated inequality, either in a way similar to that for paths (ladder inequalities with nested handle) or by first growing a new handle which intersects one or several teeth of large cut value, and then searching for new teeth and so on. Everything is done in a greedy way. These simple greedy procedures give surprisingly good results as will be reported in the section on computational results. The interested reader is referred to [620] and [621] for more details.

18. Separation outside the template paradigm

So far we have developed separation heuristics that try to find a violated inequality from a prescribed family of valid inequalities. Some other paths have been explored. Applegate, Bixby, Chvátal and Cook in [30] and in [31] have explored another direction which is the topic of this section.

Figure 2.40 shows a portion of a fractional point of the instance fnl4461. When developing a STSP solver, one very often looks at such drawing of a fractional solution. The fact that one can quite easily visualize fractional solutions, is certainly one of the keys to the progress that has been possible for this particular problem.

When doing so, one looks at small parts of the drawing to try and figure out what one has missed in terms of violated inequalities. In other words, one locally searches for violated inequalities. Therefore we will refer to the following procedure as the *local cut* procedure (which in fact was the first name their authors gave it). For example one may wonder if there is a violated inequality involving the black filled nodes of Figure 2.40. We will see that the answer “yes” will be very easy to

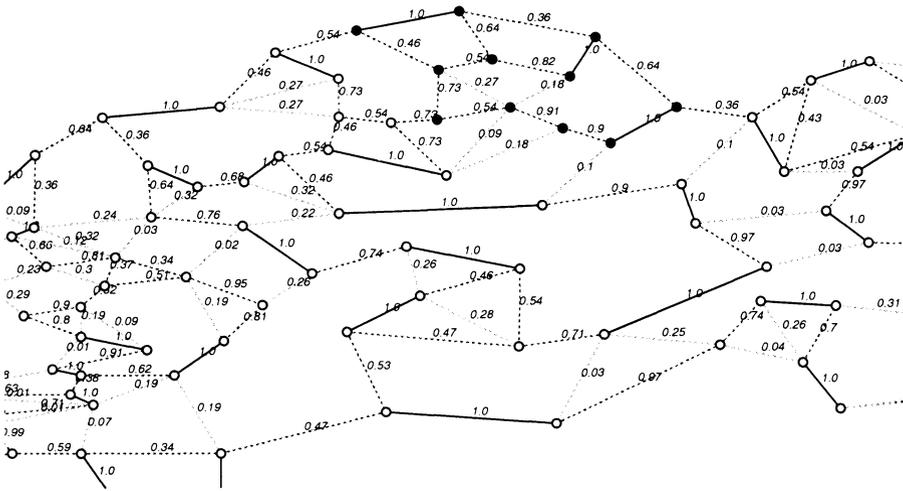


Figure 2.40. Part of a fractional solution of fnl4461

find. Much harder will be the answer to the question of knowing which valid inequality is violated.

Let us contract all the other nodes into a single one to obtain the graph of Figure 2.41 in which the larger circled node represents the node obtained from the shrinking. We will refer to that node as the *pseudo-node*. Note that in both figures, all nodes may in fact represent more than a single node in the original problem because we have applied the traditional legal graph reductions described earlier. This will only matter at the end of our computations.

It happens here that the sum of the x_e^* on the star of the pseudo-node is two. If we impose this condition, then the *local search* for violated inequalities may turn short for lack of interesting sets. This is why in their development, Applegate et al. [31], do not require this condition. If that value is less than two we have a violated subtour elimination inequality, else it could be anything greater or equal to two. To make things easier to expose, we will assume that this sum is two and try to adapt their method to this case. Let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the small graph obtained after shrinking a very large set of nodes into a single pseudo node. We continue calling x^* the fractional solution induced on it by our fractional vector x^* .

If in \tilde{G} the solution x^* (see Figure 2.41) is a convex combination of incidence vectors of Hamiltonian cycles, then x^* violates no valid inequality in \tilde{G} . Else there is necessarily at least one such inequality.

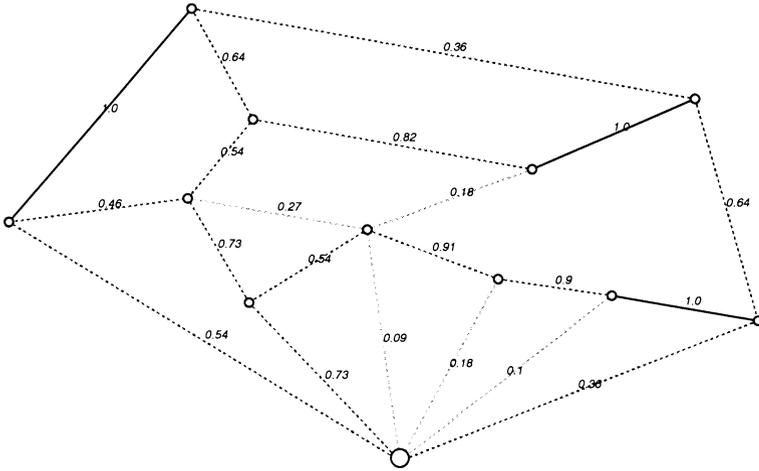


Figure 2.41. The same solution after shrinking

Note that any Hamiltonian cycle candidate to be used in such a convex combination must contain all edges e such that $x_e^* = 1$ and no edge for which $x_e^* = 0$. We call such tours *compatible* with x^* . In our example there are only four possible such Hamiltonian cycles. None uses the edge of value .18 not incident to the pseudo node, and therefore this solution is not a convex combination of incidence vectors of Hamiltonian cycles and it is worthwhile to search for a violated inequality in this graph.

To make things more precise, one can solve the following linear program which tries to write x^* as a convex combination of incidence vectors of Hamiltonian cycles. The summations are on all compatible Hamiltonian cycles Γ except when expressed differently:

$$\max \sum_{\Gamma} \lambda_{\Gamma} \tag{61}$$

$$\sum_{\{\Gamma: e \in \Gamma\}} \lambda_{\Gamma} = x_e^* \text{ for all } e \in \tilde{E} \tag{62}$$

$$\sum_{\Gamma} \lambda_{\Gamma} = 1 \tag{63}$$

$$\lambda_{\Gamma} \geq 0 \text{ for all } \Gamma \tag{64}$$

This problem can be solved by generating first all compatible Hamiltonian cycles, which in general are in small number if the number of nodes is small and we have some edges e with $x_e^* = 1$, since these restrict a lot the number of choices. Note that \tilde{G} is in general not dense since,

as we will see in the next section, we work on a very restricted set of edges (variables). This problem can be also solved by dynamic column generation. Each column generation is a TSP problem using the dual variables as costs, but on a small instance with many fixed variables. We will comment on this point later on.

If the above LP has a solution then we are out of business, else by the Farkas lemma there exists an inequality $fx \leq f_0$ which explains this infeasibility, i.e. such that $fx \leq f_0$ for all compatible tours and $fx^* > f_0$. Some commercial linear programming software, such as CPLEX, provide a function that return the coefficients f_e for $e \in \tilde{E}$ of such an inequality.

In [31] the following slightly different linear program is solved in order to get the coefficients f_e directly from the dual without having to use an ad hoc procedure. Here again Γ is a generic compatible tour.

$$\max \quad s \tag{65}$$

$$- \sum_{\{\Gamma:e \in \Gamma\}} \lambda_\Gamma + sx_e^* + w_e = 0 \text{ for all } e \in \tilde{E} \tag{66}$$

$$\sum_{\Gamma} \lambda_\Gamma - s = 0 \tag{67}$$

$$-1 \leq w_e \leq 1 \text{ for all } e \in \tilde{E} \tag{68}$$

$$\lambda_\Gamma \geq 0 \text{ for all } \Gamma \tag{69}$$

This linear program is unbounded if and only if x^* is a convex combination of incidence vectors of tours of \tilde{G} . Else letting f_e , for $e \in \tilde{E}$ be the dual variables associated to constraints (66), f_0 that associated to constraint (67), u_e and v_e those associated to the bounds on the components of w , the dual writes:

$$\min \quad \sum_{e \in \tilde{E}} (u_e + v_e) \tag{70}$$

$$- \sum_{e \in \Gamma} f_e + f_0 \geq 0 \text{ for all compatible tour } \Gamma \tag{71}$$

$$\sum_{e \in \tilde{E}} f_e x_e^* - f_0 = 1 \tag{72}$$

$$f_e + u_e - v_e = 0 \text{ for all } e \in \tilde{E} \tag{73}$$

$$u_e \geq 0, v_e \geq 0 \text{ for all } e \in \tilde{E} \tag{74}$$

Any feasible solution to this dual problem yields an inequality $fx \leq f_0$ which is valid for all compatible tours as shown by constraints (71) and such that $fx^* = f_0 + 1 > f_0$ by Constraint (72). Note that due to the Objective Function (70), if one finds a violated inequality, one finds

one which minimizes $\sum_{e \in \tilde{E}} |f_e|$, so it is in some sense a most violated inequality.

The steps in [31] (slightly modified for sake of simplicity to the case of a pseudo-node of “degree” two) are:

- 1 Turn the inequality $fx \leq f_0$ into one with integer coefficients.
- 2 Transform it into an inequality valid for all tours which only use edges in \tilde{E} .
- 3 Transform it into a facet inducing inequality on \tilde{G} .
- 4 Perform a sequential lifting to compute the coefficients on the edges not in \tilde{E} . Assume one has transformed the inequality into one in “greater or equal” form, $gx \geq g_0$, which can easily be done by multiplying by -1 both sides and then adding some degree equalities in order to bring back all coefficients non negative. Then each lifting operation refers again to a STSP on a very small graph, which does not mean that things will be necessarily easy. Experiments carried out by Jünger, Reinelt and Rinaldi [474] show that when the coefficients of the objective function are those of weird facet inducing inequalities, then the TSP may be extremely difficult to solve by Branch and Cut even for very small instances. For this last reason it is advisable to put a time limit on this phase and return a failure message if it exceeds that time.
- 5 Transform the inequality into a facet inducing inequality using a tilting procedure inspired by Minkowski. Procedure also used in Step (2). Note that Applegate et al. do not need this step, but when dealing with Hamiltonian cycles lifting an edge may increase the dimension of the underlying polytope by more than one unit, which is not the case when one uses closed walks. On the other hand, Applegate et al. need a more sophisticated routine for the column generations and the lifting procedures which are no longer TSPs.
- 6 Put in TT form. For technical reasons due to their Branch-and-Cut code, Applegate et al. only keep those TT inequalities that can be put in closed set form.
- 7 Finally the shrunk nodes are expanded using 0-liftings.

Applegate et al. report excellent results with their procedure. We will report some of their results in the section devoted to computational results.

19. Branch-and-Cut implementation of the STSP

The Branch-and-Cut method is very simple in its principle, but once we want to use it there are many technical points to address. Figure 2.42, taken from Jünger, Reinelt and Rinaldi [474], shows the flowchart of a Branch-and-Cut implementation. For a detailed study on Branch-and-Cut we refer the reader to Jünger and Thienel [477], [478], Thienel [790], Jünger, Reinelt and Rinaldi [474]. From now on lp stands for linear program. We will only address the following points:

- Management of variables

active variables: What variables should one work with?

pricing: How often should we apply variable pricing, and how should one do it?

fixing: How to perform variable fixing?

- Management of constraints

separation: What separation strategy should one choose?

storing: How should one store the constraints?

cleaning: How to keep the lp of reasonable size?

- How to obtain a good feasible solution?

- Branching

When should one decide to stop separating and start branching?

How should we perform the branching and how much effort should one invest in it?

Which subproblem has to be solved next?

We will have to refer very often to two implementations of the Branch-and-Cut algorithm for the STSP, that of Applegate, Bixby, Chvátal and Cook, and that of Jünger, Naddef, Reinelt and Thienel. We will refer to them by ABCC and JNRT.

19.1. Variables

Since we are working on complete graphs it is out of question to have all the variables present in the lps . One starts with a reasonably sized set of variables. The set of corresponding edges is usually referred to as the edge set of the *sparse graph*. There are various ways of choosing the

that the k nearest neighbors is not in general a good choice since some instances have optimal solutions containing edges linking some nodes to far neighbors. For example he points out that in the optimal solution to att532 there is an edge that links a node to its 22nd nearest neighbor. Helsgaun also noted that an optimal tour contains between 70% and 80% of the edges of the minimum cost 1–tree. A minimum cost 1–tree is obtained by taking a minimum cost spanning tree of $G \setminus \{1\}$ to which one adds the two least cost edges incident to node 1. The *marginal value* μ_e of an edge e not in the minimum cost 1–tree T , is defined as the minimum cost of a 1–tree containing edge e minus the cost of T . A marginal cost of zero means that there exists a minimum cost 1–tree that contains e , else this cost is positive. A good choice of the sparse graph is to take all edges of T and all the edges with marginal value less than a given value α . Helsgaun shows in [446] how this can be done efficiently.

Of course we have no guarantee that the optimal solution is among those edges. Therefore one must regularly perform what is called a *pricing*, complete or partial. If one started with the k nearest neighbor graph, it seems quite logical to first search among the edges in the $k + t$ nearest neighbor graph, with t small (e.g., $t=2$ or 4). The edges of the distance 2 closure of the Delaunay triangulation, that is the edges linking nodes at distance 2 on the Delaunay triangulation, seems also a good choice in the other case. These edges are often referred to as those of the *reserve graph*. We will see that this pricing operation, in some sense, guides many decisions in the design of the Branch-and-Cut code. The pricing is done using the optimal solution to the dual problem. How often should it be performed is not an easy question to answer. For separation heuristics based on template paradigms, too many edges seem to be a handicap because the lp solutions seem to become too fractional. For example starting directly with the five nearest neighbor graph (instead of the three), very often yields a worse running time and more branch and cut nodes in the enumeration tree. Conversely, it happens that trying to avoid a complete pricing, one will spend quite some time separating while the lower bound is over the (yet unknown) optimal value. In the JNRT code, a reserve pricing is performed every 10 iterations, a complete pricing is performed if less than 2 variables were priced in from the reserve graph. A complete pricing is necessary in all cases when the solution to the lp is feasible (integral), or when the lower bounds exceeds the upper bound. Even in this case, only a restricted number of variables is generated before one reoptimizes. In all cases of the TSPLIB only a very small percentage of the variables are used.

Variable fixing is an important point in Branch-and-Cut. A variable can be fixed to 0 (resp. 1) if one does not cut off all optimal solutions when we never (resp. always) take that edge. This can be done in various ways. The first is by *reduced cost*. Let GUB (Global Upper Bound) be the best known value of a tour, LB (Lower Bound) be the optimal value of our current lp and assume that a complete pricing has shown that the current lp solution x^* is globally optimal. Let e be such that either $x_e^* = 0$ or x_e is not a variable of the lp. Let \bar{c}_e be its reduced cost and assume $\bar{c}_e > 0$. If $\bar{c}_e + LB \geq GUB$ one can *fix* the corresponding variable to zero, delete it from the lp, if necessary, and definitively forget about it. One advantage is that pricing will not have to be performed on it again. Note that the case “=” in the preceding formula may eliminate all optimal solutions, but in this case it is not important since one already has at hand such an optimal solution, namely the solution that yields the current GUB . With the same argument one can fix a variable x_e to 1 which is such that $x_e^* = 1$ and $LB - \bar{c}_e \geq GUB$ (this variable is non basic and at its upper bound value).

Variables can be fixed also by *logical implication*. For example, if two incident edges have been fixed to 1, then all other edges incident to the common node can be fixed to 0. If all edges incident to a node except two are fixed to 0, then the two remaining ones can be fixed to 1. The edge linking the extremities of a path of edges fixed to 1 can be fixed to 0. These variable fixings can be done at any node of the search tree, but are only valid in the corresponding subtree. One usually talks of variable *setting* in case the node is not the *current* rootnode of the search tree, where by *current* rootnode we mean the lowest node in the search tree which is a common ancestor of all active nodes.

19.2. Constraints

As one would expect from the name of the method, constraint generation is the key point in a Branch-and-Cut code. As seen in the previous sections there are a variety of different constraint types. Should one give priority to some over the others? The strategies of the two latest codes devoted to the TSP diverge quite a bit in the separation strategies. The JNRT code will first search for violated subtour inequalities and only search for other violated inequalities once no subtour inequality is violated. It was generally taught that it was too costly to do so. The computational results that will be reported in the next section show that it is not the case. The ABCC code does not do so, following the strategy of all the preceding codes of Hong and Padberg, Grötschel and Holland, and Padberg and Rinaldi. The JNRT code only relies on

template paradigms, the ABCC code also uses more general separation procedures.

There are many theoretical ways of measuring the strength of a facet inducing inequality (see [618]).

- The number of tours that are on the facet.
- The volume of the subtour elimination polytope it cuts off.
- Its distance to the closest vertex of the subtour elimination polytope it cuts off.

Practically, the only measure that counts is the help in solving an instance? In this respect subtour elimination inequalities and comb inequalities are the strongest inequalities. For the other inequalities it depends a lot on the instance. Path inequalities seem to be the next useful family.

Before the work of Applegate, Bixby, Chvátal and Cook, several groups have thought of using general cutting planes for integer programming such as the Gomory cuts or the lift and project cuts of Balas, Ceria and Cornuéjols [68] and [69]. The major difficulty with this approach deals with the management of the variables. Remember that we are working with a sparse graph, that is not all variables are considered at a given time, and that one must be able to do pricing efficiently. To do so one must be able to retrieve very easily the coefficient of any variable in a given inequality without storing it explicitly. This can easily be done if the inequality is in closed-set form: one stores each set that defines the inequality, together with its coefficient, then the coefficient of an edge is just the weighted sum of the coboundaries it belongs to. This is why the ABCC code only keeps closed-set form inequalities in its general separation procedure.

Linear programs tend to grow very big, therefore one has to remove regularly those constraints that are no longer active. A good idea is also to remove those that have been very little active in the past k iterations, that is have had an optimal associated dual variable close to zero. This has to be done with caution in order not to destroy the basis. Note that we never delete variables except when fixing to 0. This may be necessary for some large instances in which variables would flow in. Removed subtour elimination inequalities are discarded since they can easily be recovered, the others are put into a pool which is regularly searched through. A too costly strategy for large and difficult instances and which has been in use in the first TSP codes, is to search first the pool for violated inequalities, and then call the separation heuristics in case of failure. This has the advantage of guaranteeing that no inequality

is duplicated in the pool. This strategy is no longer used because the pool of constraints grows very big and searching it is very time consuming. One must then make sure not to store the same inequality twice.

19.3. Upperbounding

We now address the problem of finding a good tour. The aim is on one hand to enable efficient variable fixing, and on the other not to explore useless branches of the search tree. Moreover, a very good upper bound makes the choice of the tree exploration strategy less critical.

Traditionally a heuristic is called to yield a first value and a tour, the edges of which are used in the sparse graph. Most codes try to exploit the lp solution, i.e., they try to take advantage of the information the lp solution x^* carries. It seems more probable that an edge e with x^* close to 1 is in an optimal solution than one with such a value close to 0. Based on this, an initial tour is built and a Lin-Kernighan ([563],[446]) type heuristic is called to try and improve it. It is important not to try and improve twice a same tour. This can be done using a hash function, see [475], [446] and [587].

Any integer feasible solution of the lp encountered on the way also can yield a better tour.

19.4. Branching

Branching occurs when one has renounced to solve the instance with the pure cutting plane method. This may occur either because one does not find any more violated inequalities or because the lower bound has not progressed in a significant manner in the past iterations. This last phenomenon is known as *tailing off*. Before branching a complete pricing is performed in order to be able to fix as many variables as possible, since these fixings will be definitive. The value of the objective function of the current lp, that is the lower bound, is known as the *root value* or *root bound*. We assume the reader familiar with the Branch and Bound terminology.

Since, in some sense, resorting to branching is felt as a failure, not much attention has been dedicated for a long time to this operation. Unfortunately powers of 2 grow very fast and it has become obvious that one must do that operation carefully.

The objective of branching is to split a certain subset of tours into two smaller subsets, to each subset corresponds a lp relaxation which does not contain the current fractional solution as a feasible solution. This can be done in various ways. The classical way is to choose some edge e such that the corresponding variable x_e^* is fractional. The tours

are split into those that contain e and those that do not. This is done by modifying the bounds of the corresponding variable in the current lp, moving the lower-bound to 1 in one case, the upper-bound to 0 in the other. This creates two new problems that are added to the list of problems to solve. To choose among all fractional variables, the mostly used criterion is to choose one of value close .5, and among those one with highest cost. Clochard and Naddef [203] advocate using subtour inequalities to perform branching. Let $S \subset V$, with $x^*(\delta(S))$ not too close to an integer value, say $2k < x^*(\delta(S)) < 2k + 2$. Then one can split the tours into those that cross the boundary of S at most $2k$ times and those that do so at least $2k + 2$ times. This is done by adding the appropriate inequality to the current lp. The difficulty is in finding a convenient set S . In the JNRT code this is done by using handles of previously found combs, star or clique trees, but there is a lot of room for improvement in this choice.

The branching inequality could be anything else. Another possibility is, given two disjoint sets $S, T \subset V$, split the tours between those which use at least an edge from $(S : T)$ and those which don't. This is efficient if the shortest edge of $(S : T)$, in terms of cost (not x^* value) is large and a fractional amount of these edges are used in the solution x^* , an amount close to .5 also seems a good choice. Figure 2.43 illustrates this last branching rule on a fractional solution to fl1400. We have not put in the edge values. Each cluster of nodes A, B and C contains several hundreds of nodes. We have $x^*(A : B) = x^*(A : C) = x^*(B : C) = 0.5$. There is only one edge e between B and C with non-zero x^* value, and it is likely to be selected by the standard criterion (close to .5 and high cost). What will happen on the side of the search tree in which one will have set $x_e = 0$? Since many edges linking B and C have costs very close to that of e , it is very likely that one will appear with that same value (or several adding up to that value). The result will be a problem that will have a lower bound almost equal to that of its father and no progress will have been made. This is typical of a case in which one should branch by saying that $x(B : C) = 0$ on one side and $x(B : C) \geq 1$ on the other side. Two other candidates are of course $x(A : C)$ and $x(A : B)$. Note that this fractional point contains some maximally violated combs which most certainly could be found using the domino separation procedure, but not by the heuristics of Naddef and Thienel, which is certainly why some codes such as the ABCC code do very well on this instance and the JNRT code does very poorly. One of these combs is shown in Figure 2.43, the set C which is one of the teeth, the other sets defining the comb are shown in thin lines.

In general branching on variables does pretty well, but it is disastrous on difficult instances. Trying to perform better on these instances unfortunately has a price on the easier instances. But this is not specific to the TSP. In fact, NP-hardness means that there are very difficult instances which may be rare. One has at some point to decide how much one is willing to sacrifice on the solution time for the easy instances in order to perform not too badly on those rare instances.

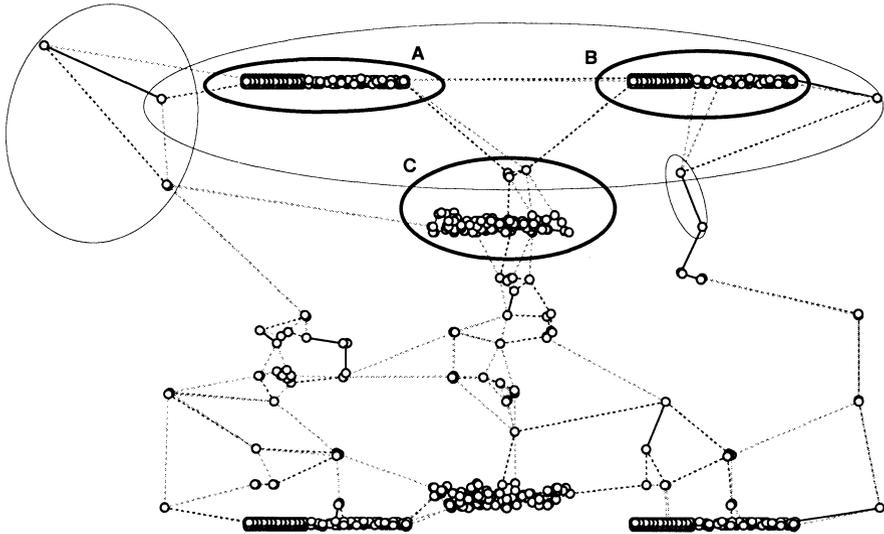


Figure 2.43. Illustration for the third branching rule

Applegate et al. introduced *strong* branching in an effort to optimize the probability of performing a good choice. It roughly goes as follows: choose a large enough set of candidates for branching (variables or inequalities). Test each of them by solving the corresponding lp relaxation only. The standard criterion is to choose the candidate for which the smallest of the two lower bounds of its sons, is the largest. Jünger et al. adapted this criterion a little by choosing among those with the same rounded smallest lower bound, the one with largest other lower bound, that is between a candidate yielding two bounds of 125.85 and 125.99 and one with 125.45 and 127.36, they choose the second. There is no reason, other than simplicity, to restrict to the rounded value; a better choice would be to choose among those which are within a certain interval, the one which has a larger upperbound. Computational experiments show that the time spent in choosing a good candidate definitively pays off, at least on difficult instances.

While performing this strong branching, one may encounter a bound which exceeds the upper bound. One has to take advantage of this by either setting the corresponding variable to the other value, or by adding the converse inequality to the lp.

We are left with the problem of choosing which open problem to solve next. It is well known in Branch-and-Bound that if the initial upper-bound equals the optimal value, whatever exploration strategy one uses, the same tree will be explored. This is no longer true in Branch-and-Cut. This is due to the fact that the inequalities one generates are globally valid, that is wherever they have been generated in the search tree, they are valid at any node of the search tree. The non-trivial inequalities, as already mentioned, are stored in a pool through which one searches regularly. The content of that pool depends on the previously visited nodes of the search tree, and since Branch-and-Cut is highly unrobust, the whole search may be considerably changed. Nevertheless it seems a good idea to go depth first if one believes that the current feasible solution is optimal. If not it seems that branching on the most promising open problem (the one with lowest lower bound) is the best choice.

20. Computational results

The two latest implementations of the Branch-and-Cut method for the TSP are that of Applegate, Bixby, Chvátal and Cook (ABCC) known as *CONCORDE*, and that of Jünger, Naddef, Reinelt and Thienel (JNRT). The latter relies on a general purpose Branch-and-Cut system known as *ABACUS* and uses the separation heuristics of Padberg and Rinaldi for subtours, and the ones of Naddef and Thienel for the other inequalities. The initial upperbounding is done using Andre Rohe's TSP implementation of Lin Kernighan. *ABACUS* is a system that manages all parts of Branch-and-Cut which are problem independent, making the development of optimization software based on Branch-and-Cut much easier since one may concentrate on problem-specific technicalities. Of course there is a price, which for the TSP ranges somewhere between 10% and 25% on CPU time.

Another main difference between these two codes is the separation strategies. JNRT only calls for separation of violated inequalities which are not subtour elimination inequalities when none of the latter are violated. They also restrict to separation into known classes of facet inducing inequalities. This is not the case for ABCC, as already seen.

Table 20 gives a comparison of times and number of nodes of the search tree between the two codes. The times reported below are for JNRT on a Sun Ultra-60 with a 295 MHz processor. For CONCORDE (ABCC) on

a 500 MHz Compaq XP1000 workstation. Following some benchmarking of ABCC, and due to the fact that JNRT use a general purpose Branch-and-Cut code, a factor of 4 or 5 seems to be reasonable to compare CPU times. These times, like all other figures should be taken with caution. Branch-and-Cut is a highly unrobust method: a slight change in some parameter setting may change considerably the number of nodes of the Branch-and-Cut tree and the CPU time. Times also fluctuate considerably depending on whether or not the upper-bound has been found early enough to save useless computations. For the JNRT code we also give, for some relevant instances, the results of runs in which the optimum value was entered as the initial upper bound. The interest of such data is to differentiate those instances for which the separation of valid inequalities is the difficulty from those for which it is the upperbounding. The data for the instance *fnl4461* for the ABCC code also corresponds to the initial upperbound set to the optimal value and for *pcb3038*, in the ABCC code the initial tour happens to be the optimal one. JNRT chose the option to privilege separation over branching, in the aim of testing how good the separation heuristics are. It may have been faster sometimes to branch much earlier.

A first conclusion is that separation using template paradigms does pretty well in most cases. The local cut separation procedures helps in some difficult instances. Another conclusion is that the size of a problem is not really what makes it difficult. Finally, if one looks into the recent history of TSP solving, one realizes that good separation is the first clue to success, the second being clever branching. The instance *ts225* was solved by Applegate et al. for the first time in over 5000 BC nodes in about two years of CPU time, then in a little over an hour and about 40 BC nodes by Naddef and Thienel using their separation routines, and now is solved in only a few seconds by CONCORDE. The same is illustrated by the instances *pcb3038* and *fnl4461* which were first solved in a few years of CPU time and now in less than one day of CPU. Table 2.2 gives the results of CONCORDE on very large instances. The parameter settings are in general different from the ones used for the previous table. Sometimes the optimal tour value was an input. See the web page of CONCORDE for more information.

21. Conclusion

In this chapter we have seen an example of the efforts that must be carried out in order to achieve good performance in the search for optimal solutions of large instances of the symmetric TSP via Branch-and-Cut. This effort is two-fold. First a deep theoretical research in

Name	BC nodes			time in mn:ss		
	JNRTopt	JNRT	ABCC	JNRTopt	JNRT	ABCC
pr76	-	1	1	-	0:35	0:02
gr120	-	1	1	-	0:01	0:02
pr136	-	1	1	-	0:02	0:04
pr144	-	1	1	-	0:01	0:02
pr152	-	1	1	-	0:02	0:08
u159	-	1	1	-	0:01	0:01
rat195	-	1	1	-	1:41	0:22
d198	-	3	3	-	0:20	0:12
kroA200	-	1	1	-	0:18	0:07
kroB200	-	1	1	-	0:11	0:04
gr202	-	1	1	-	0:09	0:05
ts225	-	57	1	-	86:26	0:21
pr226	-	1	1	-	0:03	0:04
gr229	-	3	3	-	3:31	0:39
gil262	-	1	1	-	0:43	0:13
pr264	-	1	1	-	0:03	0:03
pr299	-	1	3	-	1:06	0:17
lin318	-	1	1	-	0:40	0:10
rd400	13	15	15	2:39	4:38	2:28
fl417	-	1	5	-	0:55	0:58
gr431	9	9	13	7:02	10:40	2:13
pr439	-	5	15	-	8:39	3:36
pcb442	-	9	9	-	1:33	0:50
d493	-	1	5	-	10:29	1:53
att532	5	5	7	6:23	9:47	1:50
ali535	-	1	3	-	2:18	0:53
pa561	7	9	17	27:16	43:20	4:07
u574	-	1	1	-	0:35	0:23
rat575	7	11	25	4:47	20:38	6:03
p654	-	1	3	-	0:15	0:15
gr666	-	1	3	-	8:18	0:50
u724	9	9	11	10:05	20:45	3:45
rat783	-	1	1	-	2:13	0:38
dsj1000	-	1	7	-	75:27	6:50
pr1002	-	1	1	-	0:59	0:34
u1060	7	29	21	9:11	55:09	9:31
vm1084	7	7	11	66:25	81:21	10:04
pcb1173	13	25	19	21:49	77:23	7:48
rl1304	-	1	1	-	12:01	3:09
rl1323	33	55	25	327:35	407:01	62:20
nrv1379	-	7	19	-	29:09	9:38
d1655	-	1	5	-	25:33	4:23
vm1748	9	19	17	155:56	318:57	37:03
pr2392	-	1	1	-	7:44	1:47
pcb3038	265	-	313	5771mn	-	1347mn
fnl4461	409	-	213	9410mn	-	890mn

Table 2.2. Computational results

Name	BC nodes	time
rl5915	161	644h 20mn
rl5934	205	163h 35mn
pla7397	101	119h 10mn
rl11849	431	\simeq 155 days
usa13509	9539	\simeq 4 years
d15112	164569	\simeq 22.6 years

Table 2.3. Very large instances with Concorde

order to study the convex hull of the solutions. Note that even if one uses separation routine outside the template paradigm, this study is necessary, since some steps of the algorithm need these theoretical results to justify them. Following that theoretical study, one must invest in a relevant algorithmic and implementation effort. The implementation effort is unfortunately now far too high for a newcomer. If this very exciting field is not to die, the only hope is that large pieces of existing source code be made available to anybody interested in developing a new challenging program. We hope that this will be the case in a near future.

Chapter 3

POLYHEDRAL THEORY FOR THE ASYMMETRIC TRAVELING SALESMAN PROBLEM

Egon Balas

*Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, PA 15213, USA
eb17+@andrew.cmu.edu*

Matteo Fischetti

*D.E.I, University of Padova
Via Gradenigo, 6/A
35131 Padova, Italy
fisch@dei.unipd.it*

1. Introduction

The application of polyhedral methods to the TSP started in the mid-1970's (see [396] and [402] for the first major breakthroughs). For more than a decade, until the late 1980's, the main emphasis was on the symmetric TSP (STSP). There were several reasons for this: the traveling salesman paradigm suggests a geometric interpretation in which the costs are symmetric; some of the important real world applications, like in chip manufacturing, are symmetric; the polyhedral formulation of the symmetric TSP connects nicely to matching theory and borrows from the latter the family of facet defining 2-matching inequalities; finally, the asymmetric TSP (ATSP) can be reduced to a STSP on an undirected graph with twice as many nodes.

Nevertheless, there are good reasons to study the asymmetric TSP on its own. First, the asymmetric TSP is the more general one, which subsumes the symmetric TSP as a special case.

Second, the structure of the ATS polytope – i.e. the convex hull of incidence vectors of TS tours in a digraph – is much richer than that of the STS polytope. Every facet F of the STS polytope defined on a complete undirected graph G corresponds (trivially) to a face F' of the ATS polytope defined on the directed graph G' obtained from G by replacing every edge with a pair of antiparallel arcs. But the symmetric inequalities obtained this way define only a tiny fraction of the multitude of facets of the ATS polytope, the vast majority of the latter being defined by asymmetric inequalities that have no counterpart for the STS polytope defined on G .

Third, there are many important real-world problems that are naturally modeled as asymmetric TSP's. In industrial scheduling, the optimal sequencing of jobs on machines with setup times is an ATSP; more generally, the optimal ordering of any set of tasks or operations with sequence dependent changeover costs is an ATSP or one of its generalizations.

Fourth, the study of the ATS polyhedron provides certain insights into the structure of the STS polyhedron. While it is true that an asymmetric facet inducing inequality $\alpha'x \leq \alpha_0$ for the ATS polytope defined on the digraph G' typically has no counterpart for the STS polytope defined on G , it is also true that such an inequality gives rise to a family of facet inducing inequalities for a symmetric TS polytope defined on another, larger undirected graph, as will be discussed later in this survey.

Finally, while it is true that an asymmetric TSP can be reduced to a symmetric one, this symmetric TSP has a very special structure and is defined on an undirected graph with twice as many nodes as the digraph of the ATSP, and so this transformation is not without a price.

We next introduce the main notation used in the sequel.

Let $G = (V, A)$ be the complete (loop-free) digraph on n nodes, with $n \geq 5$. We associate a variable x_{ij} with every arc $(i, j) \in A$. For $S, T \subseteq V$, we denote $x(S, T) := \sum(x_{ij} : i \in S, j \in T, (i, j) \in A)$ and write $x(i, S)$ and $x(S, j)$ for $x(\{i\}, S)$ and $x(S, \{j\})$ respectively. Further, for any $S \subseteq V$ we denote by $\delta^+(S)$ the set of arcs with their tail in S and their head in $V \setminus S$, and by $\delta^-(S)$ the set of arcs with their tail in $V \setminus S$ and their head in S . Also, we write $\delta^+(v), \delta^-(v)$ for $\delta^+(\{v\}), \delta^-(\{v\})$, respectively. Finally, we denote $\delta(S) := \delta^+(S) \cup \delta^-(S)$.

When G has costs (usually restricted to nonnegative values) on its arcs, the traveling salesman problem defined on G is the problem of finding a minimum-cost directed Hamiltonian cycle in G . It is also called the *Asymmetric Traveling Salesman Problem* (ATSP), to distinguish it from its *symmetric* counterpart (STSP) defined on an undirected graph.

There are several known formulations of the ATSP. We will use the standard one, due to Dantzig, Fulkerson and Johnson [239]:

minimize cx

s.t.

$$x(\delta^+(v)) = 1 \qquad v \in V \qquad (1)$$

$$x(\delta^-(v)) = 1 \qquad v \in V \qquad (2)$$

$$x(S, S) \leq |S| - 1 \qquad S \subset V, \quad |S| \geq 2 \qquad (3)$$

$$x \in \{0, 1\}^A \qquad (4)$$

Here $c = (c_{ij})$ is the vector of costs, the equations (1) and (2) are the *outdegree* and *indegree* constraints, respectively (briefly, the degree constraints), while (3) are the *subtour elimination constraints* (SEC, for short). This formulation has $n^2 - n$ variables and $O(2^n)$ constraints. There are more compact formulations, but this one has proved so far the most useful.

The ATS polytope is then the convex hull of points satisfying (1)-(4).

The *monotone ATS polytope* \tilde{P} is obtained from P by replacing $=$ with \leq in all the degree constraints. It is well known [402] that P has dimension $n(n-1) - 2n + 1$ whereas \tilde{P} is of full dimension.

Whenever there is a need to specify the graph G on which P or \tilde{P} is defined, we will write $P(G)$ and $\tilde{P}(G)$, respectively. Moreover, we sometimes use the notation P_n for P defined on the complete digraph with n nodes.

The polyhedral approach to combinatorial optimization consists in trying to describe the solution set to the problem studied through a system of linear inequalities (and possibly equations) that define its convex hull. Of particular importance in this attempt is the identification of inequalities that define facets of the convex hull, because these form a minimal system. If the attempt is successful in that all the facets of the convex hull are identified, then the problem becomes a linear program. If the attempt is only partly successful, in that it fails to completely characterize the convex hull but it succeeds in describing large families of facets of the latter, then the partial description of the convex hull obtained this way usually makes it much easier to solve the problem by enumerative methods like branch and bound, branch and cut, etc.

While a complete characterization of the ATS polytope – i.e. a listing of all the inequalities defining it – is only available for $n \leq 6$ [283], several families of valid inequalities have been identified recently, and

many of them have been shown to be facet defining for the corresponding polytope. In this survey we focus on ATSP-specific results (asymmetric inequalities, lifting theorems, etc.), and refer the reader to Chapter 2 of the present book for a thorough description of the properties shared with the STSP. Also, the separation problem for ATSP inequalities is not addressed in the present chapter: the interested reader is referred to Chapter 2 for the separation of symmetric inequalities, and to Chapter 4 and to [75] and [161] for ATSP-specific separation procedures.

Finally, we assume the reader is familiar with the fundamentals of polyhedral theory, as given e.g. in the excellent survey of polyhedral theory for the TSP by Grötschel and Padberg [405].

The present chapter is organized as follows. In Section 2 we review the main classes of inequalities for the ATS polytope (for short: ATS inequalities) known at the time of the writing of [405]. Section 3 addresses the monotone ATS polytope, a widely used relaxation of the ATS polytope. Facet-lifting procedures are analyzed in Section 4. The important question of whether two different facet defining inequalities define the same facet of P , is addressed in Section 5. Sections 6 and 7 are devoted to the study of large classes of ATS-specific facet-defining inequalities introduced recently, namely the odd CAT and the SD inequalities, respectively. Lifted cycle inequalities are finally investigated in Section 8, where a characterization of this class is provided, and some specific subclasses with interesting properties are analyzed.

2. Basic ATS inequalities

Several classes of valid inequalities for both P and \tilde{P} were known at the time of the writing of the Grötschel and Padberg survey [405]. Among them, we will outline next the so-called comb, clique tree, D_k^+ , D_k^- , T_k , C2 and C3 inequalities.

2.1. Symmetric Inequalities

Since the STSP is a special case of the ATSP, any given inequality $\bar{\alpha}y := \sum_{e \in E} \bar{\alpha}_e y_e \leq \alpha_0$ defined for the STS polytope associated with a complete undirected graph $G_E = (V, E)$, has an obvious ATS counterpart $\alpha x := \sum_{(i,j) \in A} \alpha_{ij} x_i \leq \alpha_0$ obtained by defining $\alpha_{ij} := \alpha_{ji} := \bar{\alpha}_e$ for each $e = [i, j] \in E$. Conversely, every ATS inequality $\alpha x \leq \alpha_0$ which is *symmetric* (in the sense that $\alpha_{ij} = \alpha_{ji}$ for all $(i, j) \in A$, $i < j$) has an STS counterpart $\bar{\alpha}y := \sum_{e \in E} \bar{\alpha}_e y_e \leq \alpha_0$ obtained by setting $\bar{\alpha}_e := \alpha_{ij}$ ($= \alpha_{ji}$) for each $e = [i, j] \in E$. It can easily be shown that the above mapping between STS and symmetric ATS inequalities preserves validity (but not necessarily the facet-defining property, as discussed later in

this subsection). As a consequence, the ATS polytope inherits from its undirected version all classes of valid STS inequalities.

Note however that symmetric inequalities are just a small fraction of the whole set of (facet-defining) ATS inequalities. For example, Euler and Le Verge [283] showed that only 2 out of the 287 classes of inequalities representing a complete and irredundant description of the ATS polytope on 6 nodes, are symmetric (namely, the classes of SEC's (3) with $|S| = 2, 3$). This implies that only 35 out of the 319,015 facets of P_6 are defined by symmetric inequalities.

We next present the two most basic families of symmetric inequalities, the comb and clique-tree inequalities, and refer the reader to Chapter 2 of the present book for other large classes of STS inequalities.

Proposition 1 ([403, 404]) *Let $H \subset V$ be a “handle” and $T_1, \dots, T_s \subset V$ be pairwise disjoint “teeth” satisfying: (i) $|H \cap T_i| \geq 1$ and $|T_i \setminus H| \geq 1$ for all $i = 1, \dots, s$, and (ii) $s \geq 3$ and odd. The following comb inequality is valid for both P and \tilde{P} :*

$$x(H, H) + \sum_{i=1}^s x(T_i, T_i) \leq |H| + \sum_{i=1}^s (|T_i| - 1) - (s + 1)/2.$$

Theorem 2 ([295]) *Comb inequalities are facet inducing for P_n for $n \geq 7$, and for \tilde{P}_n for $n \geq 6$.*

Note that, unlike in the symmetric case, a comb inequality does not define a facet of P_n when $n = 6$, as in this case the corresponding face turns out to be the intersection of two distinct facets of P_n defined by asymmetric inequalities having no counterpart in the symmetric case – a pathological situation that will be discussed later in this subsection.

Proposition 3 ([406]) *Let $C := \{H_1, \dots, H_r, T_1, \dots, T_s\}$, with $s \geq 1$ and odd, be a family of nonempty subsets of V where the H_i ($i = 1, \dots, r$) are called handles and the T_j ($j = 1, \dots, s$) are called teeth, which satisfy (see Figure 3.1 for an illustration): (i) $T_i \cap T_j = \emptyset$, for each $i, j \in \{1, \dots, s\}$, $i \neq j$; (ii) $H_i \cap H_j = \emptyset$, for each $i, j \in \{1, \dots, r\}$, $i \neq j$; (iii) $2 \leq |T_j| \leq n - 2$ and $T_j \setminus (\bigcup_{i=1}^r H_i) \neq \emptyset$, for $j = 1, \dots, s$; (iv) the number, h_i , of teeth overlapping H_i is odd and at least 3, for $i = 1, \dots, r$; (v) the intersection graph of C (i.e., the undirected graph having one node for each subset belonging to C and one edge for each pair of overlapping subsets) is a tree.*

Then the following clique tree inequality is valid for both P and \tilde{P} :

$$\sum_{i=1}^r x(H_i, H_i) + \sum_{j=1}^s x(T_j, T_j) \leq s(C),$$

where t_j denotes the number of handles intersecting tooth T_j ($j = 1, \dots, s$), and $s(C) := \sum_{i=1}^r |H_i| + \sum_{j=1}^s (|T_j| - t_j) - (s + 1)/2$ is the size of C .

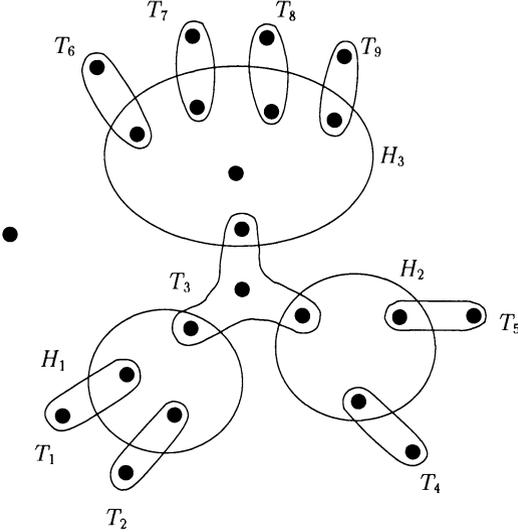


Figure 3.1. A clique tree with $r = 3$ handles and $s = 9$ teeth

Observe that clique tree inequalities with just one handle coincide with the comb inequalities introduced previously.

Clique tree inequalities have been proved by Grötschel and Pulleyblank [406] to be valid and facet-inducing (in their undirected form) for the STS counterparts, Q and \tilde{Q} , of the ATS polytopes P and \tilde{P} . As already observed, this implies that they are valid (but not necessarily facet-defining) for P and \tilde{P} . Later, Fischetti [297] proved the following:

Theorem 4 All clique tree inequalities (except for combs when $n = 6$) define facets of both P and \tilde{P} .

We next address the correspondence between facets of the STS and ATS polytopes. It was observed in [295] that the STS counterpart of every symmetric ATS inequality, say $\alpha x \leq \alpha_0$, defining a facet of P (resp., \tilde{P}) necessarily defines a facet of Q (resp., \tilde{Q}). Indeed, consider the polytope P (our reasoning trivially applies to \tilde{P} as well). Let $\bar{\alpha}y \leq \alpha_0$ be the undirected counterpart of $\alpha x \leq \alpha_0$. This inequality clearly defines a proper face of Q . Assume now that it is not facet defining for Q . Then there must exist a facet defining inequality $\bar{\beta}y \leq \beta_0$, valid for Q and satisfying $\{y \in Q : \bar{\alpha}y = \alpha_0\} \subset \{y \in Q : \bar{\beta}y = \beta_0\} \subset Q$.

But this implies that the directed counterpart $\beta x \leq \beta_0$ of $\bar{\beta}y \leq \beta_0$ is valid for P and induces a proper face which strictly contains the face $\{x \in P : \alpha x = \alpha_0\}$, impossible since $\alpha x \leq \alpha_0$ is facet-defining by assumption.

An important question raised in [405] is whether or not the directed version of a facet-defining inequality for the STS polytope is also facet defining for the ATS polytope. This issue has been investigated by Queyranne and Wang [689], who showed that several known classes of facet-defining inequalities for the STS polytope, including the path, wheelbarrow, chain, and ladder inequalities (see Chapter 2 on these inequalities), do define facets of P (with the exception of some pathological cases). Moreover, [689] proposed an operation called tree-composition and used it to produce a new large class of symmetric inequalities that define facets of the ATS (and hence of the STS) polytope.

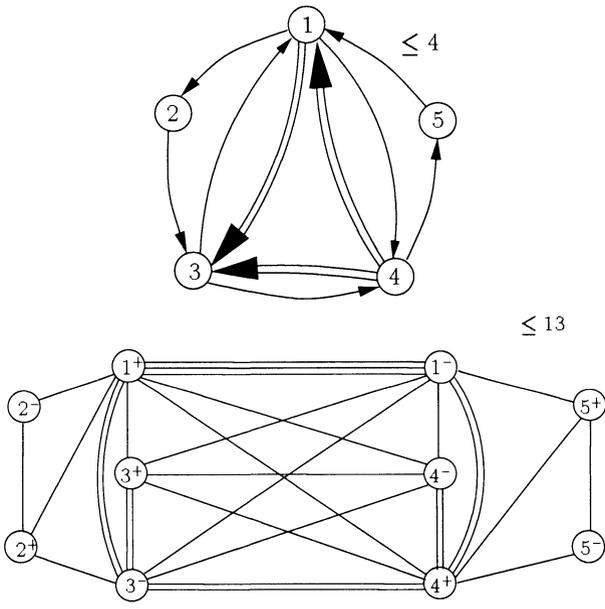


Figure 3.2. Curtin inequality for the ATSP and corresponding inequality for the STSP.

A more intriguing correspondence between STS and ATS polytopes (defined on graphs with a different number of nodes) derives from the known fact [495] that every ATSP defined on the complete digraph $G = (V, A)$ can be restated as a STSP defined on an undirected bipartite graph $G_B^* := (V^*, E_B^*)$, where V^* has a pair of nodes i^+, i^- for

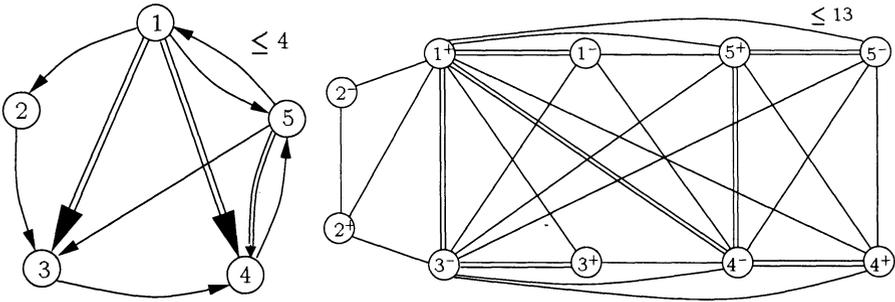


Figure 3.3. Fork inequality for the ATSP and corresponding inequality for the STSP.

every node $i \in V$, and E_B^* has an edge (i^+, j^-) for every arc $(i, j) \in A$, and an edge (i^+, i^-) for every node $i \in V$, with the condition that the only admissible tours in G_B^* are those that contain every edge (i^+, i^-) , $i = 1, \dots, n$. (Here the subscript B stands for “bipartite.”) This transformation has been used in [474] for solving hard instances of the ATSP by means of efficient computer codes designed for the STSP. (See Chapter 4 for a computational assessment of this approach). However, the same transformation has important polyhedral implications; namely, it can be used to find new facets of the STS polytope, as shown in [75].

To this end, denote by $G^* := (V^*, E^*)$ the complete graph on V^* . Clearly, $E_B^* \subset E^*$ and G_B^* is a proper subgraph of G^* , defined on the same node set V^* . Also, denote by $Q(G^*)$ and $Q(G_B^*)$ the STS polytopes defined on G^* and G_B^* , respectively. To the incidence vector $x \in \{0, 1\}^A$ of any tour in G , we associate the incidence vector $y \in \{0, 1\}^{E^*}$ of a tour in G^* , defined by $y_{i+j^-} = x_{ij}$ for all $(i, j) \in A$, and

$$\begin{aligned} y_{i+i^-} &= 1 && \text{for all } i \in V, \\ y_{i+j^+} &= y_{i-j^-} = 0 && \text{for all } i, j = 1, \dots, n, \ i \neq j. \end{aligned} \tag{5}$$

The above construction then induces a 1-1 correspondence between tours in G , i.e. vertices of the ATS polytope $P(G)$, and admissible tours in G_B^* , i.e. vertices of the STS polytope $Q(G_B^*)$. Note that $Q(G_B^*)$ is the face of $Q(G^*)$ defined by the equations (5). This implies that to every facet inducing inequality $\alpha x \leq \alpha_0$ for the ATS polytope $P(G)$ there corresponds a facet inducing inequality $\beta y \leq \alpha_0$ for the STS polytope $Q(G_B^*)$, where $\beta \in \mathbb{R}^{E_B^*}$ is defined by $\beta_{i+j^-} = \alpha_{ij}$ for all $(i, j) \in A$ and $\beta_{i+i^-} = 0$ for all $i \in V$.

Next, the inequality $\beta y \leq \alpha_0$ can be lifted into one or more facet inducing inequalities $\beta y + \gamma y' \leq \beta_0$, where y' is the vector whose components are associated with the edges of $E^* \setminus E_B^*$, and γ is the vector of

lifted coefficients. This can be done, for instance, by sequential lifting [640]. The latter consists in choosing an appropriate sequence for the variables fixed at 1 or 0, freeing them one by one, and calculating the corresponding lifting coefficients. Here β_0 typically differs from α_0 , because when the coefficient of a variable previously fixed at one is lifted, the right hand side is also changed.

The inequality $\beta y + \gamma y' \leq \beta_0$ resulting from the lifting is facet inducing for the STS polytope $Q(G^*)$ on the complete undirected graph G^* , and is often of a type unknown before. In other words, the study of the polyhedral structure of the ATS polytope provides new insights also into the structure of its symmetric counterpart.

Figures 3.2 and 3.3 show two known inequalities for the ATS polytope (the so-called *curtain* and *fork* inequalities defined in Section 8) and their counterparts for the STS polytope. The arcs in single and in double line have coefficient 1 and 2, respectively; all remaining arcs have coefficient 0. (This rule applies to all subsequent figures too.)

2.2. Asymmetric inequalities

We next introduce the main classes of asymmetric ATS inequalities known prior to the publication of [405].

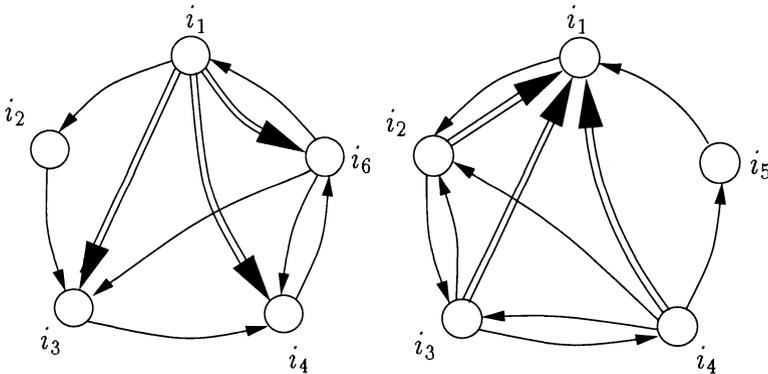


Figure 3.4. The support graph of a D_k^+ (left) and of a D_k^- (right) inequality when $k = 5$.

Proposition 5 ([396, 402]) Let $\{i_1, i_2, \dots, i_k\} \subset V$, $3 \leq k \leq n - 1$; then the D_k^+ inequality

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2x(\{i_1\}, \{i_3, \dots, i_k\}) + \sum_{j=4}^k x(\{i_j\}, \{i_3, \dots, i_{j-1}\}) \leq k-1$$

and the D_k^- inequality

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2x(\{i_2, \dots, i_{k-1}\}, \{i_1\}) + \sum_{j=3}^{k-1} x(\{i_j\}, \{i_2, \dots, i_{j-1}\}) \leq k - 1$$

are valid for both P and \tilde{P} (see Figure 3.4 for an illustration).

Notice that D_k^- inequalities can be obtained from D_k^+ inequalities by switching the coefficients of each pair of arcs (i, j) and (j, i) – and by reversing the order of the nodes i_1, i_2, \dots, i_k .

Proposition 6 ([396, 402]) *Let $S \subset V$ with $2 \leq k := |S| \leq n - 2$, and let $w \in S$, $p, q \in V \setminus S$, and $p \neq q$. Then the following T_k inequality is valid for both P and \tilde{P} (see Figure 3.5):*

$$x(S, S) + x_{pw} + x_{wq} + x_{pq} \leq k.$$

Furthermore, T_k inequalities define facets of P if and only if $k \neq n - 2$, and of \tilde{P} for any k .

T_k inequalities can be generalized by attaching a source p and a sink q to a comb, as follows.

Proposition 7 ([396, 402]) *Let $H \subset V$ be a “handle” and $T_1, \dots, T_s \subset V$ be pairwise disjoint “teeth” satisfying: (i) $|H \cap T_i| \geq 1$ and $|T_i \setminus H| \geq 1$ for all $i = 1, \dots, s$, and (ii) $s \geq 3$ and odd. For each pair of distinct vertices p and q in $(V \setminus H) \setminus (\bigcup_{i=1}^s T_i)$, the following C2 inequality is valid for P and \tilde{P} (see Figure 3.6):*

$$x(H, H) + \sum_{i=1}^s x(T_i, T_i) + \sum_{v \in H} (x_{pv} + x_{vq}) + x_{pq} \leq s(C) + 1,$$

where $s(C) = |H| + \sum_{i=1}^s (|T_i| - 1) - (s + 1)/2$.

Proposition 8 ([396, 402]) *Let i_1, i_2 , and i_3 be three different vertices, and let $W_1, W_2 \subset V$ such that: (i) $W_1 \cap W_2 = \emptyset$, and (ii) $W_j \cap \{i_1, i_2, i_3\} = \{i_j\}$ and $|W_j| \geq 2$ for $j = 1, 2$. Then the following C3 inequality is valid for both P and \tilde{P} (see Figure 3.7):*

$$x(W_1, W_1) + x(W_2, W_2) + x(\{i_1\}, W_2) + x_{i_2 i_1} + x_{i_3 i_1} + x_{i_3 i_2} \leq |W_1| + |W_2| - 1.$$

Theorem 9 [295] *All D_k^+ , D_k^- , C2, and C3 inequalities define facets of both P and \tilde{P} .*

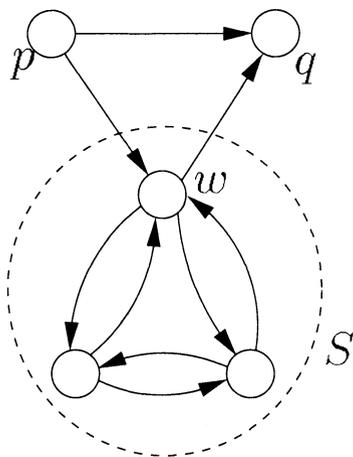


Figure 3.5. The support graph of a T_k inequality ($k = 3$).

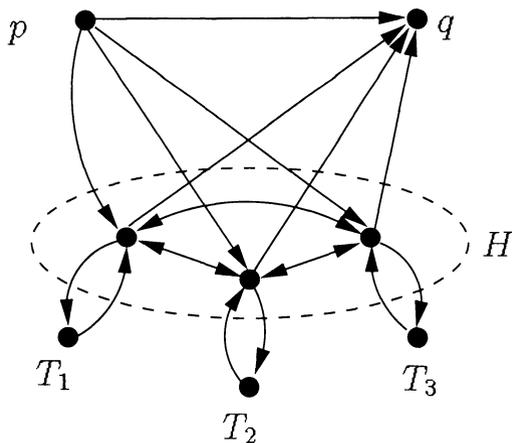


Figure 3.6. The support graph of a C2 inequality.

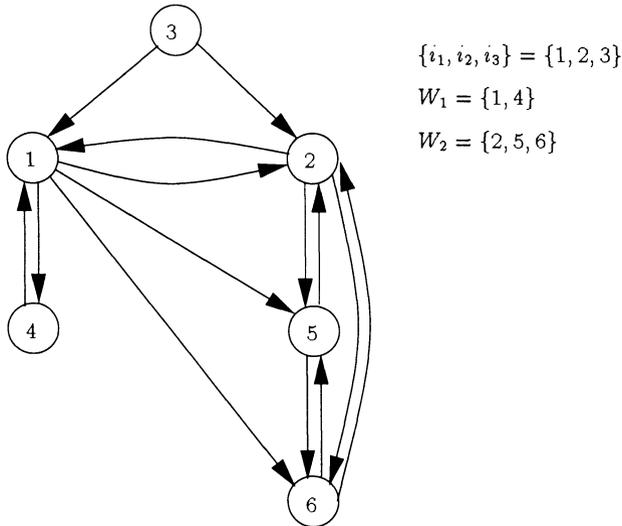


Figure 3.7. The support graph of a C3 inequality.

Other classes of relevant ATS inequalities have been described in [396, 402], including the lifted cycle inequalities discussed in greater detail in the forthcoming Section 8. Finally, Grötschel and Wakabayashi [407, 408] described “bad” facets of \tilde{P} related to hypo-Hamiltonian and hypo-semi-Hamiltonian subgraphs of G .

3. The monotone ATS polytope

Much of polyhedral theory deals with less than full dimensional polyhedra. The hallmark of such a polyhedron, say P , is the existence of certain equations satisfied by every point of P . Because of this, every valid inequality for P can be expressed in several forms, the equivalence of which is not always easy to recognize. This may cause technical difficulties when analyzing the facial structure of P . Moreover, polyhedral proof techniques often rely on interchange arguments, involving extreme points that differ only in a few components: the smaller the number of such components, the simpler the argument. Often the presence of equations in the defining system of P implies that this number cannot be very small. In particular, in the case of the ATS polytope any two distinct extreme points differ by at least six components, since at least three arcs have to be deleted from a tour, and three others added to it, in order to obtain a different tour.

A widely used technique to overcome these difficulties is to enlarge $P \subset \mathbb{R}^A$ so as to make it full dimensional, i.e. to obtain a polyhedron \tilde{P} of dimension $|A|$. When P lies in the positive orthant, a standard way of doing this is to “enlarge P below” to obtain the (*downward*) *monotonization*, or *submissive*, of P , defined as

$$\tilde{P} := \{y \in \mathbb{R}^A : 0 \leq y \leq x \text{ for some } x \in P\} = (P - \mathbb{R}_+^A) \cap \mathbb{R}_+^A.$$

Under appropriate conditions, optimizing a linear function over P is equivalent to optimizing a related linear function over its monotone \tilde{P} , so \tilde{P} can replace P insofar as optimization is concerned. Hence the interest in the study of the polyhedral structure of \tilde{P} .

It is easy to see that \tilde{P} is full dimensional if and only if for each $a \in A$, there exists $x \in P$ with $x_a \neq 0$. When $P = \text{conv}\{x \in \mathbb{Z}_+^A : A_1x = b_1, A_2x \leq b_2\}$, \tilde{P} can in some cases be obtained by replacing $A_1x = b_1$ by $A_1x \leq b_1$. This is true, for instance, in the case of the symmetric and asymmetric TS polytopes defined on a complete graph or digraph, when $A_1x = b_1$ consists of degree equations. In this situation the monotone polytope is the convex hull of incidence vectors of path systems, i.e. families of simple node-disjoint paths (possibly including isolated nodes). But it is not true in general in the case of the same polytopes defined on graphs other than complete: while the monotone polytope contains the incidence vectors of all subsets of tours, the polytope obtained by replacing $=$ with \leq may contain incidence vectors of arc sets that are not part of any tour.

Other useful relaxations (not covered in the present chapter) are the *Fixed-Outdegree 1-Arborescence polytope* studied in [72] (where the out-degree equations (1) are only imposed for a given subset F of nodes), the *Asymmetric Assignment polytope* addressed in [62] (where all SEC's (3) with $|S| \geq 3$ are relaxed), and the *Graphical Asymmetric Traveling Salesman polyhedron* of Chopra and Rinaldi [183] (where the degree conditions (1)-(4) are relaxed into $x(\delta^+(v)) = x(\delta^-(v))$ for all $v \in V$, and the SEC's (0.2) are replaced by their cut form $x(\delta^+(S)) \geq 1$ for all $\emptyset \subset S \subset V$).

We analyze next the main properties of general monotone polyhedra and their implications for the study of the ATS polytope. The reader is referred to [74] for more details and for the proof of the main propositions, as well as for a parallel treatment of the monotone STS polytope.

3.1. Monotonizations of polyhedra

Consider next an arbitrary polyhedron P contained in \mathbb{R}^N , where $N = \{1, \dots, n\}$, and assume that P is given in the form $P := \{x \in \mathbb{R}^N : Ax \leq b\}$. Although we make no assumption on the dimensionality of P ,

the construction to follow becomes meaningful only when P is less than full dimensional. Let $A^=x = b^=$ be a full row rank *equality system* for P (i.e., every equation satisfied by all points of P is a linear combination of $A^=x = b^=$), having $r := n - \dim(P)$ rows.

We introduce a generalized concept of monotonization of a polyhedron. For any partition $[N^L, N^U]$ of N , let $b_j \in \mathbb{R} \cup \{-\infty\}$, $j \in N^L$, and $b_j \in \mathbb{R} \cup \{+\infty\}$, $j \in N^U$, be such that $x_j \geq b_j$ for $j \in N^L$ and $x_j \leq b_j$ for $j \in N^U$ for every $x \in P$. Then the *g-monotonization* (g for generalized) of P is defined as

$$\text{g-mon}(P) := \left\{ y \in \mathbb{R}^N \mid \begin{array}{l} b_j \leq y_j \leq x_j, \quad j \in N^L \text{ and} \\ x_j \leq y_j \leq b_j, \quad j \in N^U \text{ for some } x \in P \end{array} \right\}.$$

Notice that when $P \subseteq \mathbb{R}_+^N$, if $N^U = \emptyset$ and $b_j = 0$, $j \in N^L = N$, then $\text{g-mon}(P) = \{y \in \mathbb{R}^N : 0 \leq y_j \leq x_j, j \in N, \text{ for some } x \in P\}$ coincides with the *submissive* of P . Moreover, if $N^L = \emptyset$ and $b_j = +\infty$, $j \in N^U = N$, then $\text{g-mon}(P) = \{y \in \mathbb{R}^N : y_j \geq x_j, j \in N^U, \text{ for some } x \in P\}$ defines the so-called *dominant* of P .

Next we address the dimension of $\text{g-mon}(P)$.

Proposition 10 *Let $W := \{j \in N : |b_j| < \infty, x_j = b_j \text{ for all } x \in \text{g-mon}(P)\}$. Then $\dim(\text{g-mon}(P)) = n - |W|$.*

For the rest of this subsection we will assume that $W = \emptyset$, i.e. $\text{g-mon}(P)$ is full dimensional.

Proposition 11 *The inequalities (i) $x_j \geq b_j$ for $j \in N^L$ such that $|b_j| \neq \infty$, and (ii) $x_j \leq b_j$ for $j \in N^U$ such that $|b_j| \neq \infty$ define facets of $\text{g-mon}(P)$.*

We will call *trivial* the facets of $\text{g-mon}(P)$ induced by $x_j \geq b_j$ or $x_j \leq b_j$. By extension, we will also call trivial every nonempty face of $\text{g-mon}(P)$, of whatever dimension, contained in a trivial facet.

Proposition 12 *Let $\alpha x \leq \alpha_0$ define a nontrivial face of $\text{g-mon}(P)$. Then $\alpha_j \geq 0$ for all $j \in N^L$ and $\alpha_j \leq 0$ for all $j \in N^U$.*

Definition 13 ([406]) *For a given valid inequality $\alpha x \leq \alpha_0$ defining a nontrivial face of $\text{g-mon}(P)$, let $N^0 := \{j \in N : \alpha_j = 0\}$ and let $A_0^=$ denote the $r \times |N^0|$ submatrix of $A^=$ whose columns are indexed by N^0 . The inequality $\alpha x \leq \alpha_0$ is said to be support reduced (with respect to P) if $A_0^=$ has full row rank.*

It can be shown that, if $\alpha x \leq \alpha_0$ defines a nontrivial facet of $\text{g-mon}(P)$, then either $\alpha x \leq \alpha_0$ is support reduced, or else it is a linear combination of equations satisfied by all points in P (i.e., there exists

$\mu \in \mathbb{R}^r \setminus \{0\}$ such that $(\alpha, \alpha_0) = \mu(A^\equiv, b^\equiv)$. This property leads to the following result.

Theorem 14 *Let $\alpha x \leq \alpha_0$ be a valid inequality for $g\text{-mon}(P)$, that defines a nontrivial facet of P . Then $\alpha x \leq \alpha_0$ defines a facet of $g\text{-mon}(P)$ if and only if it is support reduced.*

We next address the important question of whether a facet defining inequality for $g\text{-mon}(P)$ is also facet defining for P .

Definition 15 ([64, 74]) *Let $\alpha x \leq \alpha_0$ be a support reduced inequality defining a nontrivial facet of $g\text{-mon}(P)$, and let $F := \{x \in P : \alpha x = \alpha_0\}$ denote the face of P induced by $\alpha x \leq \alpha_0$. Suppose w.l.o.g. that $N^0 := \{j \in N : \alpha_j = 0\} = \{j_1, \dots, j_q\}$, where $q \geq r$, and j_1, \dots, j_r index independent columns (i.e., a basis) of A_0^\equiv . Inequality $\alpha x \leq \alpha_0$ is said to be strongly support reduced with respect to P if for every $k \in \{r+1, \dots, q\}$, if any, there exists a pair $x^1, x^2 \in F$ such that $\{j_k\} \subseteq \Delta(x^1, x^2) \subseteq \{j_1, \dots, j_k\}$, where $\Delta(x^1, x^2) := \{j \in N : x_j^1 \neq x_j^2\}$.*

Theorem 16 *Let $\alpha x \leq \alpha_0$, where $\alpha_0 \neq 0$, define a nontrivial facet of $g\text{-mon}(P)$. If $\alpha x \leq \alpha_0$ is strongly support reduced with respect to P , then it also defines a facet of P .*

3.2. Properties of the monotone ATS polytope

We now turn to the ATS polytope, P , and examine its relation to its submissive \tilde{P} . Given the complete digraph $G = (V, A)$, we define a bipartite graph $B[G] := (V^+ \cup V^-, E)$ having two nodes, $v^+ \in V^+$ and $v^- \in V^-$, for every node v of G , and an edge $[i^+, j^-] \in E$ for every arc (i, j) of G . It is easy to see that the coefficient matrix of the degree equations coincides with the node-edge incidence matrix of $B[G]$, hence a subset \tilde{A} of the arcs of G indexes a basis of the equality system of P if and only if \tilde{A} induces a spanning tree of $B[G]$.

Note that two edges of $B[G]$ are adjacent if and only if the corresponding arcs of G have either the same tail or the same head. Thus two nodes of $B[G]$ are connected by a path if and only if the corresponding nodes of G are connected by an *alternating path*, i.e., by a path of alternating arc directions. Now let $\alpha x \leq \alpha_0$ define a nontrivial facet of \tilde{P} , and denote $A^0 := \{(i, j) \in A : \alpha_{ij} = 0\}$, $A^+ := A \setminus A^0$, $G^0 := (V, A^0)$, and $G^+ := (V, A^+)$.

Since connectedness is the necessary and sufficient condition for $B[G^0]$ to contain a spanning tree and thus for A^0 to contain a basis of A^\equiv , it follows that an inequality $\alpha x \leq \alpha_0$ is support reduced with respect to P if and only if the bipartite graph $B[G^0]$ is connected. Thus when P is the ATS polytope, Theorem 14 specializes to the following:

Theorem 17 *Let $\alpha x \leq \alpha_0$ be a nontrivial facet-defining inequality for P , that is valid for \tilde{P} . Then $\alpha x \leq \alpha_0$ defines a facet of \tilde{P} if and only if the bipartite graph $B[G^0]$ is connected.*

A relevant special case in which $B[G^0]$ is obviously connected arises when

$$G^+ \text{ has an isolated node } h, \text{ and } G^0 \text{ has an arc } a^* \notin \delta(h) \quad (6)$$

as in this case the $2n - 1$ edges of $B[G^0]$ corresponding to the arcs in $\delta(h) \cup \{a^*\}$ induce a spanning tree of $B[G^0]$. Condition (6) is satisfied by many nontrivial facet inducing inequalities for the submissive of P , and is related to the concept of h -canonical form discussed in Section 5.

Suppose now that condition (6) is satisfied, guaranteeing that $\alpha x \leq \alpha_0$ is support reduced. We next give a sufficient condition for $\alpha x \leq \alpha_0$ to be strongly support reduced.

Let $G_h := G - \{h\}$, where h is the isolated node in (6). Two arcs of G_h are called α -adjacent (in G_h) if they are contained in a tour T of G_h such that $\alpha(T) = \alpha_0$. We define the (undirected) α -adjacency graph $G_h^* := (V^*, E^*)$ of G_h , as having a node for every arc in $A^0 \setminus \delta(h)$, and an edge for every pair $a, b \in V^*$ such that arcs a and b are α -adjacent in G_h . It can be shown that, under the assumption (6), a sufficient condition for $\alpha x \leq \alpha_0$ to be strongly support reduced is that graph G_h^* be connected. This leads to the following characterization that is trivial to verify without constructing G_h^* explicitly and holds in most of the relevant cases.

Theorem 18 *Let $\alpha x \leq \alpha_0$ define a nontrivial facet of \tilde{P} and a proper face of P . Assume G^+ has two isolated nodes, say h and k . If the bipartite graph $B[G^0 - \{h, k\}]$ is connected, then $\alpha x \leq \alpha_0$ is strongly support reduced, hence it defines a facet of P .*

Corollary 19 *Let $\alpha x \leq \alpha_0$ define a nontrivial facet of \tilde{P} and a proper face of P . If G^+ has three isolated nodes, then $\alpha x \leq \alpha_0$ defines a facet of P .*

Similar results apply to the STS polytope as well [74]. This gives an unexpectedly simple answer to an open problem posed by Grötschel and Padberg:

“Research problem. Find (reasonable) sufficient conditions which imply that an inequality defining a facet for \tilde{Q}_T^n (the monotone STS polytope) also defines a facet for Q_T^n (the STS polytope).” [405, p. 271]

4. Facet-lifting procedures

In this section we discuss general facet lifting procedures for the ATS polytope, which can be used to obtain new classes of inequalities from known ones. Unlike the traditional lifting procedure which calculates the coefficients of the lifted variables sequentially, often in nonpolynomial time, and with outcomes that depend on the lifting sequence, these procedures calculate sequence-independent coefficient values, often obtained by closed form expressions. We concentrate on ATS-specific liftings, and refer the reader to Queyranne and Wang [689] and Chopra and Rinaldi [183] for facet composition and lifting operations that apply to symmetric ATS inequalities only.

The subsection on cloning and clique lifting is based on [73], whereas those on T -lifting, merge lifting, and 2-cycle cloning are based on [296].

4.1. Cloning and clique lifting

Let \mathcal{F} be a given family of valid ATS inequalities, all defining proper faces, and denote by \mathcal{F}_n the restriction of \mathcal{F} to P_n .

Definition 20 *Let $\alpha x \leq \alpha_0$ be a member of \mathcal{F}_n , and h, k any two distinct nodes. Then h and k are called clones (with respect to $\alpha x \leq \alpha_0$ and \mathcal{F}) if:*

- (a) $\alpha_{ih} = \alpha_{ik}$ and $\alpha_{hi} = \alpha_{ki}$ for all $i \in V \setminus \{h, k\}$;
- (b) $\alpha_{hk} = \alpha_{kh} = \max\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij} : i, j \in V \setminus \{h, k\}, i \neq j\}$;
- (c) the inequality
$$\sum_{(i,j) \in A \setminus \delta(h)} \alpha_{ij} x_{ij} \leq \alpha_0 - \alpha_{kh}$$
 belongs to \mathcal{F}_{n-1} .

If there exists no pair of clones with respect to $\alpha x \leq \alpha_0$, the inequality $\alpha x \leq \alpha_0$ is said to be primitive.

We call a facet *regular* if it is nontrivial and its defining inequality is not the 2-cycle inequality $x_{ij} + x_{ji} \leq 1$ for some $i, j \in V$. The following lemma is used in the proof of Theorem 22, the main result of this subsection.

Lemma 21 ([62, 73]) *Let $\alpha x \leq \alpha_0$ be any inequality defining a regular facet of P , and let $k \in V$ be any node. Then there exists a sequence of $2n - 3$ tours $x^{(t)}$ with $\alpha x^{(t)} = \alpha_0$, where each tour $x^{(t)}$ ($t = 1, \dots, 2n - 3$) is associated with an arc $(i_t, j_t) \in \delta^-(k) \cup \delta^+(k)$ such that $x_{i_t j_t}^{(t)} = 1$ and $x_{i_t j_t}^{(1)} = x_{i_t j_t}^{(2)} = \dots = x_{i_t j_t}^{(t-1)} = 0$.*

Theorem 22 *If all the primitive inequalities of \mathcal{F} define regular facets of the corresponding polytopes P , then so do all the inequalities of \mathcal{F} .*

Proof: The proof is by induction on the number q of nodes that are clones. For $q = 0$ the statement is true by assumption. Suppose the theorem holds when the family \mathcal{F} is restricted to inequalities with respect to which the corresponding digraph has no more than $q_0 \geq 0$ clones, and consider any inequality of the original family with $q = q_0 + 1$ clones. Let $h, k \in V$ be any pair of clones relative to each other, with h as the “new” (q -th) clone. We will now construct a set $X \subset P$ containing $\dim(P)$ affinely independent tours x satisfying $\alpha x = \alpha_0$.

Let $\tilde{\alpha}y \leq \tilde{\alpha}_0$ be the member of \mathcal{F}_{n-1} defined at point (c) of Definition 20, where $\tilde{\alpha}_0 = \alpha_0 - \alpha_{hk}$. From the induction hypothesis, $\tilde{\alpha}y \leq \tilde{\alpha}_0$ defines a regular facet of the ATS polytope $P(\tilde{G})$ associated with the complete digraph $\tilde{G} = (\tilde{V}, \tilde{A})$ induced by $\tilde{V} := V \setminus \{h\}$. Thus there exists a set Y of $(n-1)^2 - 3(n-1) + 1$ affinely independent tours $y \in P(\tilde{G})$ satisfying $\tilde{\alpha}y = \tilde{\alpha}_0$. We initialize the set X by taking for each $y \in Y$, the tour x obtained from y by inserting node h right after k , i.e. by replacing the arc leaving k , say (k, j) , with the arcs (k, h) and (h, j) . All these tours satisfy $\alpha x = \tilde{\alpha}y + \alpha_{kh} = \tilde{\alpha}_0 + \alpha_{kh} = \alpha_0$, and are easily seen to be affinely independent. Indeed, the above transformation induces a 1-1 correspondence, φ , between \tilde{A} and a subset of A , such that for all $(i, j) \in \tilde{A}$, $x_{\varphi(i,j)} = 1$ if and only if $y_{ij} = 1$. It follows that the affine dependence of X would imply that of Y . In addition, all $x \in X$ satisfy $x_{kh} = 1$, and hence $x_{ij} = 0$ for each $(i, j) \in Q := (\delta^+(k) \cup \delta^-(h) \cup \{(h, k)\}) \setminus \{(k, h)\}$.

Now let $\{y^{(t)} : t = 1, \dots, 2(n-1) - 3\}$ be a sequence of tours of \tilde{G} with $\tilde{\alpha}y^{(t)} = \tilde{\alpha}_0$, each tour being associated with an arc (i_t, j_t) in $\tilde{Q} := (\delta^-(k) \cup \delta^+(k)) \cap \tilde{A}$ such that $y_{i_t j_t}^{(t)} = 1$ and $y_{i_t j_t}^{(1)} = \dots = y_{i_t j_t}^{(t-1)} = 0$. The existence of such a sequence follows from Lemma 21. For $t = 1, \dots, 2n - 5$, put into X the tour $x^{(t)}$ obtained from $y^{(t)}$ by inserting node h just before node k , i.e. by replacing the arc of $y^{(t)}$ entering k , say (i, k) , with arcs (i, h) and (h, k) . Clearly $\alpha x^{(t)} = \tilde{\alpha}y^{(t)} + \alpha_{hk} = \tilde{\alpha}_0 + \alpha_{hk} = \alpha_0$. As to the affine independence of the set X , it suffices to note that each $x^{(t)}$ contains an arc $(r_t, s_t) \in Q$ not contained in any previous $x \in X$ (with $(r_t, s_t) = (i_t, j_t)$ if $i_t = k$, and $(r_t, s_t) = (h, j_t)$ if $j_t = k$). At this point, the set X has $|Y| + 2n - 5 = n^2 - 3n$ members. The last tour put into X is then any tour x^* such that $\alpha x^* = \alpha_0$ and $x_{hk}^* = x_{kh}^* = 0$. To show that such x^* exists, let $(r, s) \in \tilde{A} \setminus \delta(k)$ be an arc for which the maximum in the definition of α_{hk} is attained (see point (b) of Definition 20), i.e. such that $\alpha_{hk} = \alpha_{rk} + \alpha_{ks} - \alpha_{rs}$ ($= \max_{i,j} \{\alpha_{ik} + \alpha_{kj} - \alpha_{ij}\}$), and let $y^* \in Y$ be a tour in \tilde{G} such that $\tilde{\alpha}y^* = \tilde{\alpha}_0$ and $y_{rs}^* = 1$ (such a tour exists, since $\tilde{\alpha}y \leq \tilde{\alpha}_0$ defines a

nontrivial facet of $P(\tilde{G})$. Then x^* can be obtained from y^* by inserting h into the arc (r, s) , i.e. by replacing (r, s) with the pair $(r, h), (h, s)$. The affine independence of X then immediately follows from the fact that $x_{kh}^* + x_{hk}^* = 0$, while $x_{kh} + x_{hk} = 1$ for all the other members x of X .

Thus $\alpha x \leq \alpha_0$ defines a facet of P . To see that this facet is regular, it suffices to notice that for every $(i, j) \in A$, X contains a tour x such that $x_{ij} = 1$ as well as a tour x' such that $x'_{ij} + x'_{ji} = 0$. ■

We now describe a constructive procedure, based on Theorem 22, to lift an inequality by cloning some nodes of G . Unlike the well known lifting procedure for general 0-1 polytopes [640], which calculates the lifted coefficients by solving a sequence of integer programs, one for each new coefficient, the procedure lifts simultaneously all the variables corresponding to the arcs incident with a node (or group of nodes), and calculates their coefficients by a closed form expression. Let $G' = (V', A')$ be the complete digraph induced by the node set $V' := V \cup \{n+1\}$ (where $n = |V|$).

Theorem 23 *Suppose the inequality $\alpha x \leq \alpha_0$ defines a regular facet of $P(G)$, and let k be an arbitrary but fixed seed node. Let $\delta_k := \max\{\alpha_{ik} + \alpha_{ki} - \alpha_{ij} : i, j \in A \setminus \delta(k)\}$, and $\alpha'_0 := \alpha_0 + \delta_k$. Moreover, for each $(i, j) \in A'$ let $\alpha'_{ij} := \delta_k$ if $(i, j) \in \{(k, n+1), (n+1, k)\}$; $\alpha'_{ij} := \alpha_{ik}$ if $j = n+1, i \neq k$; $\alpha'_{ij} := \alpha_{kj}$ if $i = n+1, j \neq k$; and $\alpha'_{ij} := \alpha_{ij}$ otherwise.*

Then the lifted inequality $\alpha'x' \leq \alpha'_0$ defines a regular facet of $P(G')$.

Consider now the situation arising when the above described lifting procedure is applied to a sequence of nodes according to the following scheme. An initial inequality $\alpha x \leq \alpha_0$ is given, defining a nontrivial facet of the ATS polytope associated with the complete digraph $G^1 = (V^1, A^1)$. For $\ell = 1, \dots, m$, choose any “seed” node $k^\ell \in V^\ell$, define the enlarged digraph $G^{\ell+1} = (V^{\ell+1}, A^{\ell+1})$ induced by $V^{\ell+1} := V^\ell \cup \{|V^\ell| + 1\}$, compute the coefficients α_{ij} of the new arcs $(i, h) \in A^{\ell+1} \setminus A^\ell$ according to Theorem 23, and repeat. W.l.o.g., one can assume that the seed nodes k^ℓ are always chosen in $V^1 \subseteq V^\ell$ (since each node in $V^\ell \setminus V^1$ is a clone of a node in V^1). It is not difficult to show that, for any $k \in V^1$, δ_k does not depend on the iteration in which k is selected as the seed node of the lifting. As a result, a more general lifting procedure based on Theorem 22 can be outlined as follows.

Procedure CLIQUE-LIFTING:

- 1 Let $\alpha x \leq \alpha_0$ be any valid inequality defining a regular facet of $P(G)$.

- 2 For all $k \in V$, compute $\delta_k := \max_{i,j} \{\alpha_{ik} + \alpha_{kj} - \alpha_{ij}\}$.
- 3 Define an enlarged complete digraph $G^* = (V^*, A^*)$ obtained from G by replacing each node k with a clique S_k containing $|S_k| \geq 1$ nodes.
- 4 For all $(i, j) \in A^*$, let $i \in S_{k_i}$ and $j \in S_{k_j}$, and define $\alpha_{ij}^* := \alpha_{k_i, k_j}$ if $k_i \neq k_j$, $\alpha_{ij}^* := \delta_{k_i}$ otherwise.
- 5 The inequality $\alpha^* x^* \leq \alpha_0^* := \alpha_0 + \sum_{k \in V} \delta_k (|S_k| - 1)$, where $x^* \in \mathbb{R}^{A^*}$, is then valid and defines a regular facet of $P(G^*)$.

Thus, given a complete digraph $G^* = (V^*, A^*)$, and an induced subgraph $G = (V, A)$ of G^* along with a primitive facet defining inequality $\alpha x \leq \alpha_0$ for $P(G)$, any node of $V^* \setminus V$ can be made a clone of any node of V . Applying this procedure recursively in all possible cloning combinations thus gives rise to $|V|^{|V^*| - |V|}$ distinct lifted inequalities, all facet defining for $P(G^*)$.

Clique lifting carries over to the monotone ATS polytope \tilde{P} . Indeed, it can easily be shown that, if the inequality $\alpha x \leq \alpha_0$ of Theorem 23 is support-reduced, then so is the lifted inequality $\alpha^* x^* \leq \alpha_0^*$. Therefore, as discussed in Section 3, one has the following

Theorem 24 *If $\alpha x \leq \alpha_0$ defines a nontrivial facet of both $P(G)$ and $\tilde{P}(G)$, then $\alpha^* x^* \leq \alpha_0^*$ defines a nontrivial facet of both $P(G')$ and $\tilde{P}(G')$.*

Two earlier lifting procedures for facets of P , introduced in [396], require the presence, in the support of the inequality to be lifted, of a clique satisfying certain conditions on the coefficient values. When these conditions apply, the resulting inequalities are a subset of our family, obtained by cloning a vertex of the above mentioned clique.

More recently, Padberg and Rinaldi [647] have discussed clique-liftability for the symmetric TS polytope Q^V defined on the complete undirected graph with node set V . In their terminology, a facet inducing inequality $cx \leq c_0$ for Q^V is *clique-liftable* if for every $v \in V$ there exists a real number $\alpha(c, v) \geq 0$ depending on c and v , such that for any set W with $W \cap V = \emptyset$, there exists an inequality $c^* y \leq c_0^*$, facet inducing for $Q^{V \cup W}$, whose coefficients are given by $c_{uw}^* := c_{uw}$ if $u, w \in V$, $c_{uv}^* := c_{uv}$ if $u \in V \setminus \{v\}$, $w \in W$, and $c_{vw}^* := \alpha(c, v)$ if $u, w \in W \cup \{v\}$, whereas $c_0^* := c_0 + |W| \cdot \alpha(c, v)$. From this perspective, the above results mean that in the case of the ATS polytope *all* regular facet-inducing inequalities are clique-liftable (with a definition duly modified to take into account directedness), since for each $v \in V$, $\alpha(c, v)$ exists and is equal to δ_v .

Also related to clique lifting is the *0-node lifting* introduced by Naddef and Rinaldi [616, 618] in their study of the graphical relaxation of the STS polytope (as defined in Chapter 2) and extended to the graphical ATS polytope (as defined in Section 3) by Chopra and Rinaldi [183]. An inequality $\pi x \geq \pi_0$, $\pi \geq 0$, is said to satisfy the *shortest path condition* if, for each $(i, j) \in A$, the π -length of every directed path from i to j is greater or equal to π_{ij} (a necessary condition for an inequality to be nontrivial facet-defining for the graphical ATS polytope [183]). On a complete digraph, this is equivalent to requiring that $\pi_{ij} \leq \pi_{ik} + \pi_{kj}$ for all triples $i, j, k \in V$ of distinct nodes. Moreover, the inequality $\pi x \geq \pi_0$ is said to be *tight triangular* whenever it satisfies the shortest path condition and, for each $k \in V$, there exists $(i, j) \in A \setminus \delta(k)$ such that $\pi_{ij} = \pi_{ik} + \pi_{kj}$. 0-node lifting then consists of replacing a given node v by a clique of clones connected one to each other by arcs with $\pi_{v'v''} = 0$ (hence the name 0-node lifting). This operation was first studied in the context of the graphical STS polytope by Naddef and Rinaldi [616, 618], who also provided necessary and sufficient conditions under which the 0-node lifting of a tight triangular inequality preserves the facet-defining property with respect to the standard (sometimes called *flat*) STS polytope. At a later time, Chopra and Rinaldi [183] proved that 0-node lifting always preserves the facet-defining property for the graphical ATS polytope, while they did not address the question of whether this is also true with respect to the flat ATS polytope. Observe that an inequality $\pi x \geq \pi_0$ can trivially be put in the form $\alpha x \leq \alpha_0$ by simply setting $(\alpha, \alpha_0) := -(\pi, \pi_0)$, hence $\pi x \geq \pi_0$ is tight triangular if and only if $\alpha \leq 0$ and $\alpha_{vv} = 0$ for all $v \in V$. From this perspective, 0-node lifting reduces to clique lifting, thus it has the nice property (not known to be shared by its symmetric counterpart) of always preserving the facet-defining quality with respect to the flat ATS polytope.

4.2. T-lifting

T-lifting aims at extending the following construction [402]: given the SEC $x(S, S) \leq |S| - 1$ associated with a subset S with cardinality k , $2 \leq k \leq n - 2$, choose three distinct nodes, say $p, q \in V \setminus S$ and $w \in S$, and add the term $x_{pw} + x_{wq} + x_{pq}$ to the left-hand side of the SEC, and 1 to its right-hand side, thus obtaining a T_k inequality which is known to be facet-defining for P_n for $k \neq n - 3$.

Now let $\alpha x \leq \alpha_0$ be any valid inequality for P_n . To simplify notation, in the sequel we will often use notation

$$\alpha_{kk} = \max\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij} : (i, j) \in A \setminus \delta(k)\}$$

to represent the clique-lifting coefficient δ_k defined in Theorem 23. We assume:

- (a) $\alpha_{ij} \geq 0$ and integer for all $(i, j) \in A$;
- (b) there exist two distinct *isolated* nodes p, q , i.e., $\alpha_{ih} = \alpha_{hi} = 0$ for all $i \in V$ ($h = p, q$);
- (c) a node $w \in V \setminus \{p, q\}$ with $\alpha_{ww} = 1$ exists, such that $\alpha^w y \leq \alpha_0 - 1$ (where α^w is the restriction of α onto $G(V \setminus \{w\})$) is valid and defines a nonempty face of P_{n-1} .

Note that the above conditions imply $\alpha_{ij} \in \{0, 1\}$ for all $(i, j) \in \delta(w)$. Starting with $\alpha x \leq \alpha_0$, we define the *T-lifted inequality* $\beta x := \alpha x + x_{pw} + x_{wq} + x_{pq} \leq \beta_0 := \alpha_0 + 1$, obtained by adding (and then rounding) the following 6 valid inequalities, all weighed by 1/2: $x(\delta^+(i)) \leq 1$ for $i \in \{p, w\}$; $x(\delta^-(j)) \leq 1$ for $j \in \{w, q\}$, $\alpha x \leq \alpha_0$, and the inequality obtained from $\alpha^w y \leq \alpha_0 - 1$, see condition (c) above, by re-introducing node w as a clone of p .

We now address the important question of whether $\beta x \leq \beta_0$ is facet inducing, assuming that this is the case for the starting inequality. An example of an application that does not produce a facet corresponds to T_k inequalities on $n = k + 3$ nodes. Actually, T-lifting can in some cases even produce an inequality with *non-maximal* coefficients, i.e., some β_{ij} can singularly be increased without affecting the validity of $\beta x \leq \beta_0$.

Let $\bar{V} := V \setminus \{p, w, q\}$, $J^+ := \{j \in \bar{V} : \alpha_{wj} > 0\}$, $J^0 := \bar{V} \setminus J^+$, $I^+ := \{i \in \bar{V} : \alpha_{iw} > 0\}$, and $I^0 := \bar{V} \setminus I^+$. Moreover, let $F := \{x \in P_n : \beta x = \beta_0\}$ denote the face induced by $\beta x \leq \beta_0$.

Theorem 25 *Let $\alpha x \leq \alpha_0$ satisfy conditions (a) to (c) above, and let the inequality $\alpha^w y \leq \alpha_0 - 1$ induce a nontrivial facet of P_{n-1} . Assume that all the coefficients β_{pj} , $j \in J^+$, and β_{iq} , $i \in I^+$, are maximal with respect to the T-lifted inequality $\beta x \leq \beta_0$. Then F is either a facet of P_n , or else a face of dimension $\dim(P_n) - 2$ contained in the hyperplane $H := \{x \in \mathbb{R}^A : x_{qp} = x(p, J^*) + x(I^*, q)\}$, where $J^*, I^* \subseteq \bar{V}$ are such that $J^0 \subseteq J^*$ and $I^0 \subseteq I^*$.*

In the case of a T_k inequality on $n = k + 3$ nodes, the non-maximal face induced by $\beta x \leq \beta_0$ is contained in the hyperplane H defined by the equation $x_{qp} = x_{pz} + x_{zq}$, where z is the unique node in $V \setminus (S \cup \{p, q\})$. The existence of H is however a pathological occurrence for T-lifting, as shown in the following.

Corollary 26 *Under the assumptions of Theorem 25, F is a facet of P_n if there exists $x^* \in F$ such that $x_{qp}^* = 0$ and $x_{ij}^* = 1$ for a certain $(i, j) \in (p, J^0) \cup (I^0, q)$.*

It is not hard to see that the above corollary applies, e.g., when there exists $r \in \bar{V}$ such that $\alpha_{ir} = \alpha_{ri} = 0$ for all i , and $(J^0 \cup I^0) \setminus \{r\} \neq \emptyset$. Therefore the most restrictive requirement in Theorem 25 is the coefficient maximality. The reader is referred to [296] for a discussion of necessary and sufficient conditions for such a maximality.

A large class of inequalities can be obtained by iterating the application of T-lifting on clique tree inequalities. Let $C = (W_1, \dots, W_r)$ be a clique tree on $n \geq 7$ nodes, and let $\alpha x \leq \alpha_0$ be the associated facet-inducing inequality. Now let $T := \{p_j, w_j, q_j : j = 1, \dots, m\}$ contain $3m$ nodes such that, for $j = 1, \dots, m$, nodes p_j and q_j are not covered by C , whereas w_j belongs to a tooth and to no handles. Moreover, assume that each tooth contains at least one node not belonging to any handle nor to $\{w_1, \dots, w_m\}$. Then for $n \geq 7$ and $m \geq 0$, the T -clique tree inequality $\sum_{i=1}^r x(W_i, W_i) + \sum_{j=1}^m (x_{p_j w_j} + x_{w_j q_j} + x_{p_j q_j}) \leq \sigma(C) + m$ is valid and facet-inducing for P_n (except when either $r = 1, m = 1$ and $n = |W_1| + 3$, or $r = 1, m = 2$ and $n = |W_1| + 4$). Different classes of facet-inducing inequalities can be obtained similarly, by applying T -lifting to different facet-defining $\alpha x \leq \alpha_0$ such as, e.g., D_k^+ and D_k^- -inequalities.

4.3. Merge lifting

Merge lifting is an operation that produces (under appropriate conditions) a facet inducing inequality for P_{n+1} starting from two ‘almost identical’ inequalities for P_n . Roughly speaking, the new inequality is obtained by swapping the coefficients of arcs in $\delta^+(h)$ and $\delta^-(h)$ for a certain node h (where $h = n$ is assumed for the sake of easy notation).

More specifically, we are given two valid inequalities for P_n , say $\alpha x \leq \alpha_0$ and $\beta x \leq \beta_0$, which are ‘almost identical’ in the sense that $\alpha_{ij} = \beta_{ij}$ for all $(i, j) \in A \setminus \delta(n)$. For a real parameter ω , we define an inequality $\gamma x' \leq \omega$ for P_{n+1} as follows: $\gamma_{ij} := \alpha_{ij} (= \beta_{ij})$, for all $i, j \in A \setminus \delta(n)$, $\gamma_{n,n+1} := \omega - \alpha_0, \gamma_{n+1,n} := \omega - \beta_0$, whereas for all $i \in V \setminus \{n\}$ we set $\gamma_{in} := \alpha_{in}, \gamma_{n+1,i} := \alpha_{ni}, \gamma_{ni} := \beta_{ni}, \gamma_{i,n+1} := \beta_{in}$ (see Figure 3.8 for an illustration). By construction, the above inequality is satisfied by all $x' \in P_{n+1}$ such that $x'_{n,n+1} + x'_{n+1,n} = 1$, independently of ω . It then follows that $\gamma x' \leq \omega$ is valid for P_{n+1} if and only if ω is chosen greater or equal to $\gamma_0 := \max\{\gamma x' : x' \in P_{n+1}, x'_{n,n+1} = x'_{n+1,n} = 0\}$. When $\omega = \gamma_0$, we say that $\gamma x' \leq \omega$ has been obtained from $\alpha x \leq \alpha_0$ and $\beta x \leq \beta_0$ through *merge lifting*.

It can be proved that $\gamma x' \leq \gamma_0$ defines a facet of P_{n+1} if $\alpha x \leq \alpha_0$ is facet defining for P_n , and $\beta x \leq \beta_0$ satisfies a certain regularity condition [296]. Observe that node cloning can be viewed as a special case of

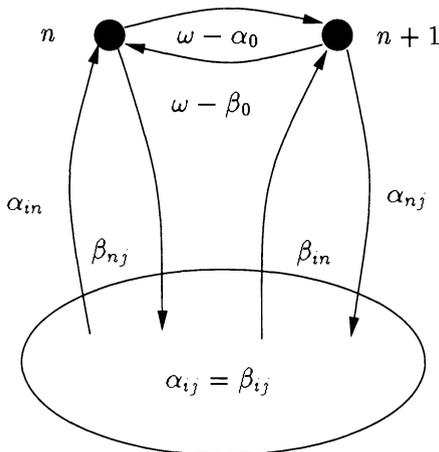


Figure 3.8. Illustration of the merge lifting operation.

merge lifting, arising when $(\alpha, \alpha_0) = (\beta, \beta_0)$; in this case, γ_0 can easily be computed as $\alpha_0 + \alpha_{nn}$.

As an example of merge lifting application, let $\alpha x \leq \alpha_0$ be the inequality associated with any clique tree $C = (W_1, \dots, W_r)$ that leaves node n uncovered, and let $\beta x \leq \beta_0$ be the one associated with the clique tree C' obtained from C by including node n into, say, $W_1 \setminus (W_2 \cup \dots \cup W_r)$. Therefore $\alpha x := \sum_{i=1}^r x(W_i, W_i) \leq \alpha_0 := \sigma(C)$, and $\beta x := x(W_1 \cup \{n\}, W_1 \cup \{n\}) + \sum_{i=2}^r x(W_i, W_i) \leq \beta_0 := \sigma(C) + 1$. Both inequalities are known to define facets of P_n . By construction, the two inequalities are suitable for merge lifting. The inequality $\gamma x' \leq \omega$ is in this case $\sum_{i=1}^r x'(W_i, W_i) + x'(n, W_1) + x'(W_1, n+1) + (1 + \theta)x'_{n,n+1} + \theta x'_{n+1,n} \leq \sigma(C) + 1 + \theta$, where we have defined $\theta := \omega - \sigma(C) - 1$ to simplify notation. For sufficiently large θ , this inequality is valid for P_{n+1} . In order to have a facet, however, one has to choose not too large a value for θ , namely $\theta = 1$ if W_1 is a tooth of C , and $\theta = 0$ if W_1 is a handle of C (in this latter case, if C is a comb we obtain a C2 inequality). Notice that, unlike the starting inequalities $\alpha x \leq \alpha_0$ and $\beta x \leq \beta_0$, the merge-lifted inequality $\gamma x \leq \gamma_0$ is asymmetric.

The iterative application of merge lifting on the T-clique tree inequality defined in the previous subsection, leads to large classes of facets. We describe one these classes, containing clique tree, C2, and T_k inequalities as special cases. Let C be a given clique tree, $T = \{p_j, q_j, w_j : j = 1, \dots, m\}$ be as defined in the previous subsection, and let $Z = \{a_j, b_j : j = 1, \dots, k\}$ contain $2k$ nodes not covered by C nor by T . Each pair a_j, b_j is associated with a distinct clique of C , say $W_{i(j)}$,

and with a value θ_j (with $\theta_j = 1$ if $W_{i(j)}$ is a tooth of C , $\theta_j = 0$ otherwise). Then for $n \geq 7$, $m \geq 0$ and $k \geq 0$ the following *ZT-clique tree inequality*

$$\sum_{i=1}^r x(W_i, W_i) + \sum_{j=1}^m (x_{p_j w_j} + x_{w_j q_j} + x_{p_j q_j}) + \sum_{j=1}^k (x(a_j, W_{i(j)}) + x(W_{i(j)}, b_j) + (1 + \theta_j) x_{a_j b_j} + \theta_j x_{b_j a_j}) \leq \sigma(C) + m + \sum_{j=1}^k (1 + \theta_j)$$

is valid and facet-inducing (except in a few pathological cases) for P_n .

4.4. Two-cycle cloning

We now describe 2-cycle cloning, an operation that under appropriate conditions constructs a facet-inducing inequality for P_{n+2} starting from a facet of P_n . This operation is related to the *edge cloning* procedure proposed for the symmetric TSP by Naddef and Rinaldi [618], and analyzed by Chopra and Rinaldi [183] in the context of symmetric ATS inequalities. However, 2-cycle cloning is ATS-specific, in that it allows one to lift both symmetric and asymmetric inequalities.

A 2-cycle is simply a cycle of length 2. Let $\alpha x \leq \alpha_0$ be a valid inequality for P_n . Given two distinct nodes h and k such that $\Delta(h, k) := (\alpha_{hh} + \alpha_{kk}) - (\alpha_{hk} + \alpha_{kh}) > 0$, let $\alpha^* x' \leq \alpha_0^* := \alpha_0 + \alpha_{hh} + \alpha_{kk}$ be the inequality for P_{n+2} obtained from $\alpha x \leq \alpha_0$ by adding nodes $h' := n + 1$ and $k' := n + 2$ as clones of h and k , respectively. We define the following (not necessarily valid) inequality for P_{n+2} :

$$\begin{aligned} \beta x' &:= \alpha^* x' - \Delta(h, k) x'_{hh'} + x'_{h'h} + x'_{kk'} + x'_{k'k} \leq \beta_0 \\ &:= \alpha_0^* - \Delta(h, k) = \alpha_0 + \alpha_{hk} + \alpha_{kh} \end{aligned}$$

and say that $\beta x' \leq \beta_0$ is obtained from $\alpha x \leq \alpha_0$ by *cloning the 2-cycle induced by h and k* . In other words, $\beta x' \leq \beta_0$ is obtained from $\alpha^* x' \leq \alpha_0^*$ by adding the invalid constraint $-(x'_{hh'} + x'_{h'h} + x'_{kk'} + x'_{k'k}) \leq -1$ weighted by $\Delta(h, k)$. Note that, although similar, nodes h and h' , as well as k and k' , are no longer clones with respect to $\beta x' \leq \beta_0$ because of the coefficient reduction.

Theorem 27 *Assume that $\alpha x \leq \alpha_0$ defines a regular facet of P_n . If $\beta x' \leq \beta_0$ is valid, then it defines a regular facet of P_{n+2} .*

By construction, $\beta x' \leq \beta_0$ is valid for P_{n+2} if and only if $\alpha^* x' \leq \alpha_0^* - \Delta(h, k)$ holds for all $x' \in P_{n+2}$ such that $x'_{hh'} = x'_{h'h} = x'_{kk'} = x'_{k'k} = 0$. In several cases, α is integer and $\Delta(h, k) = 1$, hence the above condition simply requires that no $x' \in P_{n+2}$ with $\alpha^* x' = \alpha_0^*$ exists, such that

$x'_{hh'} = x'_{h'h} = x'_{kk'} = x'_{k'k} = 0$. Easy-to-check conditions for the validity of $\beta x' \leq \beta_0$ were given recently by Caprara, Fischetti and Letchford [161] in case a rank-1 Chvátal truncation proof of validity is available for $\alpha x \leq \alpha_0$.

A relevant application of 2-cycle cloning arises when $\alpha x \leq \alpha_0$ is the ZT-clique tree inequality of the previous subsection, and the two nodes h, k belong to a 2-node *pendant* tooth (i.e., to a tooth intersecting only one handle). This enlarges the ZT-clique tree class so as to cover the Padberg-Hong [643] *chain inequalities*, a special case arising when $\alpha x \leq \alpha_0$ is associated with a comb and 2-cycle cloning is iterated on a single pendant tooth. From Theorem 27, the members of this new class define facets of P_n (this extends the corresponding result for chain inequalities, due to Queyranne and Wang [689]). Moreover, one can show that the 2-cycle induced by the pair p, q chosen for T-lifting can always be cloned, in the sense that the resulting inequality is valid (and hence facet-inducing). The same holds for the node pairs a_j, b_j of ZT-clique tree inequalities. Therefore, one can make the class of ZT-clique tree inequalities even larger by iteratively applying 2-cycle cloning, thus obtaining inequalities that generalize properly clique tree, C2, T_k , chain inequalities, as well as the CAT inequalities [62] described in Section 6

Other examples of application of 2-cycle cloning are given in [161], where extended versions of C3 inequalities and of SD inequalities (the latter described in Section 7) are analyzed.

5. Equivalence of inequalities and canonical forms

We will consider only valid inequalities with rational coefficients defining proper, nonempty faces of P . Two given inequalities $\alpha x \leq \alpha_0$ and $\alpha' x' \leq \alpha'_0$ are said to be *equivalent* if one of them is a linear combination of the other and the degree constraints, i.e., if there exist $\rho \in \mathbb{R}_+$ and $u_i, v_i \in \mathbb{R}$ ($i \in V$) such that $\alpha'_0 = \rho\alpha_0 + \sum_{i \in V}(u_i + v_i)$ and, for all $(i, j) \in A$, $\alpha'_{ij} = \rho\alpha_{ij} + u_i + v_j$. It is well known that $\alpha x \leq \alpha_0$ and $\alpha' x \leq \alpha'_0$ define the same facet of P (assuming that they are both facet inducing) if and only if they are equivalent.

As in the previous section, for all $k \in V$ we define $\alpha_{kk} := \max\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij} : (i, j) \in A \setminus \delta(k)\}$. Note that equivalence transformations affect the values α_{kk} (which are the same as the values δ_k defined in Theorem 23) the same way as the other coefficients α_{ij} , i.e., if $\rho > 0$ and $\alpha'_{ij} = \rho\alpha_{ij} + u_i + v_j$, for all $(i, j) \in A$, then also $\alpha'_{kk} = \rho\alpha_{kk} + u_k + v_k$ for each $k \in V$. Indeed, $\alpha'_{kk} := \max_{i,j}\{\alpha'_{ik} + \alpha'_{kj} - \alpha'_{ij}\} = \max_{i,j}\{\rho(\alpha_{ik} + \alpha_{kj} - \alpha_{ij}) + u_k + v_k\} = \rho \max_{i,j}\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij}\} + u_k + v_k = \rho\alpha_{kk} + u_k + v_k$.

We will analyze next the notion of h -canonical form introduced in Balas and Fischetti [72] (see also Remark 9.2 in [403, 404]).

Definition 28 *An inequality $\beta x \leq \beta_0$ is said to be in h -canonical form with respect to any given $h \in V$ if:*

- (i) $\beta_{ih} = \beta_{hi} = 0$ for all $i \in V \setminus \{h\}$.
- (ii) $\beta \geq 0$ and there exists $(r, s) \in A$ with $r, s \neq h$ and $\beta_{rs} = 0$,
- (iii) the coefficients β_{ij} and the right hand side β_0 are relatively prime integers.

The h -canonical form of a given inequality $\alpha x \leq \alpha_0$ is an inequality $\beta x \leq \beta_0$ in h -canonical form, which is equivalent to $\alpha x \leq \alpha_0$. Any inequality $\alpha x \leq \alpha_0$ can be put in this form simply by defining $\beta_{ij} := \sigma[\alpha_{ij} + (\alpha_{hh} - \alpha_{ih}) + (-\alpha_{hj})] = \sigma[\alpha_{hh} - (\alpha_{ih} + \alpha_{hj} - \alpha_{ij})] \geq 0$ for all $i, j \in V$, and $\beta_0 := \sigma[\alpha_0 + \sum_{i \in V} (\alpha_{hh} - \alpha_{ih} - \alpha_{hi})]$, where $\sigma > 0$ is a scaling factor chosen so as to satisfy the normalization condition (iii).

Theorem 29 *Two inequalities $\alpha x \leq \alpha_0$ and $\alpha' x \leq \alpha'_0$ are equivalent if and only if their h -canonical forms $\beta x \leq \beta_0$ and $\beta' x \leq \beta'_0$ are the same.*

We now state three interesting properties of the h -canonical form.

Theorem 30 *Let the inequality $\beta x \leq \beta_0$ be in h -canonical form for some $h \in V$. If there exists a symmetric inequality $\alpha x \leq \alpha_0$ equivalent to $\beta x \leq \beta_0$, then $\beta x \leq \beta_0$ is symmetric.*

Theorem 31 *If two given nodes s and t are clones with respect to some inequality $\alpha x \leq \alpha_0$, then s and t are also clones with respect to the h -canonical form $\beta x \leq \beta_0$ of $\alpha x \leq \alpha_0$.*

Thus the h -canonical form conspicuously exhibits all clones, some of which could be hidden in other formulations. In addition, when checking equivalence among the members of a given family of inequalities through transformation to h -canonical form, we can restrict ourselves to considering only the primitive inequalities of the family.

Theorem 32 *Inequalities in h -canonical form are support reduced.*

As a consequence of the above theorem, if $\alpha x \leq \alpha_0$ defines a nontrivial facet of P , then its h -canonical form $\beta x \leq \beta_0$ defines a facet of both P and \hat{P} (see Theorem 14). Note that this is not necessarily true of other inequalities equivalent to $\alpha x \leq \alpha_0$, some of which can even be invalid for \hat{P} .

Finally we note that if $\alpha x \leq \alpha_0$ defines the *improper* face of P , i.e., if for some multipliers u_i, v_j we have $\alpha_{ij} := u_i + v_j$, $(i, j) \in A$, and $\alpha_0 := \sum_i (u_i + v_i)$, then its h -canonical form $\beta x \leq \beta_0$ has $\beta = 0$ and $\beta_0 = 0$. Indeed, $\beta_{ij} = \sigma(\alpha_{hh} - \alpha_{ih} - \alpha_{hj} + \alpha_{ij}) = 0$, $\forall i, j \in V$ (note that $\alpha_{hh} = \max_{i,j} \{\alpha_{ih} + \alpha_{hj} - \alpha_{ij}\} = u_h + v_h$).

The use of h -canonical forms goes beyond determining whether two given ATS inequalities are equivalent. In particular, Queyranne and Wang [687] exploited extensively h -canonical forms to establish whether a given inequality $\alpha x \leq \alpha_0$ defines a facet of the ATS polytope. Indeed, according to the so-called *indirect method* [405] the proof of the latter consists of showing that any valid ATS inequality $\beta x \leq \beta_0$ such that $\{x \in P : \alpha x = \alpha_0\} \subseteq \{x \in P : \beta x = \beta_0\} \subset P$ is indeed equivalent to $\alpha x \leq \alpha_0$, a property which is easier to establish if one assumes without loss of generality that both $\alpha x \leq \alpha_0$ and $\beta x \leq \beta_0$ are in h -canonical form with respect to some (arbitrary) node h .

We conclude this subsection by observing that the h -canonical form has the disadvantage of depending on h , which can cause some problems when comparing two members of a family of (as opposed to single) inequalities. Thus, when choosing h as a particular vertex of the first inequality, one should consider all possible roles for h in the second inequality. A different canonical form, which does not depend on the node h , has been studied in [73]. Roughly speaking, the dependence on node h is eliminated by adding the h -canonical forms for all $h \in V$. More precisely, the *canonical form* $\beta x \leq \beta_0$ of a given inequality $\alpha x \leq \alpha_0$ is defined as follows: Let first $\Delta := \sum_{h \in V} \alpha_{hh}$, $r_i := \sum_{j \in V} \alpha_{ij}$ for $i \in V$, $c_j := \sum_{i \in V} \alpha_{ij}$ for $j \in V$, $\gamma_0 := n\Delta + n\alpha_0 - \sum_{i \in V} (r_i + c_i)$, and $\gamma_{ij} := \Delta + n\alpha_{ij} - r_i - c_j$ for $i, j \in V$, and then set $\beta_0 := \sigma(\gamma_0 - n\varepsilon)$ and $\beta_{ij} := \sigma(\gamma_{ij} - \varepsilon)$ for $i, j \in V$, where $\varepsilon := \min\{\gamma_{ij} : i, j \in V\}$ and $\sigma > 0$ is a scaling factor making the coefficients β_{ij} and β_0 relatively prime integers.

As an example, consider the following T_2 inequality, defined on a digraph with 4 nodes (see Figure 3.9): $\alpha x := x_{12} + x_{13} + x_{23} + x_{24} + x_{42} \leq 2$. Its canonical form is obtained by computing $(\alpha_{hh}) = (1, 2, 1, 1)$, $\Delta = 5$, $(r_i) = (3, 4, 1, 2)$, $(c_j) = (1, 4, 3, 2)$, $\varepsilon = 0$ and $\sigma = 1$, leading to the inequality $\beta x := 2(x_{12} + x_{23} + x_{34} + x_{41}) + 3(x_{13} + x_{31} + x_{24} + x_{42}) \leq 8$.

Theorem 33 *Theorems 29 to 31 remain valid if the h -canonical form is replaced by the canonical form of the inequality involved.*

An example of application of canonical form is given in Section 7, where it is used to point out the existence of equivalent members in the class of the SD inequalities.

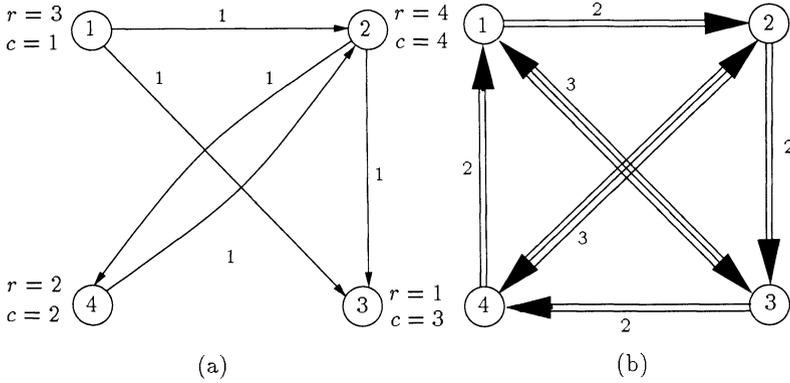


Figure 3.9. A T_2 inequality (a) and its canonical form (b).

6. Odd closed alternating trail inequalities

Odd Closed Alternating Trail (CAT) inequalities were introduced in [62]. An *assignment* in the complete digraph G is a spanning subgraph that is the node-disjoint union of directed cycles. A frequently used relaxation of the ATS problem is the *Assignment Problem* (AP), whose constraint set is obtained from the ATS formulation (1)-(3) by removing all SEC's (3). An *asymmetric* assignment is one that contains no directed 2-cycles, i.e. that satisfies the 2-node SEC's:

$$x_{ij} + x_{ji} \leq 1 \quad \text{for all } (i, j) \in A, i < j. \tag{7}$$

The *Asymmetric Assignment* (AA) polytope P_A is the convex hull of incidence vectors of asymmetric assignments, i.e.

$$P_A := \text{conv} \{x \in \{0, 1\}^A : x \text{ satisfies (1), (4), and (7)}\}.$$

Unlike the standard AP, minimizing a linear function over P_A is an NP-hard problem [813].

An arc set that is the node disjoint union of directed cycles and/or paths will be called an (asymmetric) *partial assignment*. The *Asymmetric Partial Assignment* (APA) polytope on G , or monotonization of P_A , is

$$\tilde{P}_A := \text{conv} \{x \in \{0, 1\}^A : x_{ij} + x_{ji} \leq 1 \text{ for all } (i, j) \in A, i < j, \text{ and } x(\delta^+(v)) \leq 1, x(\delta^-(v)) \leq 1 \text{ for all } v \in V\}.$$

In this section we describe classes of facet inducing inequalities for the ATS polytope P that correspond to facets of the AA polytope P_A . They are associated with certain subgraphs of G (called closed alternating

trails) that correspond to odd holes of the intersection graph of the coefficient matrix of the AA polytope. The reader is referred to [62] for proofs of the main results.

Let $G^* = (V, E)$ be the intersection graph of the coefficient matrix of the system (1)-(4) and (7). Then G^* has a vertex for every arc of G ; and two vertices of G^* corresponding, say, to arcs (p, q) and (r, s) of G , are joined by an edge of G^* if and only if either $p = r$, or $q = s$, or $p = s$ and $q = r$. Two arcs of G will be called G^* -adjacent if the corresponding vertices of G^* are adjacent. Clearly, there is a 1-1 correspondence between APA's in G and vertex packings (independent vertex sets) in G^* , hence the APA polytope \tilde{P}_A defined on G is identical to the vertex packing polytope defined on G^* .

We define an *alternating trail* in G as a sequence of distinct arcs $T = (a_1, \dots, a_t)$, such that for $k = 1, \dots, t - 1$, a_k and a_{k+1} are G^* -adjacent, but a_k, a_ℓ , $\ell > k + 1$, are not (with the possible exception of a_1 and a_t). If a_1 and a_t are G^* -adjacent, the alternating trail T is *closed*. An arc $a_k = (p, q)$ of T is called *forward* if p is the tail of a_{k-1} , or q is the head of a_{k+1} , or both; it is called *backward* if q is the head of a_{k-1} , or p is the tail of a_{k+1} , or both. The definition of an alternating trail T implies that the direction of the arcs of T alternates between forward and backward, except for pairs a_k, a_{k+1} that form a directed 2-cycle entered and exited by T through the same node, in which case a_k and a_{k+1} are both forward or both backward arcs. It also implies that all the 2-cycles of T are node-disjoint (since two 2-cycles of G that share a node define a 4-cycle in G^*). Notice that T traverses a node at most twice, and the number of arcs of T incident from (incident to) any node is at most 2.

Let $G[T]$ denote the subdigraph of G generated by T , i.e., $G[T]$ has T as its arc set, and the endpoints of the arcs of T as its node set. Further, for any $v \in N$, let $\deg_T^+(v)$ and $\deg_T^-(v)$ denote the outdegree and indegree of node v in $G[T]$, respectively.

The *length* of an alternating trail is the number of its arcs. An alternating trail will be called *even* if it is of even length, *odd* if it is of odd length. We will be interested in *closed alternating trails* (CATs for short) of odd length, the reason being the following.

Proposition 34 *There is a 1-1 correspondence between odd CATs in G and odd holes (chordless cycles) in G^* .*

It is well known [640] that the odd holes of an undirected graph give rise to facets of the vertex packing polytope defined on the subgraph generated by the odd hole, and that these facets in turn can be lifted into facets of the polytope defined on the entire graph. In order to make the lifting procedure conveniently applicable to the particular vertex pack-

ing polytope associated with G^* , we need some structural information concerning adjacency relations on G^* .

Let T be a CAT in G . A node of $G[T]$ will be called a *source* if it is the common tail of two arcs of T , and a *sink* if it is the common head of two arcs of T . A node of $G[T]$ can thus be a source, or a sink, or both, or none. A node of $G[T]$ that is neither a source nor a sink will be called *neutral*. A neutral node is incident only with the two arcs of a 2-cycle. A 2-cycle will be called *neutral* if it contains a neutral node. Several odd CATs are illustrated in Figure 3.10. The sources and sinks of $G[T_1]$ are nodes 1, 2 and 2, 4, respectively, while 3 is neutral. $G[T_2]$ has three neutral nodes, 1, 4 and 6, while nodes 2, 3 and 5 are both sources and sinks. $G[T_3]$ has sources 1 and 4, sinks 2, 3 and 4, while 5 is a neutral node. Further, $G[T_1]$, $G[T_2]$ and $G[T_3]$ have one, three and one neutral 2-cycles, respectively, and $G[T_3]$ also has a non-neutral 2-cycle.

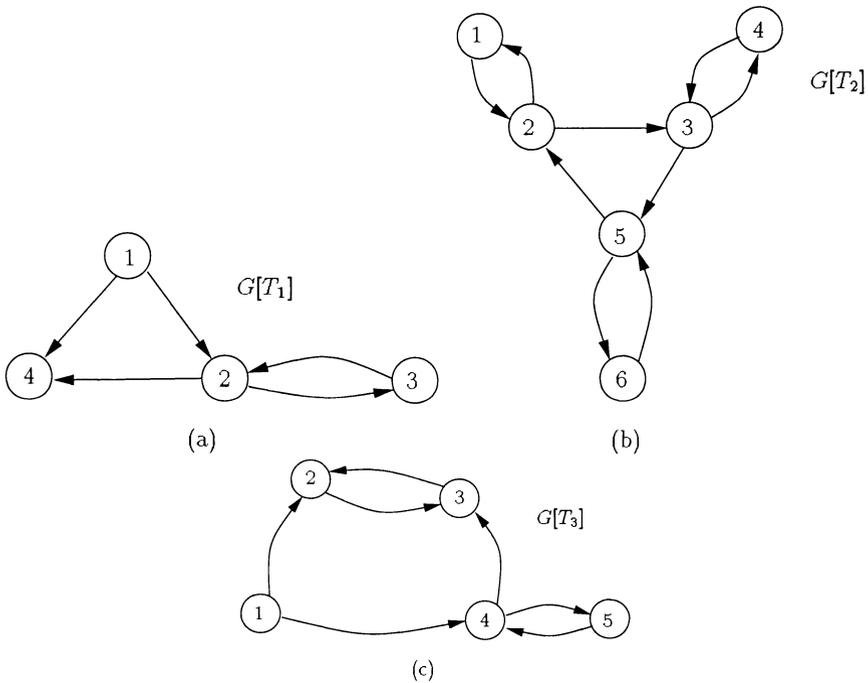


Figure 3.10. Some odd CAT's.

A *chord* of a CAT T is an arc $a \in A \setminus T$ joining two nodes of $G[T]$. One can distinguish between several types of chords, of which the most important ones are those of *type 1*. A chord (u, v) is of type 1 if it joins a source to a sink (i.e. $deg_T^+(u) = deg_T^-(v) = 2$).

We are now ready to characterize the class of facet inducing inequalities of the APA polytope \tilde{P}_A associated with odd CATs. We consider first the subgraph generated by an odd CAT.

Proposition 35 *Let T be an odd CAT of length t , and let $\tilde{P}_A(G[T])$ be the APA polytope defined on $G[T]$. Then the inequality*

$$x(T) \leq (t - 1)/2 \quad (8)$$

defines a facet of both $\tilde{P}_A(G[T])$ and $\tilde{P}(G[T])$.

Next we apply sequential lifting to (8) and identify inequalities of the form

$$x(T) + \sum (\alpha_{ij} x_{ij} : (i, j) \in A \setminus T) \leq (t - 1)/2 \quad (9)$$

that define facets of \tilde{P}_A . We take the arcs of $A \setminus T$ in any order such that the chords of Type 1 precede all other arcs. It is then not hard to show that, for any such lifting sequence, the lifting coefficient of each arc $a \in A \setminus T$ is $\alpha_a = 1$ if α is a chord of Type 1, and $\alpha_a = 0$ otherwise. This leads to the following result.

Theorem 36 *Let T be an odd CAT of length t , and let C_1 be the set of its chords of type 1. Then the odd CAT inequality*

$$x(T \cup C_1) \leq (t - 1)/2 \quad (10)$$

defines a facet of both \tilde{P}_A and \tilde{P} .

Proof outline: One can show that a chord of type 1, if lifted first, gets a coefficient of 1. It then follows by induction that all remaining arcs of C_1 , if lifted before those in $A \setminus (T \cup C_1)$, get a coefficient of 1. Finally, all arcs lifted after those in C_1 get a coefficient of 0. ■

If a different lifting sequence is used, one can get facet inducing inequalities with a coefficient of 1 for some chords not in C_1 and a coefficient of 0 for some chords in C_1 ; but these inequalities do not share the simplicity and regularity of the family (10) (see [62] for details).

The arc sets corresponding to the support of each odd CAT inequality in the digraph with 6 nodes are shown (up to isomorphism) in Figure 3.11, with the arcs of T and C_1 shown in solid and shaded lines, respectively.

Remark 37 *The odd CAT inequalities (10) have Chvátal rank 1.*

Proof: Each inequality (10) associated with an odd CAT T can be obtained by adding the outdegree equations (1) associated with each

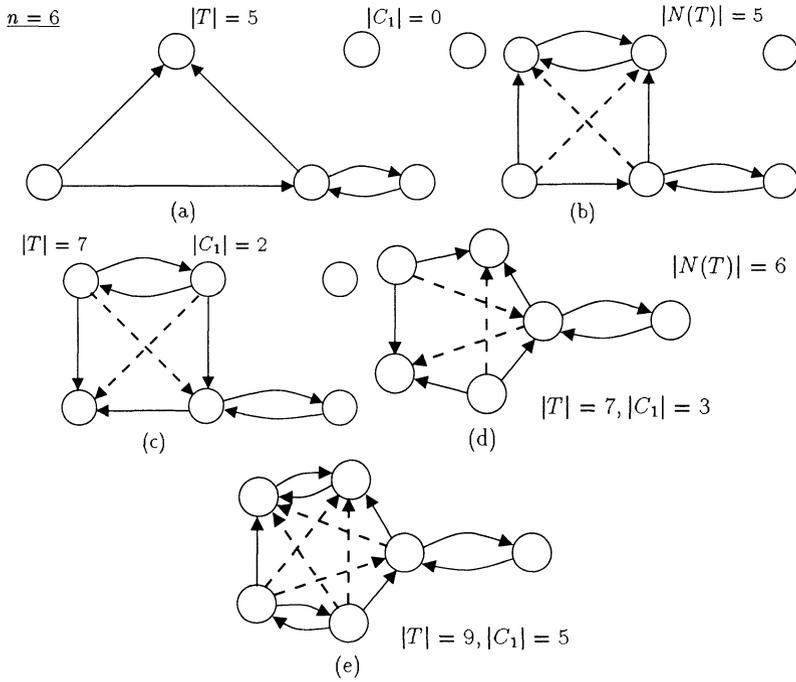


Figure 3.11.

source of $G[T]$, the indegree equations (4) associated with each sink of $G[T]$, and the inequalities (7) associated with each two-cycle of $G[T]$; then dividing by two the resulting inequality and rounding down all coefficients to the nearest integer. ■

We finally address the polytopes P_A and P , and show that, with the exception of one special case for $n = 5$ and one for $n = 6$, the odd CAT inequality defines a facet of the ATS polytope P . It then follows that it also defines a facet of the AA polytope P_A .

We consider first three cases with small n and $|T|$. The shortest odd CAT has length 5 and it uses 4 nodes, i.e. (10) is defined only for $|T| \geq 5$ and $n \geq |V[T]| \geq 4$.

Proposition 38 *Let $|T| = 5$ and $|V[T]| = 4$. Then the odd CAT inequality (10) defines a facet of P_A and P if $n = 4$ and $n = 6$, but not if $n = 5$.*

Proposition 39 *Let $|T| = 7$ and $|V[T]| = 5$. Then the odd CAT inequality (10) defines a facet of P_A and P for $n = 5$ and $n = 6$.*

Finally, for $|V[T]| = 6$ we will denote by T^* the odd CAT of Figure 3.11(f), i.e., T^* consists of three neutral 2-cycles whose non-neutral nodes are joined in a 3-cycle.

Proposition 40 *If $n = 6$ and $T = T^*$, then the odd CAT inequality (10) does not define a facet of P_A or P .*

Here is the main result of this section.

Theorem 41 *For all $n \geq 6$ the odd CAT inequality (10) defines a facet of P_A and P (except in case $n = 6$ and $T = T^*$).*

The proof of this theorem relies heavily on the following property of all nontrivial facet defining inequalities for P [62].

Theorem 42 *Suppose $\alpha x \leq \alpha_0$ defines a facet of P . Let X be the matrix whose rows are the incidence vectors of tours in G satisfying $\alpha x = \alpha_0$, and for any $p, q \in V$ let $A(p, q)$ be the set of columns (i, p) , $i \in V \setminus \{p, q\}$, and (q, j) , $j \in V \setminus \{p, q\}$, of X . Then the submatrix of X consisting of the columns in $A(p, q)$ has rank $|A(p, q)| - 1$; i.e., $2n - 3$ if $p = q$ and $2n - 5$ if $p \neq q$.*

7. Source-destination inequalities

We now describe a class of facet inducing inequalities introduced in [73] that properly generalizes several known classes of facets of the ATS

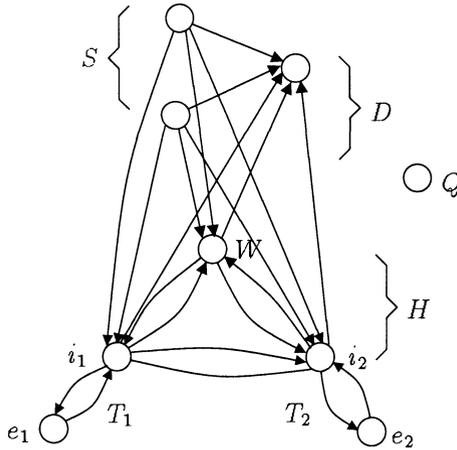


Figure 3.12. The support graph of a primitive SD inequality.

polytope; the reader is referred to [73] for further details. We first introduce the primitive (i.e., clone free) members of the family.

Let (S, D, W, I, E, Q) be a partition of V with the following properties: S is the (possibly empty) set of *sources*; D is the (possibly empty) set of *destinations*; $H := W \cup I$ is the (nonempty) *handle*, with $0 \leq |W| \leq 1$ and $|I| = s \geq 1$; $I := \{i_1, \dots, i_s\}$; $E := \{e_1, \dots, e_s\}$; $T_j := \{i_j, e_j\}$ for $j = 1, \dots, s$ are the *teeth*; $0 \leq |Q| \leq 1$; and $|S| + |D| + s$ is odd.

Theorem 43 *The (primitive) source-destination (SD) inequality*

$$x(S \cup H, D \cup H) + \sum_{j=1}^s x(T_j, T_j) \leq \frac{1}{2}(|S| + |D| + 2|H| + s - 1) \quad (11)$$

is valid for P and \tilde{P} and has Chvátal rank 1.

Proof: Multiplying the following inequalities by $\frac{1}{2}$, adding them up and rounding down all coefficients produces (11): $x(i, V) \leq 1$ for $i \in S \cup H$, $x(V, j) \leq 1$ for $j \in D \cup H$, and $x(T_j, T_j) \leq 1$ for $j = 1, \dots, s$. ■

Figure 3.12 illustrates the support graph of an SD inequality on 9 nodes. Note that the nodes in S , although interchangeable, are not clones, because the arcs joining them to each other have coefficients equal to 0, instead of 1.

A primitive SD inequality becomes a comb inequality when $|S| = |D| = 0$ and $s \geq 3$, a C2 inequality when $|S| = |D| = 1$ and $s \geq 3$, a T_2 inequality when $|S| = |D| = 1$, $W = \emptyset$ and $s = 1$, and an odd CAT

inequality when $|S| = |D|$ (with $W \neq \emptyset$ only if $s \geq 5$, or $|S| \geq 2$, or $|S| = 1$ and $s \geq 3$).

We now address the cases in which SD inequalities are *not* facet inducing.

Theorem 44 *The primitive SD inequalities do not define facets of P in the following 3 pathological cases, all arising when $|S| = |D|$ and $n \leq 6$:*

- (a) $s = 3, S = D = W = Q = \emptyset$ (hence $n = 6$),
- (b) $n \geq 5, s = 1, |S| = |D| = 1$ (hence $n \in \{5, 6\}$),
- (c) $s = 1, S = D = \emptyset, |W| = |Q| = 1$ (hence $n = 4$).

Moreover, when $|S| \neq |D|$ we have the following result.

Theorem 45 *Let $\alpha x \leq \alpha_0$ be any primitive SD inequality with $\||S| - |D|\| > \max\{s - 3, 0\}$. Then $\alpha x \leq \alpha_0$ does not define a facet of P .*

On the other hand, in all cases not covered by the previous theorems the SD-inequalities do define facets of P :

Theorem 46 *Let $\alpha x \leq \alpha_0$ be any primitive SD inequality, different from those of Theorem 44 and satisfying $\||S| - |D|\| \leq \max\{s - 3, 0\}$. Then $\alpha x \leq \alpha_0$ defines a facet of both P and \bar{P} .*

We now consider the application to the primitive SD inequality $\alpha x \leq \alpha_0$ of the clique lifting procedure described in Section 3.1. To this end it is sufficient to compute the values δ_k defined in Theorem 23, thus obtaining $\delta_k = 0$ if $k \in Q$, $\delta_k = 2$ if $k \in I$, and $\delta_k = 1$ otherwise. This leads to the following (clique lifted) SD inequality:

$$\begin{aligned} x(S \cup H, H \cup D) &+ \sum_{i=1}^s x(T_i, T_i) + \sum_{i=1}^{\sigma} x(S_i, S_i) + \sum_{i=1}^{\delta} x(D_i, D_i) \\ &\leq |H| + \sum_{i=1}^s (|T_i| - 1) + \sum_{i=1}^{\sigma} (|S_i| - 1) + \sum_{i=1}^{\delta} (|D_i| - 1) \\ &\quad + \frac{\sigma + \delta - s - 1}{2} \end{aligned}$$

where (H, T_1, \dots, T_s) defines a comb with a possibly even number s of teeth, S and D are disjoint subsets of $V \setminus (H \cup T_1 \cup \dots \cup T_s)$ which are partitioned properly into $S = \cup_{i=1}^{\sigma} S_i$ and $D = \cup_{i=1}^{\delta} D_i$, and $\sigma + \delta + s$ is an odd number.

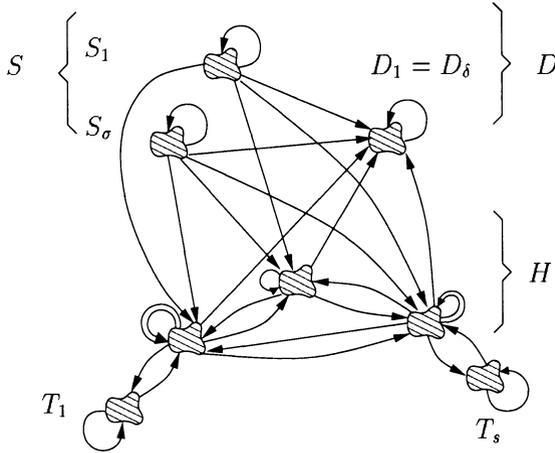


Figure 3.13. A general SD inequality

A general inequality of this type is illustrated in Figure 3.13, where the shaded regions represent cliques, while single and double lines represent coefficients of 1 and 2, respectively.

It can readily be shown that SD inequalities properly generalize the T_k , comb, C2, and odd CAT inequalities. Indeed, the T_k inequalities can be obtained starting from a primitive SD inequality with $s = 1$, $|S| = |D| = 1$, $W = \emptyset$ and introducing clones of nodes e_1 and $q \in Q$. Comb inequalities arise trivially from the case $S = D = \emptyset$ through the cloning of any node. C2 inequalities can be derived from the case $|S| = |D| = 1$ and $s \geq 3$, by adding clones of the nodes in $V \setminus (S \cup D)$. Finally, the odd CAT inequalities are obtainable from primitive SD inequalities with $|S| = |D|$ by introducing clones of $w \in W$ and at most one clone for each node in $S \cup D$.

We now address the possibility that two given (facet-defining) SD inequalities define the same facet of P , i.e., that they are equivalent. To this end we derive and compare their canonical form, as described at the end of Section 5. In view of Theorems 31 and 33 we will assume that the two SD inequalities to be compared are primitive. Then let $\alpha x \leq \alpha_0$ be any primitive SD inequality associated with the node partition $(S, D, W, Q, I := \{i_1, \dots, i_s\}, E := \{e_1, \dots, e_s\})$ and with tooth set $T := \{\{i_j, e_j\} : j = 1, \dots, s\}$, and consider any primitive SD inequality $\alpha' x \leq \alpha'_0$, different from $\alpha x \leq \alpha_0$ and associated with the node partition $(S', D', W', Q', I' := \{i'_1, \dots, i'_s\}, E' := \{e'_1, \dots, e'_s\})$ and with tooth set $T' := \{\{i'_j, e'_j\} : j = 1, \dots, s'\}$.

Theorem 47 *Two SD inequalities $\alpha x \leq \alpha_0$ and $\alpha'x \leq \alpha'_0$ are equivalent if $S' = D$, $D' = S$, $Q' = W$, $W' = Q$, $I' = E$, $E' = I$, and $T' = T$.*

The equivalence established in Theorem 47 was known earlier for comb inequalities, i.e., when $|S| = |D| = 0$; note that it also holds for the C2 and odd CAT inequalities.

A known case of equivalence not completely covered by Theorem 47 arises when $n = 4$ and $|S| = |D| = s = 1$, i.e. when considering T_2 inequalities on 4 nodes (see Figure 3.9). We give the corresponding result for the sake of completeness.

Theorem 48 *Let $n = 4$, and assume $|S| = |D| = s = 1$ and $|S'| = |D'| = s' = 1$. W.l.o.g. let $S = \{1\}$, $i_1 = 2$, $D = \{3\}$, and $e_1 = 4$. Then $\alpha'x \leq \alpha'_0$ is equivalent to $\alpha x \leq \alpha_0$ if and only if one of the following 4 cases occurs: (1) $S' = \{1\}$, $i'_1 = 2$, $D' = \{3\}$, and $e'_1 = 4$; (2) $S' = \{4\}$, $i'_1 = 1$, $D' = \{2\}$, and $e'_1 = 3$; (3) $S' = \{3\}$, $i'_1 = 4$, $D' = \{1\}$, and $e'_1 = 2$; (4) $S' = \{2\}$, $i'_1 = 3$, $D' = \{4\}$, and $e'_1 = 1$.*

We now show that, with the one exception of the pathology pointed out in Theorem 48, all the equivalences among facet-defining primitive SD inequalities are covered by Theorem 47. We will do this by transforming the generic primitive facet-inducing SD inequality $\alpha x \leq \alpha_0$ to its canonical form $\beta x \leq \beta_0$. Following the procedure stated at the end of Section 5, for all $h \in V$ we compute:

$$\alpha_{hh} := \begin{cases} 0 & \text{if } h \in Q, \\ 2 & \text{if } h \in I, \\ 1 & \text{otherwise,} \end{cases} \quad \Delta := \sum_{h=1}^n \alpha_{hh} = 3s + |W| + |S| + |D|,$$

$$r_h, c_h := \begin{cases} 1 + |D| + |W| + s, & 1 & \text{if } h \in S, \\ 1, & 1 + |S| + |W| + s & \text{if } h \in D, \\ |W| + s + |D|, & |W| + s + |S| & \text{if } h \in W, \\ 0, & 0 & \text{if } h \in Q, \\ |W| + |D| + s + 2, & |W| + |S| + s + 2 & \text{if } h \in I, \\ 2, & 2 & \text{if } h \in E. \end{cases}$$

Recall that $|S| + |D| + |W| + |Q| + 2s = n$. The canonical form $\beta x \leq \beta_0$ is then computed as $\beta_{ij} := \sigma(\Delta + n\alpha_{ij} - r_i - c_j - \varepsilon)$, $i, j \in V$, where $\sigma > 0$ and ε are defined in the normalization step. It is therefore not difficult to prove the following:

Theorem 49 *Let $n \geq 5$. Two distinct facet-inducing primitive SD inequalities are equivalent only in the case covered by Theorem 47.*

8. Lifted cycle inequalities

In this section we investigate the family of lifted cycle inequalities for the ATS polytope, and establish several properties that earmark it as one of the most important among the families of asymmetric facet inducing inequalities known to date.

Unless otherwise stated, all directed cycles considered in this section are simple. Let $S \subset N$, $S = \{i_1, i_2, \dots, i_s\}$, and let

$$C := \{(i_1, i_2), (i_2, i_3), \dots, (i_{s-1}, i_s), (i_s, i_1)\}$$

be a directed cycle visiting all the nodes in S . For the sake of simplicity, we will use i_{j+1} and i_{j-1} to denote the successor and the predecessor, respectively, of node i_j in the cycle (hence $i_{s+1} \equiv i_1$ and $i_0 \equiv i_s$). A *chord* of C is an arc $(i_h, i_k) \in A$ such that $i_k \neq i_{h+1}$. Let R denote the set of chords of C . For every subset $F \subseteq A$, let $\tilde{P}(F) := \{x \in \tilde{P} : x_a = 0 \forall a \in F\}$. It is well known [396] that $x(C) \leq |C| - 1$ defines a facet of the polytope $\tilde{P}(R)$. Moreover, let $R = \{a_1, a_2, \dots, a_m\}$; then the *lifted cycle inequality*

$$\alpha x := x(C) + \sum_{i=1}^m \alpha_{a_i} x_{a_i} \leq \alpha_0 := |C| - 1$$

defines a facet of \tilde{P} [396, 405] where the lifting coefficients α_{a_i} ($i = 1, \dots, m$) are sequentially computed as the maximum value such that inequality $\alpha x \leq \alpha_0$ is valid for $\tilde{P}(\{a_{i+1}, \dots, a_m\})$. It is well known that (a) different sequences $\{a_i; i = 1, \dots, m\}$ may lead to different inequalities $\alpha x \leq \alpha_0$, and (b) the value of a given coefficient is largest if lifted first, and is a monotone nonincreasing function of its position in the lifting sequence (with the position of the other coefficients kept fixed). In the case of lifted cycle inequalities $\alpha x \leq \alpha_0$, it is also easily seen that $\alpha_{a_i} \in \{0, 1, 2\}$ for $i = 1, \dots, m$.

We will study the lifted cycle inequalities on \tilde{P} rather than P , since \tilde{P} is full dimensional, and the following result holds by virtue of Corollary 19.

Theorem 50 *Any lifted cycle inequality for \tilde{P} whose defining cycle has at most $n - 3$ arcs, induces a facet of P .*

Unless otherwise stated, all results in the present section are from [75].

8.1. Two-liftable chord sets

In this section we characterize those sets of chords that can get a coefficient 2 in the lifting process. We first note that a given lifted cycle

inequality $\alpha x \leq \alpha_0$ obtained, say, via the chord sequence a_1, \dots, a_m , can always be obtained via an equivalent chord sequence in which all the chords with coefficient 2 appear (in any order) at the beginning of the sequence, while all the chords with coefficient 0 appear (in any order) at the end of the sequence. We call any such chord sequence *canonical*. Indeed, consider swapping the positions in the lifting sequence of two consecutive chords. Clearly, either (a) their coefficients remain unchanged, or (b) the coefficient of the chord moved to the left increases and the other one decreases. Case (b) cannot occur when a chord with coefficient 2 is moved to the left, or a chord with coefficient 0 is moved to the right. Therefore a sequence of swaps of the above type leads to the desired canonical form.

For any arc set F , we denote by $V(F)$ the set of nodes spanned by F . Given a cycle C and a chord (i, j) , we denote by C_{ij} the shorter of the two cycles contained in $C \cup \{(i, j)\}$. A given set $H \subset R$ is called *2-liftable* if there exists a chord sequence producing a lifted cycle inequality $\alpha x \leq \alpha_0$ such that $\alpha_a = 2$ for all $a \in H$. Because of the above property, such a sequence can w.l.o.g. be assumed to be canonical. It then follows that H is 2-liftable if and only if $\alpha x := x(C) + 2x(H) \leq |C| - 1$ is a valid inequality for $\tilde{P}(R \setminus H)$. We will show that in order to impose this condition it is necessary and sufficient to forbid the presence in H of certain patterns of chords.

Two distinct arcs (i, j) and (u, v) are called *compatible* when there exists a tour containing both, i.e., when $i \neq u$, $j \neq v$, and (i, j) and (u, v) do not form a 2-cycle. Two arcs that are not compatible are called *incompatible*. Given a chord (i_a, i_b) of C , we call *internal* w.r.t. (i_a, i_b) the nodes i_b, i_{b+1}, \dots, i_a ; *external* the nodes i_{a+1}, \dots, i_{b-1} . Thus the internal nodes w.r.t. (i_a, i_b) are those of $V(C_{i_a i_b})$, and the external ones are those of $V(C) \setminus V(C_{i_a i_b})$. Given two chords (i_a, i_b) and (i_c, i_d) , we say that (i_c, i_d) *crosses* (i_a, i_b) if they are compatible and nodes i_c and i_d are not both internal or both external w.r.t. (i_a, i_b) ; see Figure 3.14 for an illustration. Note that (i_c, i_d) crosses (i_a, i_b) if and only if (i_a, i_b) crosses (i_c, i_d) , i.e., the property is symmetric. Two chords that do not cross each other are called *noncrossing*. Thus all incompatible pairs are noncrossing.

We define a *noose* in $C \cup R$ as a simple alternating (in direction) cycle $Q := \{a_1, b_1, a_2, b_2, \dots, a_q, b_q\}$ of $2q \geq 4$ distinct arcs $a_i \in R$ and $b_i \in C$ ($i = 1, \dots, q$), in which all adjacent arcs in the sequence (including b_q and a_1) are incompatible and all chords are pairwise noncrossing (see Figure 3.15).

Theorem 51 *A chord set $H \subset R$ is 2-liftable if and only if $C \cup H$ contains no pair of crossing chords and no noose.*

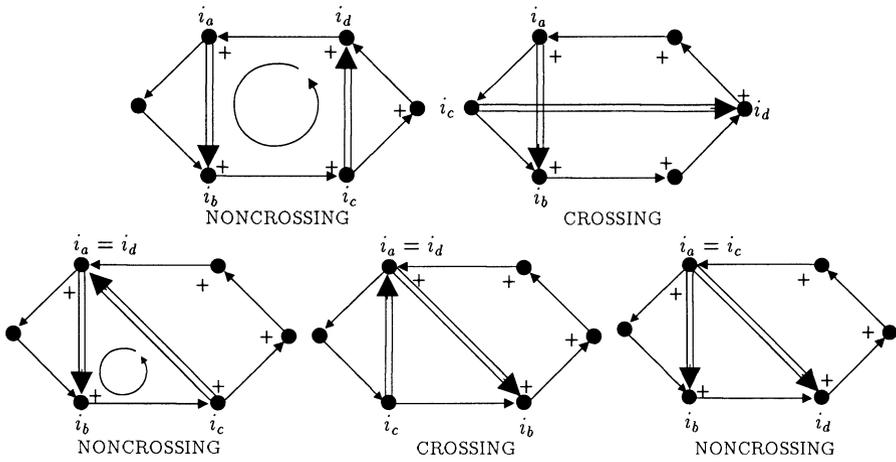


Figure 3.14. Crossing and noncrossing chords. The internal nodes w.r.t. (i_a, i_b) are marked with $+$, and the chords are drawn in double lines.

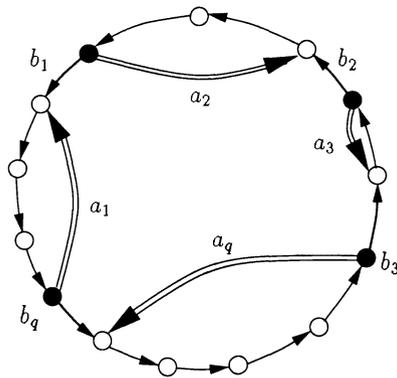


Figure 3.15. A nose Q with $q = 4$.

An immediate consequence of Theorem 51 is that the set of 2-liftable chords is never empty for $|C| \geq 3$. It then follows that the subtour elimination inequalities, in which every chord has a coefficient 1, are not sequentially lifted inequalities (they can be obtained from the corresponding cycle inequality by simultaneous lifting). Any chord that is lifted first in a sequential lifting procedure must get a coefficient 2.

We might note at this point that we have touched upon an important point of difference between the symmetric and asymmetric TS polytopes. In the case of the STS polytope, the subtour elimination inequalities are the only kinds of lifted cycle inequalities: whichever chord is lifted first, it gets a coefficient of 1, and so does the chord that is lifted last.

Assigning a chord (i, j) the coefficient $\alpha_{ij} = 2$ forces to 0 the coefficients of several other chords.

Theorem 52 *Let (i_a, i_b) be a chord of C such that $\alpha_{i_a i_b} = 2$. Then the following chords must have coefficient 0 (see Figure 3.16):*

- (i) (i_j, i_{a+1}) for all $j = b, b + 1, \dots, a - 1$;
- (ii) (i_{b-1}, i_ℓ) for all $\ell = b + 1, b + 2, \dots, a$.

Corollary 53 *If the chord (i_a, i_{a+2}) has coefficient 2, then all chords incident with node i_{a+1} must have coefficient 0.*

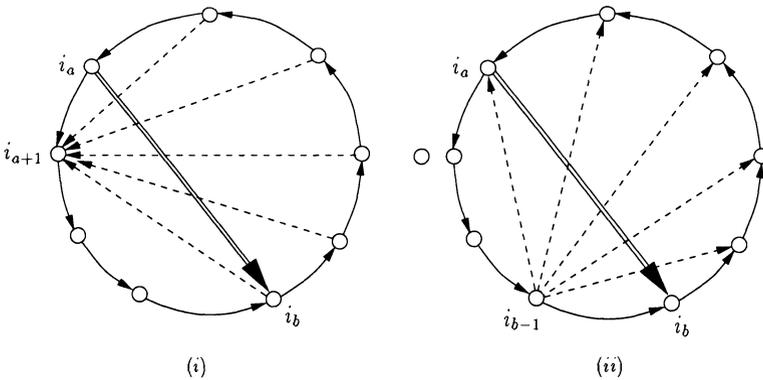


Figure 3.16. The dashed lines represent chords with coefficient 0.

8.2. Maximally 2-lifted cycle inequalities

Given a lifted cycle inequality with a given set of 2-liftable chords and a corresponding set of chords with coefficients forced to 0, the size of the

remaining coefficients depends in general on the lifting sequence. Next we characterize a class of lifted cycle inequalities whose 0-1 coefficients are largely sequence-independent.

A set Q_2 of 2-liftable chords is termed *maximal* if no set of the form $Q_2 \cup \{(i, j)\}$, $(i, j) \in R \setminus Q_2$, is 2-liftable. A lifted cycle inequality whose set of chords with coefficient 2 is maximal 2-liftable, will be called a *maximally 2-lifted cycle inequality*.

Proposition 54 *The maximum cardinality of a 2-liftable chord set is $|C| - 2$.*

Notice that not all sequentially lifted cycle inequalities are maximally 2-lifted. Examples of (facet inducing) lifted cycle inequalities having a single chord with coefficient 2 are shown in Figure 3.17. (For the first graph of that figure, the appropriate lifting sequence is $(3, 2)$, $(1, 4)$, $(3, 1)$, $(4, 2)$ etc.).

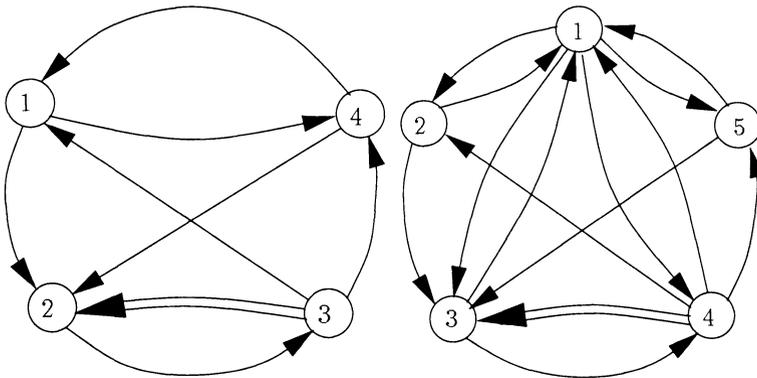


Figure 3.17. Support graphs of two lifted cycle inequalities having a single chord with coefficient 2.

On the other hand, we can identify some large classes of maximally 2-lifted cycle inequalities with useful properties. The next theorem gives a sufficient condition for a given inequality related to a cycle C to be a (facet defining) lifted cycle inequality for \tilde{P} .

Theorem 55 *Let $\alpha x \leq \alpha_0$ be an inequality with $\alpha_{ij} = 1$ for all arcs (i, j) of a given cycle C of length $|C| = \alpha_0 + 1 \leq n - 1$, where $\alpha_{ij} \in \{0, 1, 2\}$ for all chords (i, j) of C , and $\alpha_{ij} = 0$ for all other arcs. Let $Q_t := \{(i, j) \in R : \alpha_{ij} = t\}$ for $t = 0, 1, 2$, and assume the following conditions hold:*

- (a) Q_2 is a maximal 2-liftable set;

(b) all 0 coefficients are maximal, i.e., $\alpha_{ij} = 0$ implies the existence of $x^* \in \tilde{P}$ such that $\alpha x^* = \alpha_0$ and $x_{ij}^* = 1$;

(c) the inequality $\alpha x \leq \alpha_0$ is valid for \tilde{P} .

Then $\alpha x \leq \alpha_0$ is a maximally 2-lifted cycle inequality, hence facet defining for \tilde{P} .

A particularly friendly (and, it turns out, rich) class of lifted cycle inequalities is that for which $Q_0 := \{(i, j) \in R : \alpha_{ij} = 0\}$ is just the set of chords whose coefficients are forced to 0 by the conditions of Theorem 52. For this class condition (b) holds automatically, and the only condition to be checked is the inequality validity (c).

Corollary 56 *Let $\alpha x \leq \alpha_0$ and let $Q_t := \{(i, j) \in R : \alpha_{ij} = t\}$ for $t = 0, 1, 2$. If Q_2 is a maximal 2-liftable set and Q_0 is the set whose coefficients are forced to 0 by the conditions of Theorem 52, then $\alpha x \leq \alpha_0$ is a (facet defining) lifted cycle inequality for \tilde{P} if and only if it is valid for \tilde{P} .*

One familiar subclass of this class is that of the D_k^+ and D_k^- inequalities. Another subclass will be introduced in the sequel.

8.3. Maximally 2-lifted cycle inequalities of rank 1

In this section we introduce two new classes of lifted cycle inequalities of Chvátal-rank 1. We start by pointing out that the pattern of 2-liftable chords in a rank 1 inequality has to satisfy an additional condition (besides those required for 2-liftability): it has to define a nested family of node sets in the following sense. A family \mathcal{F} of sets is *nested* if for every $S_1, S_2 \in \mathcal{F}$, either $S_1 \subseteq S_2$, or $S_2 \subseteq S_1$, or $S_1 \cap S_2 = \emptyset$. Given a cycle C and a chord (i_a, i_b) of C , recall that $C_{i_a i_b}$ denotes the shorter of the two cycles contained in $C \cup \{(i_a, i_b)\}$.

Theorem 57 *Let $\alpha x \leq \alpha_0$ be a lifted cycle inequality with cycle C , chord set R and $Q_2 := \{(i, j) \in R : \alpha_{ij} = 2\}$. If $\alpha x \leq \alpha_0$ is of Chvátal rank 1, then $V(C)$ and the node sets $V(C_{i_a i_b})$ for all $(i_a, i_b) \in Q_2$ form a nested family.*

An implication of the above theorem is that rank-1 lifted cycle inequalities form a very special subclass indeed: for every $(i_a, i_b) \in Q_2$, the path from i_b to i_a in C never meets the tail of any chord in Q_2 before meeting its head. This implies, among other things, that Q_2 cannot contain a 2-cycle.

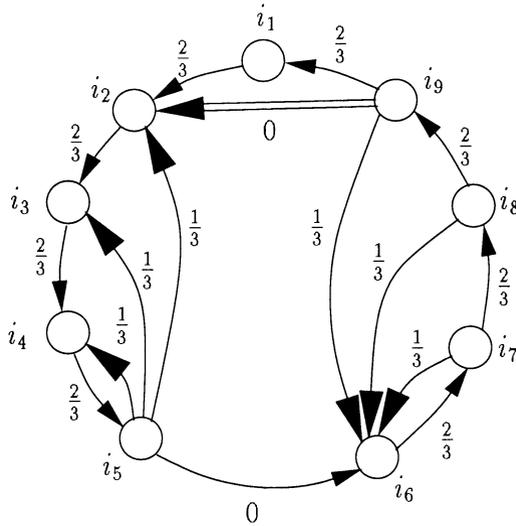


Figure 3.18. Counterexample for the converse of Theorem 57.

Theorem 57 gives a necessary condition for $\alpha x \leq \alpha_0$ to be of Chvátal rank 1. Interestingly, this condition is not sufficient, as shown by the following example. Consider the cycle C and the 2-liftable set of chords of Figure 3.18. One can easily see that node sets $V(C_{i_a i_b})$ for $(i_a, i_b) \in Q_2$ form a nested family, as required in Theorem 57. However, no lifted cycle inequality $\alpha x \leq \alpha_0$ with the 2-chord pattern of Figure 3.18 can be of rank 1, as certified by the point x^* with $x_{ij}^* := \frac{2}{3}$ for all $(i, j) \in C \setminus \{(i_5, i_6)\}$, $x_{ij}^* := \frac{1}{3}$ for all $(i, j) \in Q_2 \setminus \{(i_9, i_2)\}$, $x_{ij}^* := 0$ for all other arcs. Indeed, x^* satisfies all degree and subtour elimination inequalities, but $\alpha x^* = \frac{28}{3} > \alpha_0 + 1 = 9$. This implies [688] that $\alpha x \leq \alpha_0$ cannot be of rank 1.

We now introduce two new large classes of maximally 2-lifted cycle inequalities of Chvátal rank 1.

Theorem 58 *Let C be the cycle visiting in sequence the nodes i_1, \dots, i_k for some $4 \leq k \leq n - 1$. Then the shell inequality*

$$\alpha x := x(C) + \sum_{\substack{3 \leq j \leq k \\ j \text{ odd}}} x(\{i_{j+1}, \dots, i_k, i_1\}, i_j) + \sum_{j=3}^k x_{i_1 i_j} + \sum_{\substack{3 \leq j \leq k-1 \\ j \text{ odd}}} x_{i_{j+1} i_j} \leq k - 1 =: \alpha_0$$

is a rank-1 maximally 2-lifted cycle inequality, hence facet defining for \tilde{P} . (See Figure 3.19.)

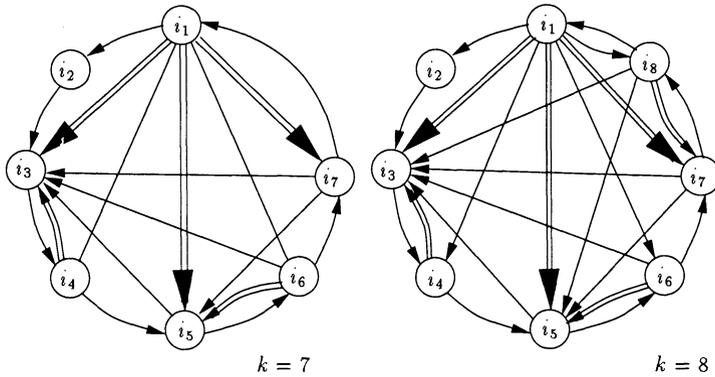


Figure 3.19. Support graphs of two shell inequalities.

Theorem 59 Let $w, a_1, \dots, a_{k_a}, b_1, \dots, b_{k_b}$, be distinct nodes with $k_a, k_b \geq 1, k_b \in \{k_a, k_a + 1\}$, and $k_a + k_b + 1 \leq n - 1$. Let C be the directed cycle visiting, in sequence, nodes $w, b_1, b_2, \dots, b_{k_b}, a_{k_a}, a_{k_a-1}, \dots, a_1, w$, and define $F := \bigcup_{i=1}^{k_a} \{(a_i, b_j) : i \leq j \leq i + 1, j \leq k_b\}$. Then the fork inequality

$$\alpha x := x(C) + x(F) + \sum_{i=1}^{k_a} \sum_{j=1}^{k_b} x_{a_i b_j} + \sum_{i=1}^{k_a-1} x(a_i, \{a_{i+1}, \dots, a_{k_a}\}) + \sum_{j=1}^{k_b-1} x(\{b_{j+1}, \dots, b_{k_b}\}, b_j) \leq k_a + k_b =: \alpha_0$$

is a rank-1 maximally 2-lifted cycle inequality, hence facet defining for \tilde{P} (see Figure 3.20).

8.4. Maximally 2-lifted cycle inequalities of unbounded rank

We next establish an important property of the family of lifted cycle inequalities.

Theorem 60 The family of lifted cycle inequalities contains members of unbounded Chvátal rank.

Proof: Let C be a cycle with node set $i_1, \dots, i_k, k \geq 8$ even, and consider the nonmaximal 2-liftable chord set $Q := \{(i_1, i_3), (i_3, i_5), \dots, (i_{k-1}, i_1)\}$. Let $\alpha x \leq \alpha_0$ be any lifted cycle inequality associated with C and such that $\alpha_{ij} = 2$ for all $(i, j) \in Q$. We claim that for any subset S of

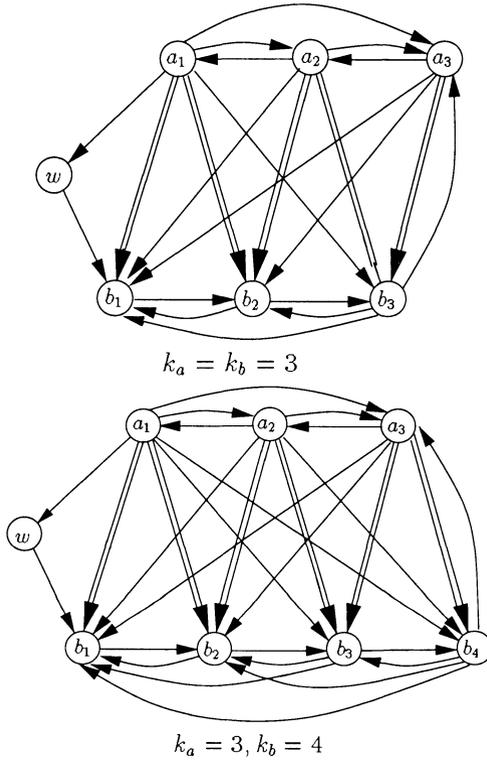


Figure 3.20. Support graphs of two fork inequalities.

the set $V_{\text{even}} := \{i_2, i_4, \dots, i_k\}$ of even nodes, the subtour elimination inequality associated with $V(C) \setminus S$ must have a positive multiplier in any Chvátal derivation of $\alpha x \leq \alpha_0$. Since the number of subsets S is exponential in k (which in turn is bounded only by n), it then follows from Chvátal, Cook, and Hartmann [198] that for any $M > 0$ there exists a lifted cycle inequality in a sufficiently large digraph, whose Chvátal rank is at least M .

To prove the claim, we assume by contradiction that there exists a Chvátal derivation of $\alpha x \leq \alpha_0$ in which the SEC associated with some $S^* := V(C) \setminus S$ with $S \subseteq V_{\text{even}}$ has 0 multiplier. Then $\alpha x \leq \alpha_0$ must be valid for the polytope \tilde{P}^* defined as the convex hull of points $x \in \{0, 1\}^A$ satisfying the degree inequalities and all the SEC's except for the one associated with S^* . But here is a point $x^* \in \tilde{P}^*$ with $\alpha x^* = |C| > \alpha_0$ that violates $\alpha x \leq \alpha_0$, whose support A^* is constructed as follows: start with $A^* := C$ and then, for each $i_j \in S$, remove from A^* the two arcs incident with i_j (having coefficient 1), and add the arc $(i_{j-1}, i_{j+1}) \in Q$ (which has coefficient 2). ■

Next we introduce a large family of maximally 2-lifted cycle inequalities whose set Q_2 of 2-lifted chords is of the type used for the proof of Theorem 60, and which therefore contains members with unbounded Chvátal rank. Notice that, in addition, these inequalities satisfy the condition of Corollary 56.

Theorem 61 *Let C be the cycle visiting in sequence the nodes i_1, i_2, \dots, i_{4k} for some integer $k \geq 2$ satisfying $4k \leq n - 1$. Further, let $S_1 := \{i_j \in N(C) : j \text{ is odd}\}$, and $C_1 := \{(i_1, i_3), (i_3, i_5), \dots, (i_{4k-1}, i_1)\}$. Then the curtain inequality*

$$\alpha x := x(C) + x(S_1, S_1) + x(C_1) + \sum_{\substack{j=3 \\ j \text{ odd}}}^{2k-1} (x_{i_j i_{4k-j+2}} + x_{i_{4k-j+2} i_j}) \leq 4k - 1 =: \alpha_0$$

is a maximally 2-lifted, hence facet defining, cycle inequality for \tilde{P} .

Proof: Let $Q_t := \{(i, j) : \alpha_{ij} = t\}$ for $t \in \{0, 1, 2\}$. Clearly, Q_2 has no crossing chords and no nooses, so it is 2-liftable. Also, $|Q_2| = |C|/2 + (|C| - 4)/2 = |C| - 2$, hence Q_2 is maximal. Further, Q_0 consists precisely of those chords whose coefficient is forced to 0 by the conditions of Corollary 53. Hence by Corollary 56, we only have to show that the curtain inequality is valid for \tilde{P} . From the properties of the 2-liftable chord set Q_2 , the inequality $x(C) + 2x(Q_2) \leq 4k - 1$ is valid for $\tilde{P}(R \setminus Q_2)$. Thus the

certain inequality is satisfied by all $x \in \tilde{P}$ such that $x(Q_1) = 0$. Now let $x \in \tilde{P}$ be such that $x(Q_1) \geq 1$. Then subtracting this inequality from the sum of the $2k$ indegree inequalities and the $2k$ outdegree inequalities for the odd nodes $i_1, i_3, \dots, i_{4k-1}$, produces an inequality $\beta x \leq 4k - 1$, with $\beta \geq \alpha$. This proves the validity of the curtain inequality for \tilde{P} . ■

The pattern of chords with coefficient 2 in a curtain inequality is illustrated in Figure 3.21, where odd and even nodes are marked by + and -, respectively. The chords with coefficient 1 are all those (not shown in the figure) joining pairs of odd nodes.

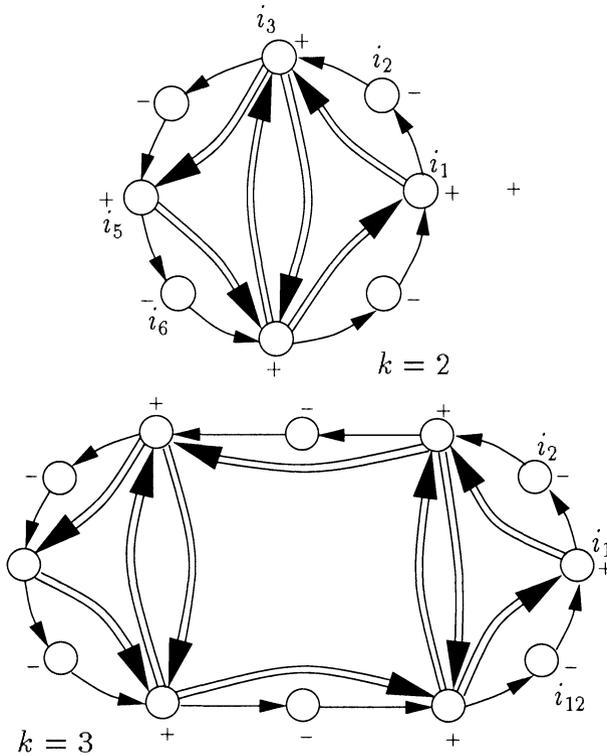


Figure 3.21. 2-liftable chord sets of curtain inequalities.

The class of curtain inequalities can be extended to cycles of length $|C| \not\equiv 0 \pmod{4}$. The corresponding graphs for $|C| = 5, 6, 7$ are shown in Figure 3.22.

For the cases $|C| \equiv 1 \pmod{4}$ and $|C| \equiv 3 \pmod{4}$ we have the following.

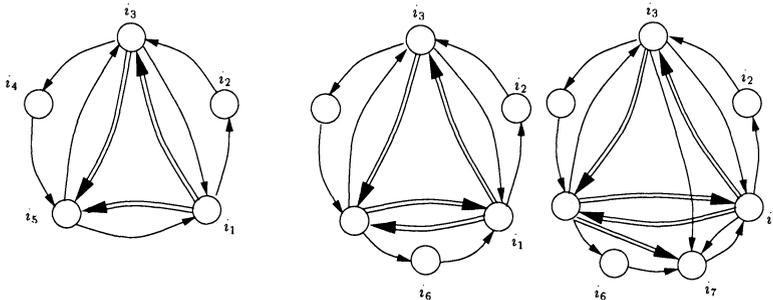


Figure 3.22. Support graphs of curtain inequalities for $|C| = 5, 6, 7$.

Theorem 62 Let C be the cycle visiting in sequence the nodes i_1, \dots, i_{4k+1} for some integer $k, 2 \leq k \leq (n - 2)/4$. Further, let $S_1 := \{i_j \in V(C) : j \text{ is odd}\}$, and $P_1 := \{(i_1, i_3), (i_3, i_5), \dots, (i_{4k-1}, i_{4k+1})\}$. Then the curtain inequality

$$\alpha x := x(C) + x(S_1, S_1) + x(P_1) + \sum_{\substack{j=3 \\ j \text{ odd}}}^{2k-1} (x_{i_j i_{4k-j+2}} + x_{i_{4k-j+2} i_j}) + x_{i_1 i_{4k+1}} \leq 4k =: \alpha_0$$

is a maximally 2-lifted, hence facet defining, inequality for \tilde{P} .

Proof: Parallels the proof of Theorem 61. As in that case, we only have to show that the inequality is satisfied by all $x \in \tilde{P}$ such that $x(Q_1) \geq 1$. Let x have this property. Then adding

- the outdegree inequalities for nodes $i_1, i_3, \dots, i_{4k-1}$
- the indegree inequalities for node $i_3, i_5, \dots, i_{4k+1}$
- 1/2 times the outdegree inequality for node i_{4k+1}
- 1/2 times the indegree inequality for node i_1
- 1/2 times the inequality $-x(Q_1) \leq -1$

we obtain an inequality $\beta x \leq 4k + 0.5$, with $\beta \geq \alpha$. Rounding down the coefficients on both sides then yields an inequality that implies $\alpha x \leq 4k$.

■

Theorem 63 Let C be the cycle visiting in sequence the nodes i_1, \dots, i_{4k+3} for some integer $k, 2 \leq k \leq (n - 4)/4$. Further, let $S_1 :=$

$\{i_j \in N(C) : j \text{ is odd}\}$, and $P_1 := \{(i_1, i_3), (i_3, i_5), \dots, (i_{4k+1}, i_{4k+3})\}$.
Then the curtain inequality

$$\alpha x := x(C) + x(S_1, S_1) + x(P_1) + \sum_{\substack{j=1 \\ j \text{ odd}}}^{2k-1} (x_{i_j i_{4k-j+2}} + x_{i_{4k-j+2} i_j}) - \sum_{\substack{j=3 \\ j \text{ odd}}}^{4k+1} x_{i_{4k+3} i_j} \leq 4k + 2 := \alpha_0$$

is a maximally 2-lifted, hence facet defining, inequality for \tilde{P} .

Proof: As in the case of Theorem 61, Q_2 can easily be seen to be a maximally 2-liftable chord set. Also, from Theorem 52, all chords incident from or to even nodes have 0 coefficients. Further, from the same theorem as it applies to the 2-chord (i_{4k+1}, i_1) , all chords incident from i_{4k+3} have 0 coefficients. Finally, to prove validity, the inequality can be shown to be satisfied by all $x \in \tilde{P}$ such that $x(Q_1) \geq 1$ by adding (1) the outdegree inequalities for nodes $i_1, i_3, \dots, i_{4k+1}$; (2) the indegree inequalities for nodes $i_1, i_3, \dots, i_{4k+3}$; and (3) the inequality $-x(Q_1) \leq -1$. ■

Finally, for the case $|C| \equiv 2 \pmod{4}$ we have a stronger result, i.e., we can identify a larger class of facet defining inequalities that contains as a special case the curtain inequality with $|C| \equiv 2 \pmod{4}$.

Theorem 64 *Let C be an even length cycle visiting nodes $i_1, \dots, i_{|C|}$, with $|C| \leq n - 1$. Define the cycle $C_1 := \{(i_1, i_3), (i_3, i_5), \dots, (i_{|C|-1}, i_1)\}$ and let S_1 be the node set of C_1 . Then for any maximally 2-liftable chord set Q_2 containing C_1 , the inequality*

$$\alpha x := x(C) + x(S_1, S_1) + x(Q_2) \leq |C| - 1$$

is a maximally 2-lifted, hence facet defining, inequality for \tilde{P} .

Proof: Since Q_2 is maximally 2-liftable, every $x \in \tilde{P}(R \setminus Q_2)$ satisfies $x(C) + 2x(Q_2) \leq |C| - 1$. Moreover, Q_0 consists precisely of those chords whose coefficient is forced to 0 by the conditions of Theorem 52. Thus by Corollary 56, we only need to prove that the inequality of the theorem is valid for \tilde{P} . Clearly, all $x \in \tilde{P}$ such that $x(Q_1) = 0$, where Q_1 is the set of chords with coefficient 1, satisfies the inequality. Now let $x \in \tilde{P}$ be such that $x(Q_1) \geq 1$, and note that $|S_1| = |C|/2$. Then adding up (1) the outdegree inequalities for nodes $i \in S_1$; (2) the indegree inequalities for nodes $i \in S_1$; and (3) the inequality $-x(Q_1) \leq -1$, we obtain an inequality $\beta x \leq |C| - 1$, where $\beta \geq \alpha$. □

The curtain inequality for $|C| \equiv 2 \pmod{4}$ is then a special case of the inequality of Theorem 64.

Corollary 65 *Let C be the cycle visiting in sequence the nodes $i_1, i_2, \dots, i_{4k+2}$ for some integer $k \geq 2$ satisfying $4k + 2 \leq n - 1$. Further, let $S_1 := \{i_j \in V(C) : j \text{ is odd}\}$, and $C_1 := \{(i_1, i_3), (i_3, i_5), \dots, (i_{4k+1}, i_1)\}$. Then the curtain inequality*

$$\alpha x := x(C) + x(S_1, S_1) + x(C_1) + \sum_{\substack{j=3 \\ j \text{ odd}}}^{2k-1} (x_{i_j i_{4k-j+2}} + x_{i_{4k-j+2} i_j}) + x_{i_1 i_{4k+1}} \\ \leq 4k + 1 =: \alpha_0$$

is a maximally 2-lifted, hence facet defining, cycle inequality for \tilde{P} .

Acknowledgements

The work of the first author was supported by the National Science Foundation under grant DMI-9802773 and by the Office of Naval Research under contract N00014-98-1-0196. The work of the second author was supported by M.I.U.R. and by C.N.R., Italy.

Chapter 4

EXACT METHODS FOR THE ASYMMETRIC TRAVELING SALESMAN PROBLEM

Matteo Fischetti

D.E.I., University of Padova

Via Gradenigo 6/A, 35100 Padova, Italy

fisch@dei.unipd.it

Andrea Lodi

D.E.I.S., University of Bologna

Viale Risorgimento 2, 40136 Bologna, Italy

alodi@deis.unibo.it

Paolo Toth

D.E.I.S., University of Bologna

Viale Risorgimento 2, 40136 Bologna, Italy

ptoth@deis.unibo.it

1. Introduction

In the present chapter we concentrate on the exact solution methods for the Asymmetric *TSP* proposed in the literature after the writing of the survey of Balas and Toth [81]. In Section 2 two specific branch-and-bound methods, based on the solution of the assignment problem as a relaxation, are presented and compared. In Section 3 a branch-and-bound method based on the computation of an additive bound is described, while in Section 4 a branch-and-cut approach is discussed. Finally, in Section 5 all these methods are computationally tested on a large set of instances, and compared with an effective branch-and-cut code for the symmetric *TSP*.

A formal definition of the problem is as follows. Let $G = (V, A)$ be a given complete digraph, where $V = \{1, \dots, n\}$ is the vertex set and

$A = \{(i, j) : i, j \in V\}$ the arc set, and let c_{ij} be the cost associated with arc $(i, j) \in A$ (with $c_{ii} = +\infty$ for each $i \in V$). A *Hamiltonian directed cycle (tour)* of G is a directed cycle visiting each vertex of V exactly once, i.e., a spanning subdigraph $\tilde{G} = (V, \tilde{A})$ of G such that $|\tilde{A}| = n$, and \tilde{G} is strongly connected, i.e., for each pair of distinct vertices $i, j \in V$, $i < j$, both paths from i to j and from j to i exist in \tilde{G} .

The *Asymmetric Traveling Salesman Problem (ATSP)* is to find a Hamiltonian directed cycle $G^* = (V, A^*)$ of G whose cost $\sum_{(i,j) \in A^*} c_{ij}$ is a minimum. Without loss of generality, we assume $c_{ij} \geq 0$ for any arc $(i, j) \in A$.

The following Integer Linear Programming formulation of *ATSP* is well-known:

$$v(ATSP) = \min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{i \in V} x_{ij} = 1 \quad j \in V \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1 \quad i \in V \tag{3}$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \quad S \subset V : S \neq \emptyset \tag{4}$$

$$x_{ij} \geq 0 \quad i, j \in V \tag{5}$$

$$x_{ij} \text{ integer} \quad i, j \in V \tag{6}$$

where $x_{ij} = 1$ if and only if arc (i, j) is in the optimal tour. Constraints (2) and (3) impose the in-degree and out-degree of each vertex be equal to one, respectively, while constraints (4) impose strong connectivity. Because of (2) and (3), conditions (4) can be equivalently re-written as the *Subtour Elimination Constraints (SECs)*:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad S \subset V : S \neq \emptyset \tag{7}$$

Moreover, it is well known that one can halve the number of constraints (4) by replacing them with

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \quad S \subset V : r \in S \tag{8}$$

or with

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \quad S \subset V : S \neq \emptyset, r \notin S \tag{9}$$

where r is any fixed vertex.

Several substructures of *ATSP* can be pointed out, each associated with a subset of constraints defining a well-structured relaxation whose solution value gives a valid lower bound for *ATSP*.

Constraints (2), (3) and (5), with objective function (1), define the well-known *min-sum Assignment Problem (AP)*. Such a problem always has an integer optimal solution, and requires finding a minimum-cost collection of vertex-disjoint *subtours* visiting all the vertices of G . If an optimal solution of *AP* determines only one directed cycle, then it satisfies all constraints (4) and hence is optimal for *ATSP* as well. Otherwise, each vertex subset S whose vertices are visited by the same subtour, determines a violated constraint (4). Relaxation *AP* can be solved in $O(n^3)$ time (see, e.g., Lawler [547]).

Constraints (2), (8) and (5), with objective function (1), define the well-known shortest *Spanning r -Arborescence Problem (r -SAP)*. Such a problem always has an integer optimal solution, and corresponds to finding a minimum-cost spanning subdigraph $\bar{G} = (V, \bar{A})$ of G such that (i) the in-degree of each vertex is exactly one, and (ii) each vertex can be reached from the *root* vertex r . If an optimal solution of *r -SAP* leaves each vertex with out-degree equal to one, then it satisfies all constraints (3) and hence is optimal for *ATSP* as well. Otherwise, each vertex having out-degree different from one, determines a violated constraint (3). Relaxation *r -SAP* can be solved in $O(n^2)$ time by finding the shortest spanning arborescence rooted at vertex r , and by adding to it a minimum-cost arc entering vertex r . Efficient algorithms for the shortest arborescence problem have been proposed by Edmonds [267], Fulkerson [338], Tarjan [788], and Camerini, Fratta and Maffioli [154, 155]; an efficient implementation of Tarjan's algorithm can be found in Fischetti and Toth [305]. Fischetti [294] described a modified $O(n^2)$ -time method to compute an improved lower bound not depending on the root vertex r .

A third substructure, corresponding to constraints (3), (9) and (5), with objective function (1), defines the shortest *Spanning r -Antiarborescence Problem (r -SAAP)*. Such a problem can easily be transformed into *r -SAP* by simply transposing the input cost matrix, hence it can be solved in $O(n^2)$ time.

In order to obtain tighter lower bounds, two enhanced relaxations, *r -SADP* and *r -SAADP*, can be introduced.

Relaxation *r -SADP* is obtained from *r -SAP* by adding constraint (3) for $i = r$, i.e., $\sum_{j \in V} x_{rj} = 1$, which imposes out-degree equal to one for the root vertex r . Such a problem can be transformed into *r -SAP* (and hence solved in $O(n^2)$ time) by considering a modified cost matrix

obtained by adding a large positive value M to costs c_{rj} for all $j \in V$, the optimal value of r -SADP being $v(r\text{-SAP}) - M$.

Relaxation r -SAADP is obtained in a similar way from r -SAAP, by adding constraint (2) for $j = r$, i.e., $\sum_{i \in V} x_{ir} = 1$, which imposes in-degree equal to one for the root vertex r . Such a problem can be solved in $O(n^2)$ time by transforming it into r -SADP through transposition of the input cost matrix.

2. AP-Based Branch-and-Bound Methods

In this section we review the AP-based branch-and-bound algorithms that have been proposed since the writing of the Balas and Toth [81] survey. All these algorithms are derived from the lowest-first branch and bound procedure TSP1 presented in Carpaneto and Toth [166] which is outlined below.

At each node h of the decision tree, procedure TSP1 solves a *Modified Assignment Problem* (MAP_h) defined by (1), (2), (3), (5) and the additional variable-fixing constraints associated with the following arc subsets:

$$\begin{aligned} E_h &= \{(i, j) \in A : x_{ij} \text{ is fixed to } 0\} \quad (\text{excluded arcs}) \\ I_h &= \{(i, j) \in A : x_{ij} \text{ is fixed to } 1\} \quad (\text{included arcs}) \end{aligned}$$

MAP_h can easily be transformed into a standard AP by properly modifying the cost matrix so as to take care of the additional constraints.

If the optimal solution to MAP_h does not define a Hamiltonian directed cycle and its value LB_h (yielding the lower bound associated with node h) is smaller than the current optimal solution value, say UB , then m descending nodes are generated from node h according to the following branching scheme (which is a modification of the subtour elimination rule proposed by Bellmore and Malone [97]).

Let $G_p = (V_p, A_p)$ be a subtour in the optimal MAP_h solution having the minimum number of not included arcs, i.e., such that $m := |A_p \setminus I_h|$ is a minimum, and let $(s_1, t_1), \dots, (s_m, t_m)$ be the non-included arcs of A_p taken in the same order as they appear along the subtour. The subsets of the excluded/included arcs associated with the j -th descending node of the current branching node h , say $g(j)$, are defined as follows ($j = 1, \dots, m$) (see also Figure 4.1 for an illustration):

$$\begin{aligned} E_{g(j)} &= E_h \cup \{(s_j, t_j)\} \\ I_{g(j)} &= I_h \cup \{(s_i, t_i) : i = 1, \dots, j - 1\} \end{aligned}$$

Moreover, each subset $E_{g(j)}$, $j > 1$, is enlarged by means of the arc (t_{j-1}, s_1) , so as to avoid subtours with just one non-included arc.

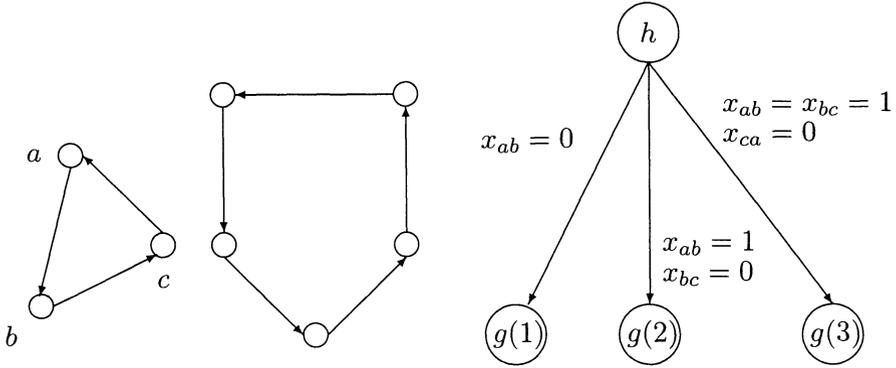


Figure 4.1. Branching on subtour $G_p = (V_p, A_p)$, where $V_p = \{a, b, c\}$ and $A_p = \{(a, b), (b, c), (c, a)\}$.

2.1. The Algorithm of Carpaneto, Dell’Amico and Toth

The approach of Carpaneto, Dell’Amico and Toth [164] differs from that presented in [166] in the following main respects:

- a) at the root node of the branch-decision tree, application of a *reduction procedure* to remove from G some arcs that cannot belong to an optimal tour; in this way the original digraph G can be transformed into a sparse one, say $\tilde{G} = (V, \tilde{A})$, allowing the use of procedures specialized for sparse graphs;
- b) use of an efficient *parametric technique* for the solution of the MAP ’s, allowing each MAP_h to be solved in $O(|\tilde{A}| \log n)$ time;
- c) application, at each branching node h , of a *subtour merging* procedure to decrease the number of subtours defined by the optimal MAP_h solution.

2.1.1 Reduction Procedure.

At the root node of the branch-decision tree, the AP corresponding to the original complete cost matrix, c , is solved through the $O(n^3)$ primal-dual procedure CTCS presented in Carpaneto and Toth [168]. Let (u_i) and (v_j) be an optimal solution of the dual problem associated with AP , and let LB_0 be the corresponding solution value. It is well known that, for each arc $(i, j) \in A$, the *reduced cost* $\bar{c}_{ij} = c_{ij} - u_i - v_j \geq 0$ represents a lower bound on the increase of the optimal AP solution value corresponding to the inclusion of arc (i, j) . If an $ATSP$ feasible solution of value UB is known, then each arc $(i, j) \in A$ such that $\bar{c}_{ij} \geq UB - LB_0$ can be removed from A , since its

inclusion in any *ATSP* solution cannot lead to a solution value smaller than UB . The original complete digraph G can thus be reduced into a sparse one, $\tilde{G} = (V, \tilde{A})$, where $\tilde{A} = \{(i, j) \in A : \bar{c}_{ij} < UB - LB_0\}$.

Value UB can be obtained through any heuristic procedure for *ATSP*; in [164] the patching algorithm proposed by Karp [498] is used. An alternative way is to compute an “artificial” upper bound by simply setting $UB = \alpha LB_0$, where $\alpha > 1$ is a given parameter. However, if at the end of the branch and bound algorithm no feasible solution of value less than UB is found, then αLB_0 is not a valid upper bound, so α must be increased and a new run needs to be performed.

2.1.2 Parametric *MAP* Solution. The effectiveness of the overall *ATSP* algorithm greatly depends on the efficiency of the *MAP* algorithm used. At each node h of the decision tree, instead of solving MAP_h from scratch, a parametric technique is used which finds only one shortest augmenting path. Indeed, when generating a descending node h from its father node k , only one arc, say (s, t) , is excluded from the solution of MAP_k . So, to obtain the optimal solution of MAP_h from that of MAP_k it is only necessary to satisfy constraint (2) for $j = t$ and constraint (3) for $i = s$, i.e., one only needs to find a single shortest augmenting path from vertex s to vertex t in the bipartite graph corresponding to MAP_h with respect to the current reduced cost matrix \bar{c} . Note that the addition of the new included arcs contained in $I_h \setminus I_k$ does not affect the parametrization, as these arcs already belong to the optimal solution of MAP_k . As graph \tilde{G} is sparse, the shortest augmenting path is found through a procedure derived from the labelling algorithm proposed by Johnson [459] for the computation of shortest paths in sparse graphs, which uses a heap queue. Hence, the resulting time complexity for solving each MAP_h is $O(|\tilde{A}| \log n)$.

The computation of the shortest augmenting path at each node h is stopped as soon as its current reduced cost becomes greater or equal to the gap between the current upper bound value UB and the optimal value of MAP_k .

2.1.3 Subtour Merging. Consider a node h of the decision tree for which several optimal *MAP* solutions exist. Computational experience shows that the optimal solution which generally leads to the smallest number of nodes in the subtree descending from h is that having the minimum number of subtours. A heuristic procedure which tries to

decrease the number of subtours is obtained by iteratively applying the following rule.

Given two subtours $G_a = (V_a, A_a)$ and $G_b = (V_b, A_b)$, if there exists an arc pair $(i_a, j_a) \in A_a$ and $(i_b, j_b) \in A_b$ such that arcs (i_a, j_b) and (i_b, j_a) have zero reduced costs (i.e., $\bar{c}_{i_a j_b} = \bar{c}_{i_b j_a} = 0$), then an equivalent optimal solution to MAP_h can be obtained by connecting subtours G_a and G_b to form a single subtour $G_a = (V_a \cup V_b, A_a \cup A_b \setminus \{(i_a, j_a), (i_b, j_b)\} \cup \{(i_a, j_b), (i_b, j_a)\})$. If, at the end of the procedure, a Hamiltonian directed cycle is found, then an optimal solution to the ATSP associated with node h has been found and no descending nodes need to be generated.

The above subtour-merging procedure is always applied at the root node of the decision tree. As to the other nodes, it is applied only if the total number of zero reduced cost arcs at the root node is greater than a given threshold β (e.g., $\beta = 2.5n$). Indeed, the procedure is typically effective only if the subdigraph corresponding to the zero reduced-cost arcs contains a sufficiently large number of arcs. (Computational experiments have shown that an adaptive strategy, which counts the number of zero reduced-cost arcs at each node and then decides on the opportunity to apply the procedure, often gives worse results than the threshold method above.)

2.2. The Algorithms of Miller and Pekny

Effective procedures for the solution of the ATSP have been proposed by Miller and Pekny in the early nineties [596, 597, 665, 664]. These methods are also based on the general approach presented in Carpaneto and Toth [166], the main differences and similarities between them being discussed below.

In [596], Miller and Pekny presented a preliminary algorithm which is a parallelization of the approach of Carpaneto and Toth, improved with the application of the patching heuristic [498] at the root node.

The algorithm presented in [664] represents a substantial improvement of the original parallel procedure. The MAP 's at the nodes are solved through an $O(n^2)$ parametric procedure which computes a single augmenting path using a d -heap. Moreover, the patching algorithm is applied at the root node, and to the other nodes with decreasing frequency as search progresses. In addition, the branch-and-bound phase is preceded by a sparsification of the cost matrix obtained by removing all the entries with cost greater than a given threshold λ . A sufficient

condition is given to check whether the optimal solution obtained with respect to the sparse matrix is optimal for the original matrix as well.

The algorithm presented in [665] is a modification of that presented in [664], obtained with the application, at each node, of an exact procedure to find a Hamiltonian directed cycle on the subdigraph defined by the arcs with zero reduced cost. The most sophisticated version of the Miller and Pekny codes appears to be that presented in [597], which includes all the improvements previously proposed by the authors.

The similarities among the approach of Carpaneto, Dell'Amico and Toth [164] and the algorithms of Miller and Pekny are the following: (a) the branching rule is that proposed in [166]; (b) the *MAP*'s at the various branching nodes are solved through an $O(n^2)$ procedure; and (c) the patching algorithm is applied at the root node. The two approaches differ in the following aspects: (a) for the sparsification phase, [164] proposes a criterion based on the comparison between the reduced costs given by the initial *AP* and the gap between lower and upper bound; (b) an efficient technique to store and retrieve the subproblems is proposed in [164] so that the exploration of the branch-decision tree is accelerated; and (c) a fast heuristic algorithm to find a Hamiltonian directed cycle on the subdigraph defined by the arcs with zero reduced cost is applied in [164].

Comparing the computational results obtained by Miller and Pekny with those presented in [164], it appears that the latter code is slower than the algorithm presented in [597] for small cost ranges (and random instances), but it seems to be faster for large cost ranges. On the whole, the two methods exhibit a comparable performance.

3. An Additive Branch-and-Bound Method

This section describes the solution approach proposed by Fischetti and Toth [304], who embedded a more sophisticated bounding procedure within the standard branch-and-bound method of Carpaneto and Toth [166]. Observe that *AP*, *r-SADP* and *r-SAADP* relaxations (as defined in Section 1) are complementary to each other. Indeed, *AP* imposes the degree constraints for all vertices, while connectivity constraints are completely neglected. Relaxation *r-SADP*, instead, imposes reachability from vertex *r* to all the other vertices, while out-degree constraints are neglected for all vertices different from *r*. Finally, *r-SAADP* imposes reachability from all the vertices to vertex *r*, while in-degree constraints are neglected for all vertices different from *r*. A possible way of combining the three relaxations is to apply the so-called *additive approach* introduced by Fischetti and Toth [303].

3.1. An Additive Bounding Procedure

An additive bounding procedure for *ATSP* can be outlined as follows. Let $\mathcal{L}^{(1)}, \mathcal{L}^{(2)}, \dots, \mathcal{L}^{(q)}$ be q bounding procedures available for *ATSP*. Suppose that, for $h = 1, 2, \dots, q$ and for any cost matrix \bar{c} , procedure $\mathcal{L}^{(h)}(\bar{c})$, when applied to the *ATSP* instance having cost matrix \bar{c} , returns its lower bound $\delta^{(h)}$ as well as a so-called *residual cost* matrix $c^{(h)}$ such that:

- i) $c_{ij}^{(h)} \geq 0$ for each $i, j \in V$;
- ii) $\delta^{(h)} + \sum_{(i,j) \in A} c_{ij}^{(h)} x_{ij} \leq \sum_{(i,j) \in A} \bar{c}_{ij} x_{ij}$ for each feasible *ATSP* solution x .

The additive approach generates a sequence of *ATSP* instances, each obtained by considering the residual cost matrix corresponding to the previous instance and by applying a different bounding procedure. A Pascal-like outline of the approach follows.

ALGORITHM ADDITIVE:

1. **input:** cost matrix c ;
2. **output:** lower bound δ and the residual-cost matrix $c^{(q)}$;
begin
3. initialize $c^{(0)} := c$, $\delta := 0$;
4. **for** $h := 1$ **to** q **do**
 begin
5. apply $\mathcal{L}^{(h)}(c^{(h-1)})$, thus obtaining value $\delta^{(h)}$
 and the residual cost matrix $c^{(h)}$;
6. $\delta := \delta + \delta^{(h)}$
- end**
- end.**

An inductive argument shows that the values δ computed at step 6 give a non decreasing sequence of valid lower bounds for *ATSP*. Moreover, the final residual-cost matrix $c^{(q)}$ can be used for reduction purposes.

Related approaches, using reduced costs for improving lower bounds for *ATSP*, are those of Christofides [188] and Balas and Christofides [70]. For a comparison of the additive approach with the restricted Lagrangian approach of Balas and Christofides, the reader is referred to [304].

Note that, because of condition ii) above, each bounding procedure $\mathcal{L}^{(h)}$ of the sequence introduces an *incremental gap*

$$\gamma^{(h)} = v^{(h-1)}(ATSP) - (v^{(h)}(ATSP) + \delta^{(h)}) \geq 0,$$

where $v^{(k)}(ATSP)$ denotes the optimal solution value of the *ATSP* instance associated with cost matrix $c^{(k)}$. It follows that the overall gap γ between $v(ATSP)$ and the final lower bound $\delta = \sum_{h=1}^q \delta^{(h)}$ cannot be less than $\sum_{h=1}^q \gamma^{(h)}$.

Procedures $\mathcal{L}^{(1)}, \mathcal{L}^{(2)}, \dots, \mathcal{L}^{(q)}$ can clearly be applied in a different sequence, thus producing different lower bound values and residual costs. As a heuristic rule, it is worthwhile to apply procedures $\mathcal{L}^{(h)}$ according to increasing (estimated) percentage incremental gaps, so as to avoid the introduction of high percentage incremental gaps at the beginning of the sequence, when the current value $v^{(h)}(ATSP)$ is still large.

A key step of the above algorithm is the computation of the residual costs. Since all the bounding procedures considered in [304] are based on linear programming relaxations, valid residual cost matrices can be obtained by computing the *reduced cost* matrices associated with the corresponding *LP*-dual optimal solutions.

Reduced costs associated with the *AP* relaxation can easily be obtained without extra computational effort. As to the reduced costs for *r-SAP*, those associated with the arcs not entering the root vertex r are the reduced costs of the shortest spanning arborescence problem (which can be computed in $O(n^2)$ time through a procedure given in [305]), while those associated with the arcs entering r are obtained by subtracting their minimum from the input costs. Reduced costs for problems *r-SAAP*, *r-SADP* and *r-SAADP* can be obtained in a similar way.

Here is an overall additive bounding algorithm, subdivided into four stages.

ALGORITHM ADD-ATSP:

1. **input:** cost matrix c ;
2. **output:** lower bound δ and the residual-cost matrix \bar{c} ;
begin
- Stage 1:*
3. solve problem *AP* on the original cost matrix c and let \bar{c} be the corresponding reduced cost matrix;
 $\delta := v(AP)$;
- Stage 2:*
4. solve problem *1-SAP* on cost matrix \bar{c} and update \bar{c} to become the corresponding reduced cost matrix;
 $\delta := \delta + v(1-SAP)$;
5. solve problem *1-SAAP* on cost matrix \bar{c} and update \bar{c} to become the corresponding reduced cost matrix;
 $\delta := \delta + v(1-SAAP)$;

Stage 3:

```

6.   for  $r := 1$  to  $n$  do
      begin
7.       solve problem  $r$ -SADP on cost matrix  $\bar{c}$  and
          update  $\bar{c}$  to become the corresponding reduced
          cost matrix;
           $\delta := \delta + v(r\text{-SADP})$ 
      end;

```

Stage 4:

```

8.   for  $r := 1$  to  $n$  do
      begin
9.       solve problem  $r$ -SAADP on cost matrix  $\bar{c}$  and
          update  $\bar{c}$  to become the corresponding reduced
          cost matrix;
           $\delta := \delta + v(r\text{-SAADP})$ 
      end
end.

```

Let $G_0 = (V, A_0)$ denote the spanning subdigraph of G defined by the arcs whose current reduced cost is zero.

At Stage 1, the bounding procedure based on the AP relaxation is applied. After this stage, each vertex in G_0 has at least one entering and leaving arc; however, G_0 is not guaranteed to be strongly connected.

At Stage 2, one forces the strong connectivity of G_0 by applying the bounding procedures based on 1-SAP and 1-SAAP. Indeed, after step 4 each vertex can be reached from vertex 1 in G_0 , whereas after step 5 each vertex can reach vertex 1.

The current spanning subdigraph G_0 may at this point contain a tour, in which case lower bound δ cannot be further increased through an additive approach. If such a tour has been detected, it corresponds to a heuristic solution to $ATSP$, whose optimality can be checked by comparing its original cost with lower bound δ . More often, however, spanning subdigraph G_0 is non-Hamiltonian.

Let the forward and backward star of a node h in a given digraph $G' = (V', A')$ be defined as $\{j \in V' : (h, j) \in A'\}$ and $\{i \in V' : (i, h) \in A'\}$, respectively. We say that a vertex $r \in V$ is a *forward articulation point* of G_0 if none of the vertices of its forward star can reach all other vertices in $V \setminus \{r\}$ without passing through vertex r . Analogously, a vertex $r \in V$ is said to be a *backward articulation point* of G_0 if none of the vertices of its backward star can be reached from all the other vertices in $V \setminus \{r\}$ without passing through vertex r . Clearly, the existence of

a forward or backward articulation point is a sufficient condition for G_0 to be non-Hamiltonian.

A related concept is that of (undirected) articulation point: a vertex r is an articulation point of G_0 if the underlying undirected subdigraph of G_0 induced by vertex subset $V \setminus \{r\}$ has more than one connected component. Notice that concept of forward (or backward) articulation is stronger than that of (undirected) articulation. Indeed, if vertex r is an (undirected) articulation point of G_0 , then it is also a forward and a backward articulation point of G_0 , while the opposite does not always hold.

The existence of a forward (resp. backward) articulation point r of G_0 can be exploited to increase the current lower bound δ by solving relaxation r -SADP (resp. r -SAADP). Indeed, in this case no zero cost r -arborescence (resp. r -antiarborescence) in which r has out-degree (resp. in-degree) equal to one, exists with respect to the current reduced costs \bar{c}_{ij} . Accordingly, for each vertex r one applies bounding procedures based on relaxations r -SADP (at Stage 3) and r -SAADP (at Stage 4), so as to increase the current lower bound δ in case the root vertex r is a forward or backward articulation point, respectively. After each execution of steps 7 and 9, the current vertex r is guaranteed not to be a forward or backward articulation point of the current graph G_0 , respectively.

The overall time complexity of algorithm ADD-ATSP is $O(n^3)$, the most time-consuming steps being step 3 ($O(n^3)$), and steps 7 and 9 ($O(n^2)$), which are executed n times.

The tightness of the final lower bound δ greatly depends on the reduced costs obtained after each lower bound computation. In particular, consider the LP-dual of AP, defined by:

$$v(D-AP) = \max \sum_{i \in V} (u_i + v_i)$$

subject to

$$c_{ij} - u_i - v_j \geq 0 \quad i, j \in V$$

and let (\bar{u}, \bar{v}) be the dual optimal solution found at step 3. For each vertex $h \in V$, let \bar{L}_h be the cost of the shortest path from vertex 1 to vertex h , computed with respect to the current reduced costs $\bar{c}_{ij} = c_{ij} - \bar{u}_i - \bar{v}_j$. It is known (see, e.g., [505]) that an alternative dual optimal solution (u^*, v^*) is given by $u_i^* = \bar{u}_i - \bar{L}_i$ and $v_j^* = \bar{v}_j + \bar{L}_j$ for each $i \in V$. (Indeed, one has $\sum_{i \in V} (u_i^* + v_j^*) = \sum_{i \in V} (\bar{u}_i + \bar{v}_i)$ while, for each $j \in V$, $c_{ij} - u_i^* - v_j^* = \bar{c}_{ij} + \bar{L}_i - \bar{L}_j \geq 0$ follows from the definition of \bar{L}_h 's as costs of shortest paths.) Now, let $c_{ij}^* = c_{ij} - u_i^* - v_j^*$ be the reduced costs associated with this alternative dual optimal solution. One can easily

verify that the cost P_{hk}^* of any simple path from vertex h to vertex k (computed with respect to c^*), is equal to $\bar{P}_{hk} + \bar{L}_h - \bar{L}_k$, where \bar{P}_{hk} is the cost of the same path computed with respect to \bar{c} . Therefore, c^* can be viewed as a biased reduced-cost matrix obtained from \bar{c} by reducing the cost of the paths emanating from vertex 1, while increasing the cost of the paths towards vertex 1.

A new additive bounding algorithm, B-ADD-ATSP (B for biased), can now be obtained from ADD-ATSP by adding the following step right after step 3:

- 3'. compute (on the current reduced cost matrix \bar{c}) the cost \bar{L}_h of the shortest path from vertex 1 to all vertices $h \in V$;
 for each $i, j \in V$ do $\bar{c}_{ij} := \bar{c}_{ij} + \bar{L}_i - \bar{L}_j$

Note that, after step 3', spanning subdigraph G_0 contains a 1-arborescence, hence step 4 can be omitted in B-ADD-ATSP.

At first glance, algorithm B-ADD-ATSP appears to be weaker than ADD-ATSP, since a step producing a possible increase on the current lower bound (step 4) has been replaced by a step which gives no improvement (step 3'). However, the cost biasing introduced at step 3' may allow the subsequent step 5 to increase its contribution to the current lower bound. Computational experience has shown that B-ADD-ATSP typically outperforms ADD-ATSP, hence algorithm B-ADD-ATSP is chosen in [304].

As to the experimental computing time of algorithm B-ADD-ATSP, it can greatly be reduced by the implementation given in [304].

4. A Branch-and-Cut Approach

We next outline the polyhedral method of Fischetti and Toth [306]. Branch-and-cut methods for *ATSP* with side constraints have been proposed recently by Ascheuer [43], Ascheuer, Jünger and Reinelt [46], and Ascheuer, Fischetti and Grötschel [44, 45], among others. The Fischetti-Toth method is based on model (1)–(6), and exploits additional classes of facet-inducing inequalities for the *ATSP* polytope P that proved to be of crucial importance for the solution of some real-world instances. For each class, we will address the associated *separation* problem (in its optimization version), defined as follows: Given a point $x^* \geq 0$ satisfying the degree equations, and a family \mathcal{F} of *ATSP* inequalities, find a most violated member of \mathcal{F} , i.e., an inequality $\alpha x \leq \alpha_0$ belonging to \mathcal{F} and maximizing the degree of violation $\alpha x^* - \alpha_0$. The reader is referred to Chapter 3 of the present book for a polyhedral analysis of the *ATSP*

polytope, and to Chapter 2 for the design of branch-and-cut methods for the symmetric *TSP*.

To simplify notation, for any $f : A \rightarrow \mathbb{R}$ and $S_1, S_2 \subseteq V$, we write $f(S_1, S_2)$ for $\sum_{i \in S_1} \sum_{j \in S_2} f_{ij}$; moreover, we write $f(i, S_2)$ or $f(S_1, i)$ whenever $S_1 = \{i\}$ or $S_2 = \{i\}$, respectively.

4.1. Separation of Symmetric Inequalities

An *ATSP* inequality $\alpha x \leq \alpha_0$ is called *symmetric* when $\alpha_{ij} = \alpha_{ji}$ for all $(i, j) \in A$. Symmetric inequalities can be thought of as derived from valid inequalities for the *Symmetric Traveling Salesman Problem (STSP)*, defined as the problem of finding a minimum-cost Hamiltonian cycle in a given undirected graph $G_E = (V, E)$. Indeed, let $y_e = 1$ if edge $e \in E$ belongs to the optimal *STSP* solution; $y_e = 0$ otherwise. Every inequality $\sum_{e \in E} \alpha_e y_e \leq \alpha_0$ for *STSP* can be transformed into a valid *ATSP* inequality by simply replacing y_e by $x_{ij} + x_{ji}$ for all edges $e = (i, j) \in E$. This produces the symmetric inequality $\alpha x \leq \alpha_0$, where $\alpha_{ij} = \alpha_{ji} = \alpha_{(i,j)}$ for all $i, j \in V$, $i \neq j$. Conversely, every symmetric *ATSP* inequality $\alpha x \leq \alpha_0$ corresponds to the valid *STSP* inequality $\sum_{(i,j) \in E} \alpha_{ij} y_{(i,j)} \leq \alpha_0$.

The above correspondence implies that every separation algorithm for *STSP* can be used, as a “black box”, for *ATSP* as well. To this end, given the *ATSP* (fractional) point x^* one first defines the undirected counterpart y^* of x^* by means of the transformation

$$y_e^* := x_{ij}^* + x_{ji}^* \quad \text{for all edges } e = (i, j) \in E$$

and then applies the *STSP* separation algorithm to y^* . On return, the detected most violated *STSP* inequality is transformed into its *ATSP* counterpart, both inequalities having the same degree of violation.

Several exact/heuristic separation algorithms for *STSP* have been proposed in recent years, all of which can be used for *ATSP*; see Chapter 2 of the present book for further details. In [306] only two such separation tools are used, namely:

- i) the Padberg-Rinaldi [646] exact algorithm for SECs; and
- ii) the simplest heuristic scheme for comb (actually, 2-matching) constraints in which the components of the graph induced by the edges $e \in E$ with fractional y_e^* are considered as potential handles of the comb.

4.2. Separation of D_k^+ and D_k^- Inequalities

The following D_k^+ inequalities have been proposed by Grötschel and Padberg [405]:

$$x_{i_1 i_k} + \sum_{h=2}^k x_{i_h i_{h-1}} + 2 \sum_{h=2}^{k-1} x_{i_1 i_h} + \sum_{h=3}^{k-1} x(\{i_2, \dots, i_{h-1}\}, i_h) \leq k - 1 \quad (10)$$

where (i_1, \dots, i_k) is any sequence of $k \in \{3, \dots, n - 1\}$ distinct vertices. D_k^+ inequalities are facet-inducing for the ATSP polytope [295], and are obtained by lifting the cycle inequality $\sum_{(i,j) \in C} x_{ij} \leq k - 1$ associated with the subtour $C := \{(i_1, i_k), (i_k, i_{k-1}), \dots, (i_2, i_1)\}$. Notice that the vertex indices along C are different from those used in the original Grötschel-Padberg definition [405], so as to allow for a simplified description of the forthcoming separation procedure.

As a slight extension of the original definition, we allow for $k = 1, 2$ in the sequel, in which cases (10) degenerates into the valid constraints $x_{i_1 i_1} \leq 0$ and $x_{i_1 i_2} + x_{i_2 i_1} \leq 1$, respectively.

The separation problem for the class of D_k^+ inequalities calls for a vertex sequence (i_1, \dots, i_k) , $1 \leq k \leq n - 1$, for which the degree of violation

$$\begin{aligned} \phi(i_1, \dots, i_k) := & x_{i_1 i_k}^* + \sum_{h=2}^k x_{i_h i_{h-1}}^* + 2 \sum_{h=2}^{k-1} x_{i_1 i_h}^* \\ & + \sum_{h=3}^{k-1} x^*(\{i_2, \dots, i_{h-1}\}, i_h) - k + 1 \end{aligned} \quad (11)$$

is as large as possible. This is itself a combinatorial optimization problem that can be solved by the following simple implicit enumeration scheme.

We start with an empty node sequence. Then, iteratively, we extend the current sequence in any possible way and evaluate the degree of violation of the corresponding D_k^+ inequality. The process can be visualized by means of a branch-decision tree. The root node (level 0) of the tree represents the empty sequence. Each node at level k ($1 \leq k \leq n - 1$) corresponds to a sequence of the type (i_1, \dots, i_k) ; when $k \leq n - 2$, each such node generates $n - k$ descending nodes, one for each possible extended sequence $(i_1, \dots, i_k, i_{k+1})$. Exhaustive enumeration of all nodes of the tree is clearly impractical, even for small values of n . On the other hand, a very large number of these nodes can be pruned (along with the associated subtrees) by means of the following simple upper

bound computation. Let (i_1, \dots, i_k) be the sequence associated with the current branching node, say μ , and let ϕ_{max} denote the maximum degree of violation so far found during the enumeration. Consider any potential descendent node of μ , associated with a sequence of the type $(i_1, \dots, i_k, i_{k+1}, \dots, i_m)$. Then, directly from definition (11) one has

$$\begin{aligned} \phi(i_1, \dots, i_k, i_{k+1}, \dots, i_m) &\leq \pi(x^*; i_1, \dots, i_k) + x_{i_{k+1}i_k}^* + [x^*(i_1, V) - 1] \\ &\quad + \sum_{h=k+1}^{m-1} [x^*(V, i_h) - 1] = \pi(i_1, \dots, i_k) + x_{i_{k+1}i_k}^* \end{aligned} \tag{12}$$

where we have defined

$$\pi(i_1, \dots, i_k) := \sum_{h=2}^k x_{i_h i_{h-1}}^* + \sum_{h=2}^k x^*({i_1, \dots, i_{h-1}}, i_h) - k + 1$$

Observe that $\pi(i_1, \dots, i_k)$ cannot exceed the degree of violation of the SEC associated with $S := \{i_1, \dots, i_k\}$; hence one has $\pi(i_1, \dots, i_k) \leq 0$ whenever all SECs are satisfied by x^* .

According to (12), the only descending nodes of μ that need to be generated are those associated with a sequence $(i_1, \dots, i_k, i_{k+1})$ such that

$$x_{i_k i_{k+1}}^* > \phi_{max} - \pi(i_1, \dots, i_k) \tag{13}$$

Notice that both quantities $\phi(i_1, \dots, i_k)$ and $\pi(i_1, \dots, i_k)$ can parametrically be computed along the branching tree as:

$$\phi(i_1, \dots, i_k) = \phi(i_1, \dots, i_{k-1}) + x_{i_1 i_k}^* + x_{i_k i_{k-1}}^* + x^*({i_1, \dots, i_{k-2}}, i_{k-1}) - 1$$

and

$$\pi(i_1, \dots, i_k) = \pi(i_1, \dots, i_{k-1}) + x_{i_k i_{k-1}}^* + x^*({i_1, \dots, i_{k-1}}, i_k) - 1$$

where $\phi(i_1) := \pi(i_1) := 0$ for all singleton sequences (i_1) .

Restriction (13) is very effective in practice, and dramatically reduces the number of nodes typically generated in the enumeration. Nevertheless, in some cases one may be interested in further reducing the computing time spent in the procedure. To this end, before running the above-described exact enumeration, one can try a “truncated” version of it in which each node at level $k \geq 2$ generates at most one descending node, namely the one associated with the sequence $(i_1, \dots, i_k, i_{k+1})$, if any, where $x_{i_{k+1}i_k}^* > 0$ and $x_{i_1 i_{k+1}}^* + x_{i_{k+1}i_k}^*$ is as large as possible.

The performance of the overall branch-and-cut algorithm is generally improved if one generates, at each round of separation, a number of violated cuts (rather than the most violated one) for each family. In [306],

the most violated D_k^+ inequality associated with a node sequence starting with i_1 is generated for each $i_1 \in V$. This is obtained by searching the decision tree in a depth-first manner, and resetting to zero the value ϕ_{max} of the incumbent best sequence whenever one backtracks to a node at level 1.

We conclude this section by addressing the following D_k^- inequalities:

$$x_{i_k i_1} + \sum_{h=2}^k x_{i_{h-1} i_h} + 2 \sum_{h=2}^{k-1} x_{i_h i_1} + \sum_{h=3}^{k-1} x(i_h, \{i_2, \dots, i_{h-1}\}) \leq k - 1 \quad (14)$$

where (i_1, \dots, i_k) is any sequence of $k \in \{3, \dots, n - 1\}$ distinct nodes. D_k^- inequalities are valid [405] and facet-inducing [295] for P ; they can be obtained by lifting the cycle inequality $\sum_{(i,j) \in C} x_{ij} \leq k - 1$ associated with the directed cycle $C := \{(i_1, i_2), \dots, (i_{k-1}, i_k), (i_k, i_1)\}$.

D_k^- inequalities can be thought of as derived from D_k^+ inequalities by swapping the coefficient of the two arcs (i, j) and (j, i) for all $i, j \in V, i < j$. This is a perfectly general operation, called *transposition* in [405], that works as follows.

For every $\alpha \in \mathbb{R}^A$, let $\alpha^T \in \mathbb{R}^A$ be defined by: $\alpha_{ij}^T := \alpha_{ji}$ for all $(i, j) \in A$. Clearly, inequality $\alpha x \leq \alpha_0$ is valid (or facet-inducing) for the ATSP polytope P if and only if its transposed version, $\alpha^T x \leq \alpha_0$, is. This follows from the obvious fact that $\alpha^T x = \alpha x^T$, where $x \in P$ if and only if $x^T \in P$. Moreover, every separation procedure for $\alpha x \leq \alpha_0$ can also be used, as a black box, to deal with $\alpha^T x \leq \alpha_0$. To this end one gives the transposed point $(x^*)^T$ (instead of x^*) on input to the procedure, and then transposes the returned inequality.

The above considerations show that both the heuristic and exact separation algorithms designed for D_k^+ inequalities can be used for D_k^- inequalities as well.

4.3. Separation of Odd CAT Inequalities

The following class of inequalities has been proposed by Balas [62]. Two distinct arcs (i, j) and (u, v) are called *incompatible* if $i = u$, or $j = v$, or $i = v$ and $j = u$; *compatible* otherwise. A *Closed Alternating Trail* (CAT, for short) is a sequence $T = \{a_1, \dots, a_t\}$ of t distinct arcs such that, for $k = 1, \dots, t$, arc a_k is incompatible with arcs a_{k-1} and a_{k+1} , and compatible with all other arcs in T (with $a_0 := a_t$ and $a_{t+1} := a_1$). Let $\delta^+(v)$ and $\delta^-(v)$ denote the set of the arcs of G leaving and entering any vertex $v \in V$, respectively. Given a CAT T , a node v is called a *source* if $|\delta^+(v) \cap T| = 2$, whereas it is called a *sink* if $|\delta^-(v) \cap T| = 2$. Notice that a node can play both source and sink roles. Let Q be the set of the arcs $(i, j) \in A \setminus T$ such that i is a source and j is a sink node.

For any CAT of odd length t , the following *odd CAT inequality*

$$\sum_{(i,j) \in T \cup Q} x_{ij} \leq \frac{|T| - 1}{2} \tag{15}$$

is valid and facet-defining (except in two pathological cases arising for $n \leq 6$) for the *ATSP* polytope [62].

We next describe a heuristic separation algorithm for the family of odd CAT inequalities. This algorithm is based on the known fact that odd CAT inequalities correspond to odd cycles on an auxiliary “incompatibility” graph [62]. Also, the separation algorithm can be viewed as a specialized version of a scheme proposed by Caprara and Fischetti [159] for the separation of a subclass of Chvátal-Gomory cuts for general integer programming problems.

Given the point x^* , we set-up an edge-weighted undirected graph $\tilde{G} = (\tilde{N}, \tilde{E})$ having a node ν_a for each arc $a \in A$ with $x_a^* > 0$, and an edge $e = (\nu_a, \nu_b)$ for each pair a, b of incompatible arcs, whose weight is defined as $w_e := 1 - (x_a^* + x_b^*)$. We assume that x^* satisfies all degree equations as well as all trivial SECs of the form $x_{ij} + x_{ji} \leq 1$; this implies $w_e \geq 0$ for all $e \in \tilde{E}$.

Let $\tilde{\delta}(v)$ contain the edges in \tilde{E} incident with a given node $v \in \tilde{N}$. A *cycle* \tilde{C} of \tilde{G} is an edge subset of \tilde{E} inducing a connected subdigraph of \tilde{G} , and such that $|\tilde{C} \cap \tilde{\delta}(v)|$ is even for all $v \in \tilde{N}$. Cycle \tilde{C} is called (i) *odd* if $|\tilde{C}|$ is odd; (ii) *simple* if $|\tilde{C} \cap \tilde{\delta}(v)| \in \{0, 2\}$ for all $v \in \tilde{N}$; and (iii) *chordless* if the subdigraph of \tilde{G} induced by the nodes covered by \tilde{C} has no other edges than those in \tilde{C} .

By construction, every simple and chordless odd cycle \tilde{C} in \tilde{G} corresponds to an odd CAT T , where $a \in T$ if and only if ν_a is covered by \tilde{C} . In addition, the total weight of \tilde{C} is

$$w(\tilde{C}) := \sum_{e \in \tilde{C}} w_e = \sum_{(\nu_a, \nu_b) \in \tilde{C}} (1 - x_a^* - x_b^*) = |T| - 2 \sum_{a \in T} x_a^*$$

hence $(1 - w(\tilde{C}))/2$ gives a lower bound on the degree of violation of the corresponding CAT inequality, computed as

$$\phi(T) := (2 \sum_{a \in T \cup Q} x_a^* - |T| + 1)/2$$

The heuristic separation algorithm used in [306] computes, for each $e \in \tilde{E}$, a minimum-weight odd cycle \tilde{C}_e that uses edge e . If \tilde{C}_e happens to be simple and chordless, then it corresponds to an odd CAT, say T . If, in addition, the lower bound $(1 - w(\tilde{C}_e))/2$ exceeds a given threshold $\theta =$

$-1/2$, then the corresponding inequality is hopefully violated; hence one evaluates its actual degree of violation, $\phi(T)$, and stores the inequality if $\phi(T) > 0$. In order to avoid detecting twice the same inequality, edge e is removed from \tilde{G} after the computation of each \tilde{C}_e .

In order to increase the chances of finding odd cycles that are simple and chordless, all edge weights can be made strictly positive by adding to them a small positive value $\epsilon := 0.001$. This guarantees that ties are broken in favor of inclusion-minimal sets \tilde{C}_e . Notice, however, that a generic minimum-weight odd cycle \tilde{C}_e does not need to be neither simple nor chordless even in this case, due to the fact that one imposes $e \in \tilde{C}_e$. For example, \tilde{C}_e may decompose into 2 simple cycles, say \tilde{C}_e^1 and \tilde{C}_e^2 , where \tilde{C}_e^1 is of even cardinality and goes through edge e , and \tilde{C}_e^2 is of odd cardinality and overlaps \tilde{C}_e^1 in a node.

The key point of the algorithm is the computation in \tilde{G} of a minimum-weight odd cycle going through a given edge. Assuming that the edge weights are all nonnegative, this problem is known to be polynomially solvable as it can be transformed into a shortest path problem; see Gerards and Schrijver [357]. To this end one constructs an auxiliary bipartite undirected graph $G_B = (N'_B \cup N''_B, E_B)$ obtained from \tilde{G} as follows. For each ν in \tilde{G} there are two nodes in G_B , say ν' and ν'' . For each edge $e = (\nu_1, \nu_2)$ of \tilde{G} there are two edges in G_b , namely edge (ν'_1, ν''_2) and edge (ν'_2, ν''_1) , both having weight w_e . By construction, every minimum-weight odd cycle \tilde{C}_e of \tilde{G} going through edge $e = (\nu_1, \nu_2)$ corresponds in G_B to a shortest path from ν'_1 to ν'_2 , plus the edge (ν'_2, ν''_1) . Hence, the computation of all \tilde{C}_e 's can be performed efficiently by computing, for each ν'_1 , the shortest path from ν'_1 to all other nodes in N'_B .

4.4. Clique Lifting and Shrinking

Clique lifting can be described as follows, see Balas and Fischetti [73] for details. Let $P(G')$ denote the ATSP polytope associated with a given complete digraph $G' = (V', A')$. Given a valid inequality $\beta y \leq \beta_0$ for $P(G')$, we define

$$\beta_{hh} := \max\{\beta_{ih} + \beta_{hj} - \beta_{ij} : i, j \in V' \setminus \{h\}, i \neq j\} \quad \text{for all } h \in V'$$

and construct an enlarged complete digraph $G = (V, A)$ obtained from G' by replacing each node $h \in V'$ by a clique S_h containing at least one node (hence, $|V| = \sum_{h \in V'} |S_h| \geq |V'|$). In other words $(S_1, \dots, S_{|V'|})$ is a proper partition of V , in which the h -th set corresponds to the h -th node in V' .

For all $v \in V$, let $v \in S_{h(v)}$. We define a new *clique lifted* inequality for $P(G)$, say $\alpha x \leq \alpha_0$, where $\alpha_0 := \beta_0 + \sum_{h \in V'} \beta_{hh}(|S_h| - 1)$ and

$\alpha_{ij} := \beta_{h(i)h(j)}$ for each $(i, j) \in A$. It is shown in [73] that the new inequality is always valid for $P(G)$; in addition, if the starting inequality $\beta x \leq \beta_0$ defines a facet of $P(G')$, then $\alpha x \leq \alpha_0$ is guaranteed to be facet-inducing for $P(G)$.

Clique lifting is a powerful theoretical tool for extending known classes of inequalities. Also, it has important applications in the design of separation algorithms in that it allows one to simplify the separation problem through the following *shrinking* procedure [646].

Let $S \subset V$, $2 \leq |S| \leq n - 2$, be a vertex subset saturated by x^* , in the sense that $x^*(S, S) = |S| - 1$, and suppose S is shrunk into a single node, say σ , and x^* is updated accordingly. Let $G' = (V', A')$ denote the shrunk digraph, where $V' := V \setminus S \cup \{\sigma\}$, and let y^* be the shrunk counterpart of x^* . Every valid inequality $\beta y \leq \beta_0$ for $P(G')$ that is violated by y^* corresponds in G to a violated inequality, say $\alpha x \leq \alpha_0$, obtained through clique lifting by replacing back σ with the original set S . As observed by Padberg and Rinaldi [647], however, this shrinking operation can affect the possibility of detecting violated cuts on G' , as it may produce a point y^* belonging to $P(G')$ even when $x^* \notin P(G)$.

For instance, let $n := 4$ and $x_{ij}^* := 1/2$ if $(i, j) \in \{(1, 2), (1, 4), (2, 1), (2, 3), (3, 1), (3, 4), (4, 2), (4, 3)\}$; $x_{ij}^* = 0$ otherwise. One readily checks that $x^* \notin P(G)$, as x^* violates, e.g., the D_3^+ inequality $x_{12} + x_{23} + x_{31} + 2x_{21} \leq 2$. On the other hand, shrinking the saturated set $S := \{1, 2\}$ produces a digraph G' with vertex set $V' := \{\sigma, 3, 4\}$, and a point y^* with $y_{ij}^* = 1/2$ for all $(i, j) \in A'$. But then y^* is the convex combination of the characteristic vectors of the two tours $(\sigma, 3, 4)$ and $(\sigma, 4, 3)$, hence y^* cannot be cut off by any linear inequality as it belongs to $P(G')$.

The above example shows that shrinking has to be applied with some care. There are however simple conditions on the choice of S that guarantee $y^* \notin P(G')$, provided $x^* \notin P(G)$ as in the cases of interest for separation.

The simplest such condition concerns the shrinking of *1-arcs* (i.e., arcs (i, j) with $x_{ij}^* = 1$), and requires $S = \{i, j\}$ for a certain node pair i, j with $x_{ij}^* = 1$. To see the validity of the condition, assume by contradiction that $y^* \in P(G')$. This implies $y^* = \sum_{k=1}^M \lambda_k y^k$, where y^1, \dots, y^M are characteristic vectors of tours in G' , and $\lambda_1, \dots, \lambda_M$ are nonnegative multipliers with $\sum_{k=1}^M \lambda_k = 1$. For each $k \in \{1, \dots, M\}$ we define x^k as the characteristic vector of the tour of G obtained from y^k by replacing node σ with the arc (i, j) . Then, by construction, $x^* = \sum_{k=1}^M \lambda_k x^k$, which contradicts the assumption $x^* \notin P(G)$.

It is known that 1-edges cannot be shrunk for *STSP*, instead. In this respect *ATSP* behaves more nicely than *STSP*, in that the informa-

tion associated with the orientation of the arcs allows for more powerful shrinkings. Here is a polyhedral interpretation of this behavior.

In the separation problem, we are given a point x^* which satisfies the valid inequality $x_{ij} \leq 1$ with equality, and we want to separate it from the ATSP polytope, P . The above discussion shows that this is possible if and only if x^* can be separated from $F := \{x \in P : x_{ij} = 1\}$, hence F can replace P insofar the separation of x^* is concerned. This property is perfectly general, and applies to any nonempty face F of any polytope P . Indeed, let $F := \{x \in P : \beta x = \beta_0\} \neq \emptyset$ be the face of P induced by any valid inequality $\beta x \leq \beta_0$ for P , and assume that $\beta x^* = \beta_0$. If x^* cannot be separated from F , then $x^* \in F \subseteq P$, hence it cannot be separated from P too. Conversely, suppose there exists a valid inequality $\alpha x \leq \alpha_0$ for F which is violated by x^* by the amount $\delta^* := \alpha x^* - \alpha_0 > 0$. Then for a sufficiently large value M the “lifted” inequality $(\alpha + M\beta)x \leq \alpha_0 + M\beta_0$ is valid for P , but violated by x^* by the amount $(\alpha + M\beta)x^* - (\alpha_0 + M\beta_0) = \alpha x^* - \alpha_0 = \delta^* > 0$.

The different behavior between the asymmetric and symmetric TSP polytopes then has its roots in the basic property that fixing $x_{ij} = 1$ for some ATSP arc (i, j) yields again an ATSP instance – obtained by contracting (i, j) into a single vertex – whereas the same construction does not work when fixing $y_e = 1$ for some STSP undirected edge e . In polyhedral terms, this means that the face F of the ATSP polytope induced by $x_{ij} \leq 1$ is in 1-1 correspondence with the ATSP polytope on $n-1$ nodes. Hence, in the asymmetric case, the face F can be interpreted again as an ATSP polytope on a shrunken graph, whereas for STSP a similar interpretation is not possible.

In [306] 1-arc shrinking is applied iteratively, so as to replace each path of 1-arcs by a single node. As a result of this pre-processing on x^* , all the nonzero variables are fractional. Notice that a similar result cannot be obtained for the symmetric TSP, where each 1-edge chain can be replaced by a single 1-edge, but not by a single node.

More sophisticated shrinking procedures have not been used in [306]. The above discussion suggests however other cases in which a saturated set S can be shrunk. For instance, suppose there exist 3 distinct nodes i, j and k such that $x_{ij}^* + x_{ji}^* = 1$ and $x_{jk}^* + x_{kj}^* = 1$, i.e., $x_{ij}^* = x_{jk}^* = \mu$ and $x_{kj}^* = x_{ji}^* = 1 - \mu$ for some $0 \leq \mu \leq 1$. We claim that in this case the saturated set $S = \{i, j\}$ can be shrunk. Indeed, the given point x^* satisfies the valid ATSP inequality $\beta x := x_{ij} + x_{ji} + x_{jk} + x_{kj} \leq \beta_0 := 2$ with equality, hence from the above discussion one can replace P with its face $F := \{x \in P : \beta x = \beta_0\}$. Now, every extreme point of F corresponds to a tour using either the path $\{(i, j), (j, k)\}$ or the path $\{(k, j), (j, i)\}$. This property induces a 1-1 correspondence between the

extreme points of F and those of the $ATSP$ polytope in which i and j have been shrunk into a single node.

4.5. Pricing with Degeneracy

Pricing is an important ingredient of branch-and-cut codes, in that it allows one to effectively handle LP problems involving a huge number of columns. Let

$$z := \min\{cx : Mx \equiv b, x \geq 0\} \tag{16}$$

be the LP problem to be solved. M is an $m \times |A|$ matrix whose columns are indexed by the arcs $(i, j) \in A$. The first $2n - 1$ rows of M correspond to the degree equations (2)-(3) (with the redundant constraint $x(1, V) = 1$ omitted), whereas the remaining rows, if any, correspond to some of the cuts generated through separation. Notation “ \equiv ” stands for “ $=$ ” for the first $2n - 1$ rows of M , and “ \leq ” for the remaining rows. Let M_{ij}^h denote the entry of M indexed by row h and column (i, j) .

In order to keep the size of the LP as small as possible, the following *pricing* scheme is commonly used. We determine a (small) *core* set of arcs, say \tilde{A} , and decide to temporarily fix $x_{ij} = 0$ for all $(i, j) \in A \setminus \tilde{A}$. We then solve the restricted LP problem

$$\tilde{z} := \min\{\tilde{c}\tilde{x} : \tilde{M}\tilde{x} \equiv b, \tilde{x} \geq 0\} \tag{17}$$

where \tilde{c} , \tilde{x} , and \tilde{M} are obtained from c , x , and M , respectively, by removing all entries indexed by $A \setminus \tilde{A}$.

Assume problem (17) is feasible, and let \tilde{x}^* and \tilde{u}^* be the optimal primal and dual basic solutions found, respectively. Clearly, $\tilde{z} \geq z$. We are interested in easily-checkable conditions that guarantee $\tilde{z} = z$, thus proving that \tilde{x}^* (with $\tilde{x}_{ij}^* := 0$ for all $(i, j) \in A \setminus \tilde{A}$) is an optimal basic solution to (16), and hence that its value \tilde{z} is a valid lower bound on $v(ATSP)$. To this end we compute the LP reduced costs associated with \tilde{u}^* , namely

$$\bar{c}_{ij} := c_{ij} - \sum_{h=1}^m M_{ij}^h \tilde{u}_h^* \quad \text{for } (i, j) \in A$$

and check whether $\bar{c}_{ij} \geq 0$ for all $(i, j) \in A$. If this is indeed the case, then $\tilde{z} = z$ and we are done. Otherwise, the current core set \tilde{A} is enlarged by adding (some of) the arcs with negative reduced cost, and the whole procedure is iterated. This iterative solution of (17), followed by the possible updating of \tilde{A} , is generally referred to as the *pricing loop*.

According to common computational experience, the first iterations of the pricing loop tend to add a very large number of new columns to

the LP even when $\tilde{z} = z$, due to the typically high primal degeneracy of (17).

As an illustration of this behavior, consider the situation arising when the first LP is solved at the root node of the branching tree. In this case (16) contains the $2n - 1$ degree equations only, hence its optimal solution, x^* , can be computed efficiently through any *AP* code. Suppose we now initialize the core set \tilde{A} to contain the n arcs chosen in the optimal *AP* solution. In order to have an LP basis, we add $n - 1$ additional arcs to \tilde{A} , chosen so as to determine an $(2n - 1) \times (2n - 1)$ nonsingular matrix \tilde{M} . By construction, problem (17) has a unique feasible solution – namely, the characteristic vector of the optimal *AP* solution found – hence we know that $\tilde{z} = z$ holds in this case. However, depending on the possibly “wrong” choice of the last $n - 1$ arcs in the core set, the solution \tilde{u}^* can be dual infeasible for (16), i.e., a usually very large number of reduced costs \bar{c}_{ij} are negative. Iterating the pricing procedure produces a similar behavior, and a long sequence of pricings is typically required before all arcs price-out correctly.

The above example shows that checking the reduced-cost signs can lead to an overweak sufficient condition for proving $\tilde{z} = z$. The standard way to cope with this weakness consists in a more careful initialization of the core set, e.g., by taking the 15 smallest-cost arcs leaving each node.

We next describe a different technique, called *AP pricing* in [306], in which the pricing condition is strengthened by exploiting the fact that any feasible solution to (16) cannot select the arcs with negative reduced cost in an arbitrary way, as the degree equations —among other constraints— have to be fulfilled. The technique is related to the so-called *Lagrangian pricing* introduced independently by Löbel [566] as a powerful method for solving large-scale vehicle scheduling problems.

Let us consider the dual solution \tilde{u}^* to (17) as a vector of Lagrangian multipliers, and the LP reduced costs \bar{c}_{ij} as the corresponding Lagrangian costs. In this view, standard pricing consists of solving the following trivial relaxation of (16):

$$LB_1 := \min_{x \geq 0} [cx + \tilde{u}^*(b - Mx)] = \tilde{u}^*b + \min_{x \geq 0} \bar{c}x \quad (18)$$

where $\tilde{u}^*b = \tilde{z}$ by LP duality. Therefore one has $\tilde{z} + \min_{x \geq 0} \bar{c}x \leq z \leq \tilde{z}$, from which $\tilde{z} = z$ in case $\min_{x \geq 0} \bar{c}x = 0$, i.e., $\bar{c}_{ij} \geq 0$ for all i, j . The strengthening then consists in replacing condition $x \geq 0$ in (18) by

$$x \in F(AP) := \{x \in \{0, 1\}^A : x(i, V) = x(V, i) = 1 \text{ for all } i \in V\}$$

In this way we compute an improved lower bound on z , namely

$$LB_2 := \tilde{u}^*b + \min_{x \in F(AP)} \bar{c}x = \tilde{z} + \Delta_{AP}$$

where $\Delta_{AP} := \min_{x \in F(AP)} \bar{c}x$ is computed efficiently by solving the AP on the Lagrangian costs \bar{c}_{ij} . As before, $\tilde{z} + \Delta_{AP} \leq z \leq \tilde{z}$, hence $\Delta_{AP} = 0$ implies $\tilde{z} = z$. When $\Delta_{AP} < 0$, instead, one has to iterate the procedure, after having added to the core set \tilde{A} the arcs in $A \setminus \tilde{A}$ that are selected in the optimal AP solution found.

The new approach has two main advantages, namely: (1) an improved check for proving $\tilde{z} = z$; and (2) a better rule to select the arcs to be added to the core arc set. Moreover, LB_2 always gives a lower bound on z (and hence on $v(ATSP)$), which can in some cases succeed in fathoming the current branching node even when $\Delta_{AP} < 0$. Finally, the nonnegative AP reduced cost vector \bar{c} available after solving $\min_{x \in F(AP)} \bar{c}x$ can be used for fixing $x_{ij} = 0$ for all $(i, j) \in A$ such that $LB_2 + \bar{c}_{ij}$ is at least as large as the value of the best known $ATSP$ solution.

The new pricing scheme can be adapted to other problems having an easily-solvable relaxation. For example, in Fischetti and Vigo [307] the approach is applied to the resource constrained arborescence problem, the relaxation used for pricing being in this case the min-sum arborescence problem. Unlike AP , the latter problem involves exponentially many constraints, hence the pricing scheme can in some cases also detect violated cuts that are not present in the current LP, chosen from among those implicitly used during the solution of the relaxation. In other words, variable-pricing also produces, as a by-product, a heuristic “pricing” of some exponential classes of cuts, i.e., a separation tool.

AP pricing requires the computation of all reduced costs \bar{c}_{ij} , e.g., through the following “row-by-row” scheme. We initialize $\bar{c}_{ij} := c_{ij}$ for all $(i, j) \in A$ and then consider, in sequence, the rows h of M with $\tilde{u}_h^* \neq 0$. For each such row h , we determine the set $A_h := \{(i, j) \in A : M_{ij}^h \neq 0\}$, and update $\bar{c}_{ij} := \bar{c}_{ij} - \tilde{u}_h^* M_{ij}^h$ for $(i, j) \in A_h$. Because of the simple combinatorial structure of most cuts, the (implicit) construction of A_h can typically be carried out in $O(|A_h|)$ time, hence the overall time spent for computing all reduced costs is $O(n^2)$ for the initialization, plus $O(\sum_{h=1}^m |A_h|)$ for the actual reduced-cost computation (notice that this latter term is linear in the number of nonzero entries in M).

A drawback of the AP pricing is the extra computing time spent for the AP solution, which can however be reduced considerably through the following strategy [306]. After each LP solution we compute the reduced costs \bar{c}_{ij} and determine the cardinality μ of $A^- := \{(i, j) \in A : \bar{c}_{ij} < 0\}$. If $\mu = 0$, we exit the pricing loop. If $0 < \mu \leq n/10$, we use standard pricing, i.e., we update $\tilde{A} := \tilde{A} \cup A^-$ and repeat. If $\mu > n/10$, instead, we resort to AP pricing, and solve the AP problem on the reduced costs. We also determine (through a technique described, e.g., in [306]) the arc set Q containing the $2n - 1$ arcs defining an optimal LP basis for

this AP problem, and compute the corresponding nonnegative reduced costs \bar{c}_{ij} for all $(i, j) \in A$. We then remove from Q all the arcs already in \tilde{A} , and enlarge Q by iteratively adding arcs in $(i, j) \in A \setminus (\tilde{A} \cup Q)$ with $\bar{c}_{ij} = 0$, until no such arc exists, or $|Q| \geq 50 + n/3$. In this way Q contains a significant number of arcs that are likely to be selected in (16). We finally update $\tilde{A} := \tilde{A} \cup Q$ (even in case $\Delta_{AP} = 0$), and repeat (if $\Delta_{AP} < 0$) or exit (if $\Delta_{AP} = 0$) the pricing loop.

4.6. The Overall Algorithm

The algorithm is a lowest-first branch-and-cut procedure. At each node of the branching tree, the LP relaxation is initialized by taking all the constraints present in the last LP solved at the father node (for the root node, only the degree equations are taken). As to variables, one retrieves from a scratch file the optimal basis associated with the last LP solved at the father node, and initializes the core variable set, \tilde{A} , by taking all the arcs belonging to this basis (for the root node, \tilde{A} contains the $2n - 1$ variables in the optimal AP basis found by solving AP on the original costs c_{ij}). In addition, \tilde{A} contains all the arcs of the best known $ATSP$ solution. Starting with the above advanced basis, one iteratively solves the current LP, applies the AP pricing (and variable fixing) procedure described in Section 4.5, and repeats if needed. Observe that the pricing/fixing procedure is applied after each LP solution.

On exit of the pricing loop (case $\Delta_{AP} = 0$), the cuts whose associated slack exceeds 0.01 are removed from the current LP (unless the number of these cuts is less than 10), and the LP basis is updated accordingly. Moreover, separation algorithms are applied to find, if any, facet-defining $ATSP$ inequalities that cut off the current LP optimal solution, say x^* . As a heuristic rule, the violated cuts with degree of violation less than 0.1 (0.01 for SECs) are skipped, and the separation phase is interrupted as soon as $20 + \lfloor n/5 \rfloor$ violated cuts are found.

One first checks for violation the cuts generated during the processing of the current or previous nodes, all of which are stored in a global data-structure called the constraint *pool*. If some of these cuts are indeed violated by x^* , the separation phase ends. Otherwise, the Padberg-Rinaldi [646] MINCUT algorithm for SEC separation is applied, and the separation phase is interrupted if violated SECs are found. When this is not the case, one shrinks the 1-arc paths of x^* (as described in Section 4.4), and applies the separation algorithms for comb (Section 4.1), D_k^+ and D_k^- (Section 4.2), and odd CAT (Section 4.3) inequalities. In order to avoid finding equivalent inequalities, D_3^- inequalities (which are the same as D_3^+ inequalities), are never separated, and odd CAT separation

is skipped when a violated comb is found (as the class of comb and odd CAT inequalities overlap). When violated cuts are found, one adds them to the current LP, and repeats.

When separation fails and x^* is integer, the current best *ATSP* solution is updated, and a backtracking step occurs. If x^* is fractional, instead, the current LP basis is saved in a file, and one branches on the variable x_{ij} with $0 < x_{ij}^* < 1$ that maximizes the score $\sigma(i, j) := c_{ij} \cdot \min\{x_{ij}^*, 1 - x_{ij}^*\}$. As a heuristic rule, a large priority is given to the variables with $0.4 \leq x_{ij}^* \leq 0.6$ (if any), so as to produce a significant change in both descending nodes.

As a heuristic tailing-off rule, one also branches when the current x^* is fractional and the lower bound did not increase in the last 5 (10 for the root node) LP/pricing/separation iterations.

A simple heuristic algorithm is used to hopefully update the current best optimal *ATSP* solution. The algorithm is based on the information associated with the current LP, and consists of a complete enumeration of the Hamiltonian directed cycles in the support graph of x^* , defined as $G^* := (V, \{(i, j) \in A : x_{ij}^* > 0\})$. To this end Martello's [585] implicit enumeration algorithm HC is used, with at most $100 + 10n$ backtracking steps allowed. As G^* is typically very sparse, this upper bound on the number of backtrackings is seldom attained, and HC almost always succeeds in completing the enumeration within a short computing time. The heuristic is applied whenever SEC separation fails, since in this case G^* is guaranteed to be strongly connected.

5. Computational Experiments

The algorithms described in the previous sections have been computationally tested on a large set of *ATSP* instances namely:

- 42 instances by Cirasella, Johnson, McGeoch and Zhang [200]; the instances in this set are randomly generated to simulate real-world applications arising in many different fields;
- 5 scheduling instances provided by Balas [67];
- 2 additional real-world instances (ftv180 and uk66);
- 10 random instances whose integer costs are uniformly generated in range $[1, 1000]$;
- all the 27 *ATSP* instances collected in TSPLIB [709].

For a detailed description of the first 42 instances the reader is referred to [200], while details for the ones in the TSPLIB can be found in the

associated web page [709]. The 5 instances provided by Balas come from *Widget*, a generator of realistic instances from the chemical industry developed by Donald Miller. Instance `ftv180` represents pharmaceutical product delivery within Bologna down-town and is obtained from the TSPLIB instance `ftv170` by considering 10 additional vertices in the graph representing down-town Bologna. Finally, instance `uk66` is a real-world problem arising in routing applications and has been provided us by Kousgaard [518].

All instances have integer nonnegative costs, and are available, on request, from the authors. For each instance we report in Table 4.1 the name (Name), the size (n), the optimal (or best known) solution value (OptVal), and the source of the instance (source).

Three specific *ATSP* codes have been tested: (1) the *AP*-based branch-and-bound CDT code [163], as described in Section 2.1; (2) **FT-add** code, corresponding to the additive approach [304] described in Section 3; and (3) **FT-b&c** code, corresponding to the branch-and-cut algorithm [306] described in Section 4.

In addition, the branch-and-cut code **Concorde** by Applegate, Bixby, Chvátal and Cook [29] has been considered, as described in Chapter 2. This code is specific for the symmetric *TSP*, so symmetric instances have to be constructed through one of the following two transformations:

- the *3-node* transformation proposed by Karp [495]. A complete *undirected* graph with $3n$ vertices is obtained from the original complete *directed* one by adding two copies, $n + i$ and $2n + i$, of each vertex $i \in V$, and by (i) setting to 0 the cost of the edges $(i, n + i)$ and $(n + i, 2n + i)$ for each $i \in V$, (ii) setting to c_{ij} the cost of edge $(2n + i, j) \forall i, j \in V$, and (iii) setting to $+\infty$ the costs of all the remaining edges;
- the *2-node* transformation proposed by Jonker and Volgenant [471] (see also Jünger, Reinelt and Rinaldi [474]). A complete *undirected* graph with $2n$ vertices is obtained from the original complete *directed* one by adding a copy, $n + i$, of each vertex $i \in V$, and by (i) setting to 0 the cost of the edge $(i, n + i)$ for each $i \in V$, (ii) setting to $c_{ij} + M$ the cost of edge $(n + i, j) \forall i, j \in V$, where M is a sufficiently large positive value, and (iii) setting to $+\infty$ the costs of all the remaining edges. The transformation value nM has to be subtracted from the *STSP* optimal cost.

All tests have been executed on a Digital Alpha 533 MHz with 512 MB of RAM memory under the Unix Operating System, with Cplex 6.5.3 as LP solver. In all tables, we report the percentage gaps corresponding to the

Table 4.1. ATSP instances.

Name	n	OptVal	source	Name	n	OptVal	source
coin100.0	100	10360	[200]	balas84	84	199	[67]
coin100.1	100	10600	[200]	balas108	108	152	[67]
coin100.2	100	10980	[200]	balas120	120	286	[67]
coin100.3	100	10540	[200]	balas160	160	397	[67]
coin100.4	100	10440	[200]	balas200	200	403	[67]
coin316.10	316	32360	[200]	ftv180	181	2918	this Chapter
crane100.0	100	7777997	[200]	uk66	66	2791	[518]
crane100.1	100	7615069	[200]	ran1000.0	1000	7897	$c_{ij} \in [1, 1000]$
crane100.2	100	8062054	[200]	ran1000.1	1000	8003	$c_{ij} \in [1, 1000]$
crane100.3	100	7018782	[200]	ran1000.2	1000	7991	$c_{ij} \in [1, 1000]$
crane100.4	100	7786309	[200]	ran1000.3	1000	7956	$c_{ij} \in [1, 1000]$
crane316.10	316	13132907	[200]	ran1000.4	1000	8014	$c_{ij} \in [1, 1000]$
disk100.0	100	17471906	[200]	ran500.0	500	5507	$c_{ij} \in [1, 1000]$
disk100.1	100	18186198	[200]	ran500.1	500	5398	$c_{ij} \in [1, 1000]$
disk100.2	100	15568681	[200]	ran500.2	500	5394	$c_{ij} \in [1, 1000]$
disk100.3	100	18326394	[200]	ran500.3	500	5354	$c_{ij} \in [1, 1000]$
disk100.4	100	17862733	[200]	ran500.4	500	5337	$c_{ij} \in [1, 1000]$
disk316.10	316	28904480	[200]	br17	17	39	TSPLIB
rtilt100.0	100	9465148	[200]	ft53	53	6905	TSPLIB
rtilt100.1	100	9623330	[200]	ft70	70	38673	TSPLIB
rtilt100.2	100	9411004	[200]	ftv33	34	1286	TSPLIB
rtilt100.3	100	9584646	[200]	ftv35	36	1473	TSPLIB
rtilt100.4	100	10265172	[200]	ftv38	39	1530	TSPLIB
rtilt316.10	316	16738334	[200]	ftv44	45	1613	TSPLIB
shop100.0	100	143019	[200]	ftv47	48	1776	TSPLIB
shop100.1	100	147815	[200]	ftv55	56	1608	TSPLIB
shop100.2	100	148602	[200]	ftv64	65	1839	TSPLIB
shop100.3	100	148413	[200]	ftv70	71	1950	TSPLIB
shop100.4	100	144270	[200]	ftv90	91	1579	TSPLIB
shop316.10	316	427004	[200]	ftv100	101	1788	TSPLIB
stilt100.0	100	15214154	[200]	ftv110	111	1958	TSPLIB
stilt100.1	100	15543791	[200]	ftv120	121	2166	TSPLIB
stilt100.2	100	15199456	[200]	ftv130	131	2307	TSPLIB
stilt100.3	100	15499387	[200]	ftv140	141	2420	TSPLIB
stilt100.4	100	16303671	[200]	ftv150	151	2611	TSPLIB
stilt316.10	316	26715915*	[200]	ftv160	161	2683	TSPLIB
super100.0	100	785	[200]	ftv170	171	2755	TSPLIB
super100.1	100	780	[200]	kro124p	100	36230	TSPLIB
super100.2	100	780	[200]	p43	43	5620	TSPLIB
super100.3	100	776	[200]	rbg323	323	1326	TSPLIB
super100.4	100	809	[200]	rbg358	358	1163	TSPLIB
super316.10	316	2109	[200]	rbg403	403	2465	TSPLIB
				rbg443	443	2720	TSPLIB
				ry48p	48	14422	TSPLIB

*best know solution value.

lower bound at the root node (Root), the final lower bound (fLB), and the final upper bound (fUB), all computed with respect to the optimal (or best known) solution value. Moreover, the number of nodes of the search tree (Nodes), and the computing time in seconds (Time) are given.

5.1. Code Tuning

In this section we analyze variants of the above mentioned codes with the aim of determining, for each code, the best average behavior.

CDT Code. No specific modifications of this code have been implemented.

FT-add Code. The original implementation emphasized the lower bounding procedures. As in the CDT code, improved performances of the overall algorithm have been obtained by using the patching heuristic of Karp [498] (applied at each node), and the subtour merging operation described in Section 2.1.3. This leads to a reduction of both the number of branching nodes and the overall computing time; in particular, instances `balas84` and `ftv160` could not be solved by the original code within the time limit of 1,000 CPU seconds.

FT-b&c Code. A new branching criterion, called *Fractionality Persistence* (FP) has been tested. Roughly speaking, the FP criterion gives priority for branching to the variables that have been persistently fractional in the last LP optimal solutions determined at the current branching node. This general strategy has been proposed and computationally analyzed in [298]. Table 4.2 compares the original and modified codes on a relevant subset of instances when imposing a time limit of 10,000 CPU seconds. According to these results, the FP criterion tends to avoid pathological situations due to branching (two more instances solved to optimality), and leads to a more robust code.

Concorde Code. The code has been used with default parameters by setting the *random seed* parameter (“-s 123”) so as to be able to reproduce each run. Both *ATSP-to-STSP* transformations have been tested by imposing a time limit of 10,000 CPU seconds. For the *2-node* transformation, parameter M has been set to 1,000,000 (but for the instances with $n = 1,000$ and for instance `rtilt316.10`, for which we used $M = 100,000$ so as to avoid numerical problems). The comparison of the results obtained by *Concorde* by using the two transformations is given in Table 4.3 (first two sections), where the same (hard) instances of Table 4.2 are considered. According to these results, no dominance ex-

Table 4.2. FT-b&c without and with Fractionality Persistency (10,000 seconds time limit).

Name	FT-b&c					FT-b&c + FP				
	%Gap			Nodes	Time	%Gap			Nodes	Time
Root	fLB	fUB	Root			fLB	fUB			
coin100.0	0.40	-	-	41	12.4	0.40	-	-	41	8.6
coin316.10	1.04	0.53	0.19	1296	10000.5	1.04	0.47	0.19	1407	10000.2
crane100.2	1.35	-	-	829	297.3	1.35	-	-	707	204.8
crane100.4	1.73	-	-	1005	210.1	1.73	-	-	751	117.6
crane316.10	0.62	0.10	0.01	2894	10000.0	0.62	0.11	0.06	2851	10000.2
rtilt100.0	0.80	-	-	417	197.2	0.80	-	-	501	227.6
rtilt100.4	0.11	-	-	13	5.4	0.11	-	-	13	4.6
rtilt316.10	0.25	0.04	0.00	1439	10000.1	0.25	-	-	1727	8887.1
stilt100.0	1.10	-	-	173	65.6	1.10	-	-	171	49.2
stilt100.3	1.73	-	-	377	157.0	1.73	-	-	325	141.3
stilt100.4	1.58	-	-	2211	1457.7	1.58	-	-	1925	1299.5
stilt316.10	2.35	1.67	19.59	1235	10000.1	2.35	1.66	19.59	1205	10000.3
balas84	1.01	-	-	93	36.3	1.01	-	-	61	15.7
balas108	1.97	-	-	215	89.8	1.97	-	-	267	89.0
balas120	1.05	-	-	662	506.2	1.05	-	-	1339	1276.3
balas160	1.26	-	-	437	522.4	1.26	-	-	737	671.1
balas200	1.24	-	-	1241	1801.2	1.24	-	-	1495	1712.8
ftv180	1.20	0.51	0.00	5226	6911.1°	1.20	-	-	939	366.0
ran1000.0	0.00	-	-	1	3.6	0.00	-	-	1	3.4
ran1000.1	0.00	-	-	3	25.6	0.00	-	-	3	23.5
ran1000.2	0.00	-	-	14	103.2	0.00	-	-	14	150.7
ran1000.3	0.01	-	-	17	98.3	0.01	-	-	11	62.2
ran1000.4	0.01	-	-	16	203.0	0.01	-	-	18	148.7
ran500.3	0.02	-	-	59	55.4	0.02	-	-	65	55.0

°execution aborted for search-tree space limit.

ists between the two transformations with respect to neither the quality of the lower bound at the root node nor the computing time. However, by considering the average behavior, the 2-node transformation seems to be preferable.

Although a fine tuning of the *Concorde* parameters is out of the scope of this section, Table 4.3 also analyzes the code sensitivity to the “chunk size” parameter (last three sections), which controls the implementation of the *local cuts* paradigm [29] used for separation (see Chapter 2 for details). In particular, setting this size to 0 (“-C 0”) disables the generation of the “local” cuts and lets *Concorde* behave as a pure *STSP* code, whereas options “-C 16” (the default) and “-C 24” allow for the generation of additional cuts based on the “instance-specific” enumeration of partial solutions over vertex subsets of size up to 16 and 24, respec-

Table 4.3. Sensitivity of Concorde to 3- and 2-node transformations and to the chunk size parameter.

Name	(3-node) Concorde -s 123 -C 16					(2-node) Concorde -s 123 -C 16				
	Root	%Gap		Nodes	Time	Root	%Gap		Nodes	Time
		fLB	fUB				fLB	fUB		
coin100.0	0.30	-	-	25	185.7	0.29	-	-	25	106.3
coin316.10	0.80	0.31	0.06	227	10139.1	0.85	0.33	0.25	233	10101.2
crane100.2	0.69	-	-	39	347.5	0.94	-	-	41	355.5
crane100.4	1.29	-	-	23	238.9	1.27	-	-	27	277.4
crane316.10	0.34	-	-	287	5142.3	0.34	-	-	255	4287.2
rtilt100.0	0.92	-	-	23	191.7	0.94	-	-	23	129.2
rtilt100.4	0.22	-	-	15	60.8	0.27	-	-	9	26.8
rtilt316.10	0.08	-	-	27	543.3	0.12	-	-	37	593.5
stilt100.0	0.92	-	-	99	760.9	0.83	-	-	69	524.3
stilt100.3	1.49	-	-	77	769.8	1.39	-	-	75	662.5
stilt100.4	1.16	-	-	209	2271.5	1.21	-	-	179	1695.3
stilt316.10	1.99	1.40	7.68	253	10140.0	2.07	1.44	2.25	283	10132.0
balas84	1.01	-	-	31	106.8	1.01	-	-	25	78.0
balas108	2.63	-	-	497	1534.8	2.63	-	-	423	1416.0
balas120	1.05	-	-	633	5694.7	1.05	-	-	755	7186.9
balas160	1.26	-	-	541	6716.8	1.26	-	-	739	7848.0
balas200	0.50	-	-	123	1392.3	0.74	-	-	239	2294.2
ftv180	0.65	-	-	21	337.0	0.69	-	-	29	236.2
ran1000.0	0.00	-	-	61	1146.6	0.00	-	-	21	219.2
ran1000.1	0.00	-	-	35	689.7	0.00	-	-	19	191.7
ran1000.2	0.00	-	-	27	767.8	0.00	-	-	95	900.4
ran1000.3	0.01	-	-	429	7569.0	0.01	-	-	343	3977.2
ran1000.4	0.05	0.00	0.04	445	10129.7	0.01	-	-	379	3122.2
ran500.3	0.02	-	-	79	579.4	0.04	-	-	75	232.4

Name	(2-node) Concorde -s 123 -C 0					(2-node) Concorde -s 123 -C 24				
	Root	%Gap		Nodes	Time	Root	%Gap		Nodes	Time
		fLB	fUB				fLB	fUB		
coin100.0	0.82	-	-	49	88.4	0.28	-	-	15	364.5
coin316.10	1.15	0.42	0.06	567	10026.0	0.73	0.41	0.25	77	10957.5
crane100.2	2.00	-	-	141	272.0	0.44	-	-	15	1096.7
crane100.4	1.96	-	-	79	158.2	0.88	-	-	17	1148.5
crane316.10	0.98	-	-	771	5303.3	0.30	-	-	119	4931.6
rtilt100.0	1.50	-	-	59	124.6	1.04	-	-	29	836.9
rtilt100.4	0.58	-	-	9	26.4	0.17	-	-	3	257.4
rtilt316.10	0.26	-	-	51	304.5	0.06	-	-	13	2826.3
stilt100.0	1.64	-	-	203	333.3	0.70	-	-	39	2465.9
stilt100.3	2.24	-	-	535	1272.5	1.19	-	-	65	3433.6
stilt100.4	2.03	-	-	737	2350.9	0.95	-	-	129	6688.3
stilt316.10	2.27	1.74	5.44	841	10041.3	1.85	1.37	0.00	75	10663.4
balas84	1.01	-	-	17	15.2	1.01	-	-	33	547.5
balas108	2.63	-	-	1023	1269.9	2.63	-	-	343	4110.5
balas120	2.10	0.00	1.40	2849	10007.3	1.05	-	-	221	6244.1
balas160	2.02	0.25	4.03	2165	10007.1	1.01	0.00	2.27	273	10129.5
balas200	2.48	0.74	5.96	1955	10008.0	0.50	-	-	99	2918.8
ftv180	1.58	-	-	91	204.0	0.38	-	-	17	955.9
ran1000.0	0.00	-	-	11	145.2	0.00	-	-	11	186.7
ran1000.1	0.00	-	-	15	136.9	0.00	-	-	21	230.7
ran1000.2	0.00	-	-	3	79.5	0.00	-	-	33	354.1
ran1000.3	0.03	-	-	387	2836.9	0.01	-	-	393	5065.2
ran1000.4	0.01	-	-	153	957.1	0.01	-	-	177	1749.7
ran500.3	0.04	-	-	93	225.9	0.04	-	-	103	538.8

tively. In this way, we aimed at analyzing the capability of the “local” cuts method to automatically generate cuts playing an important role for the *ATSP* instance to be solved. The results of Table 4.3 show that the “local” cuts (“-C 16” and “-C 24”) play a very important role, allowing one to solve to optimality difficult instances (e.g., instances *balas120*, *balas160*, and *balas200* are solved with chunk equal to 16, and remain unsolved without “local” cuts). Not surprisingly, the main exception concerns the uniformly-random instances (*ran1000*) for which even the *AP* bound is already very tight. Among the chunk sizes, even after the 2-node transformation, the best choice appears to be the default (“-C 16”) which gives the best compromise between the separation overhead and the lower bound improvement. We have also tested chunk sizes 8 and 32, without obtaining better results. (The 3-node transformation exhibits a similar behavior.)

5.2. Code Comparison

A comparison of the four algorithms, namely, *CDT*, *FT-add* (modified version), *FT-b&c* (“+ FP” version) and *Concorde* (“(2-node) -C 16” version), is given in Tables 4.4, 4.5, 4.6 and 4.7 on the complete set of 86 instances. As to time limit, we imposed 1,000 CPU seconds for *CDT* and *FT-add*, and 10,000 CPU seconds for *FT-b&c* and *Concorde*. The smaller time limit given to the first two algorithms is chosen so as to keep the required search-tree space reasonable, but it does not affect the comparison with the other algorithms: according to preliminary computational experiments on some hard instances, either the branch-and-bound approaches solve an instance within 1,000 CPU seconds, or the final gap is too large to hope in a convergence within 10,000 seconds.

As to the lower bound at the root node, the tables show that the additive approach obtains significantly better results than the *AP* lower bound, but is dominated by both cutting plane approaches. In its pure *STSP* version (“-C 0”), the *Concorde* code obtains a root-node lower bound which is dominated by the *FT-b&c* one, thus showing the effectiveness of addressing the *ATSP* in its original (directed) version. Of course, one can expect to improve the performance of *FT-b&c* by exploiting additional classes of *ATSP*-specific cuts such as the lifted cycle inequalities described in Chapter 3. As to *Concorde*, we observe that the use of the “local” cuts leads to a considerable improvement of the root-node lower bound. Not surprisingly, this improvement appears more substantial than in the case of pure *STSP* instances: in our view, this is again an indication of the importance of exploiting the structure of the original asymmetric problem, which results into a very special structure

Table 4.4. Comparison of branch-and-bound codes. Time limit of 1,000 seconds.

Name	CDT					FT-add					
	Root	%Gap		fUB	Nodes	Time	Root	%Gap		fUB	Nodes
coin100.0	12.93	5.41	23.17	192324	1000.0	4.28	0.77	0.19	185361	1000.0	
coin100.1	15.66	9.62	21.51	210379	1000.0	3.68	1.89	1.51	154925	1000.0	
coin100.2	14.57	7.65	18.40	215216	1000.0	5.24	1.82	0.36	162688	1000.0	
coin100.3	20.68	12.52	20.49	214915	1000.0	5.02	1.33	0.57	166505	1000.0	
coin100.4	13.03	6.70	14.37	219186	1000.0	5.91	1.34	1.72	167641	1000.0	
coin316.10	14.94	12.22	19.14	229130	1000.0	6.96	5.19	14.01	13470	1000.0	
crane100.0	8.55	1.58	8.08	164790	1000.0	4.09	0.19	0.00	168273	1000.0	
crane100.1	5.57	1.48	3.43	170445	1000.0	3.12	0.16	0.00	170634	1000.0	
crane100.2	10.30	2.41	8.41	165701	1000.0	5.84	1.06	0.37	103956	1000.0	
crane100.3	7.02	0.65	3.87	155420	1000.0	3.38	-	-	8643	40.7	
crane100.4	8.69	4.20	13.79	162903	1000.0	4.78	2.11	2.15	96990	1000.0	
crane316.10	8.75	5.76	8.14	180724	1000.0	3.66	2.93	1.93	9940	1000.3	
disk100.0	3.00	-	-	5360	0.7	2.31	-	-	538	0.9	
disk100.1	4.34	0.41	1.81	151974	1000.0	2.81	-	-	12706	29.6	
disk100.2	2.75	-	-	465	0.0	2.17	-	-	167	0.7	
disk100.3	1.33	-	-	4909	0.7	1.28	-	-	738	1.5	
disk100.4	2.12	-	-	143	0.0	1.47	-	-	74	0.3	
disk316.10	1.21	-	-	3397	2.1	0.76	-	-	502	43.5	
rtilt100.0	22.59	13.19	10.62	238487	1000.0	7.52	2.93	3.00	96426	1000.0	
rtilt100.1	20.83	12.29	18.83	224840	1000.0	7.38	2.73	1.98	109535	1000.0	
rtilt100.2	23.34	13.36	18.82	215193	1000.0	6.73	2.64	3.30	105315	1000.0	
rtilt100.3	18.86	10.28	26.64	247630	1000.0	3.27	1.76	1.13	115796	1000.0	
rtilt100.4	13.52	7.11	14.05	213573	1000.0	3.89	0.83	0.94	108540	1000.0	
rtilt316.10	18.39	15.14	19.11	207103	1000.0	9.07	6.06	11.95	8314	1000.2	
shop100.0	0.28	-	-	64	0.0	0.14	-	-	81	0.4	
shop100.1	0.67	-	-	3234	1.0	0.38	-	-	587	2.8	
shop100.2	0.98	-	-	1345	0.4	0.41	-	-	990	5.2	
shop100.3	0.36	-	-	604	0.2	0.17	-	-	872	2.9	
shop100.4	0.35	-	-	217	0.1	0.28	-	-	273	1.3	
shop316.10	0.16	-	-	1228	3.1	0.11	-	-	661	91.1	
stilt100.0	22.15	11.36	21.47	213441	1000.0	9.32	2.52	1.62	97255	1000.0	
stilt100.1	19.34	10.55	24.20	213919	1000.0	6.49	2.78	2.49	103875	1000.0	
stilt100.2	23.61	11.21	24.43	199478	1000.0	9.81	3.23	2.84	110778	1000.0	
stilt100.3	17.30	8.86	24.97	198401	1000.0	7.05	3.77	2.91	116219	1000.0	
stilt100.4	13.19	7.35	22.05	202776	1000.0	6.93	2.19	0.60	99972	1000.0	
stilt316.10	16.36	14.05	24.21	174030	1000.0	9.82	6.70	16.39	6508	1000.1	
super100.0	0.25	-	-	36	0.0	0.13	-	-	11	0.1	
super100.1	1.28	-	-	38	0.0	0.26	-	-	72	0.2	
super100.2	1.79	-	-	169	0.1	0.64	-	-	68	0.2	
super100.3	0.90	-	-	69	0.1	0.39	-	-	122	0.3	
super100.4	0.25	-	-	6	0.0	0.25	-	-	16	0.1	
super316.10	1.33	0.57	3.37	103934	1000.0	0.52	-	-	8515	236.7	

Table 4.5. Comparison of branch-and-cut codes. Time limit of 10,000 seconds.

Name	FT-b&c + FP					(2-node) Concorde -s 123 -C 16				
	%Gap			Nodes	Time	%Gap			Nodes	Time
Root	fLB	fUB	Root			fLB	fUB			
coin100.0	0.40	-	-	41	8.6	0.29	-	-	25	106.3
coin100.1	0.47	-	-	29	5.9	0.30	-	-	7	49.1
coin100.2	0.45	-	-	25	9.1	0.13	-	-	3	40.4
coin100.3	0.19	-	-	9	2.8	0.08	-	-	3	21.2
coin100.4	0.48	-	-	17	4.9	1.05	-	-	7	64.1
coin316.10	1.04	0.47	0.19	1407	10000.2	0.85	0.33	0.25	233	10101.2
crane100.0	0.32	-	-	7	1.4	0.00	-	-	1	6.5
crane100.1	0.28	-	-	7	1.5	0.01	-	-	3	15.9
crane100.2	1.35	-	-	707	204.8	0.94	-	-	41	355.5
crane100.3	0.03	-	-	3	0.5	0.00	-	-	1	3.4
crane100.4	1.73	-	-	751	117.6	1.27	-	-	27	277.4
crane316.10	0.62	0.11	0.06	2851	10000.2	0.34	-	-	255	4287.2
disk100.0	0.22	-	-	11	1.0	0.14	-	-	3	7.1
disk100.1	0.35	-	-	29	2.9	0.32	-	-	21	40.4
disk100.2	0.04	-	-	3	0.3	0.04	-	-	3	5.6
disk100.3	0.00	-	-	1	0.1	0.00	-	-	1	2.5
disk100.4	0.00	-	-	3	0.3	0.15	-	-	3	9.3
disk316.10	0.03	-	-	5	5.6	0.04	-	-	7	31.1
rtilt100.0	0.80	-	-	501	227.6	0.94	-	-	23	129.2
rtilt100.1	0.24	-	-	23	6.8	0.09	-	-	5	26.7
rtilt100.2	0.00	-	-	1	0.4	0.00	-	-	1	8.1
rtilt100.3	0.32	-	-	19	4.2	0.00	-	-	1	19.3
rtilt100.4	0.11	-	-	13	4.6	0.27	-	-	9	26.8
rtilt316.10	0.25	-	-	1727	8887.1	0.12	-	-	37	593.5
shop100.0	0.01	-	-	7	0.7	0.03	-	-	13	28.8
shop100.1	0.02	-	-	13	1.4	0.02	-	-	17	39.7
shop100.2	0.02	-	-	11	1.1	0.02	-	-	7	17.7
shop100.3	0.03	-	-	17	1.6	0.03	-	-	15	24.4
shop100.4	0.02	-	-	5	0.5	0.01	-	-	9	16.7
shop316.10	0.01	-	-	17	14.3	0.01	-	-	15	122.2
stilt100.0	1.10	-	-	171	49.2	0.83	-	-	69	524.3
stilt100.1	1.29	-	-	167	57.1	1.13	-	-	23	221.4
stilt100.2	0.57	-	-	15	6.1	0.22	-	-	5	56.9
stilt100.3	1.73	-	-	325	141.3	1.39	-	-	75	662.5
stilt100.4	1.58	-	-	1925	1299.5	1.21	-	-	179	1695.3
stilt316.10	2.35	1.66	19.59	1205	10000.3	2.77	1.44	2.25	283	10132.0
super100.0	0.00	-	-	6	0.5	0.00	-	-	7	6.0
super100.1	0.00	-	-	1	0.2	0.00	-	-	9	8.5
super100.2	0.00	-	-	1	0.1	0.00	-	-	1	1.7
super100.3	0.00	-	-	1	0.2	0.00	-	-	1	3.5
super100.4	0.00	-	-	1	0.1	0.00	-	-	5	6.1
super316.10	0.00	-	-	4	3.9	0.00	-	-	1	14.9

Table 4.6. Comparison of branch-and-bound codes. Time limit of 1,000 seconds.

Name	CDT					FT-add				
	%Gap			Nodes	Time	%Gap			Nodes	Time
Root	fLB	fUB	Root			fLB	fUB			
balas84	14.07	5.53	34.17	192409	1000.0	5.53	-	-	612994	986.6
balas108	25.00	14.47	53.95	166420	1000.0	9.87	1.97	0.66	164537	1000.0
balas120	21.68	12.24	36.71	154920	1000.0	13.29	7.34	7.34	122080	1000.0
balas160	19.40	10.58	43.83	148006	1000.0	11.34	8.31	13.35	70260	1000.0
balas200	15.63	12.66	47.64	141847	1000.0	8.68	6.70	21.09	47711	1000.0
ftv180	5.07	0.65	2.06	150773	1000.0	4.28	0.62	0.10	64938	1000.0
uk66	72.91	5.84	35.51	193383	1000.0	31.71	-	-	50005	152.1
ran1000.0	0.00	-	-	61	0.7	0.00	-	-	811	94.2
ran1000.1	0.05	-	-	50	0.7	0.05	-	-	97	9.1
ran1000.2	0.09	-	-	182	3.9	0.09	-	-	1022	90.0
ran1000.3	0.05	-	-	73	1.1	0.05	-	-	795	42.9
ran1000.4	0.07	-	-	87	1.5	0.07	-	-	577	48.5
ran500.0	0.00	-	-	15	0.1	0.00	-	-	1	0.3
ran500.1	0.06	-	-	11	0.1	0.06	-	-	12	1.1
ran500.2	0.15	-	-	58	0.2	0.15	-	-	160	3.1
ran500.3	0.06	-	-	52	0.1	0.06	-	-	263	6.6
ran500.4	0.07	-	-	37	0.1	0.07	-	-	85	4.6
br17	100.00	-	-	43879	3.6	0.00	-	-	11	0.0
ft53	14.11	5.53	15.28	154920	1000.0	1.56	-	-	131	0.2
ft70	1.80	-	-	2624	0.4	0.57	-	-	220	0.3
ftv33	7.85	-	-	81	0.0	3.73	-	-	120	0.1
ftv35	6.25	-	-	428	0.0	3.53	-	-	647	0.2
ftv38	6.01	-	-	424	0.0	3.01	-	-	854	0.3
ftv44	5.70	-	-	214	0.0	4.46	-	-	289	0.1
ftv47	6.98	-	-	585	0.1	3.49	-	-	1026	0.4
ftv55	10.76	-	-	6989	1.1	6.59	-	-	3253	1.5
ftv64	6.42	-	-	4552	0.8	4.89	-	-	2553	1.3
ftv70	9.44	-	-	10328	3.3	6.92	-	-	4347	3.7
ftv90	6.33	-	-	4220	0.7	3.17	-	-	1670	2.5
ftv100	6.60	-	-	27700	16.6	3.91	-	-	11089	29.6
ftv110	5.87	-	-	14393	5.0	4.49	-	-	12931	38.4
ftv120	6.51	-	-	35382	50.1	5.77	-	-	32600	99.4
ftv130	4.46	-	-	5888	6.4	3.77	-	-	3512	16.6
ftv140	4.92	-	-	11128	13.9	4.26	-	-	7083	38.8
ftv150	3.91	-	-	2645	3.1	3.03	-	-	2778	17.0
ftv160	4.58	-	-	49169	93.8	4.03	-	-	38007	316.2
ftv170	4.50	0.36	1.96	155326	1000.0	4.14	0.18	0.00	85236	1000.0
kro124p	6.22	1.20	10.95	166664	1000.0	2.73	-	-	18909	135.7
p43	97.37	95.62	0.09	186698	1000.0	0.37	0.30	0.02	321417	1000.0
rbg323	0.00	-	-	1	0.1	0.00	-	-	1	0.3
rbg358	0.00	-	-	1	0.1	0.00	-	-	1	0.5
rbg403	0.00	-	-	1	0.1	0.00	-	-	1	1.0
rbg443	0.00	-	-	1	0.1	0.00	-	-	1	1.2
ry48p	13.21	1.27	0.00	196577	1000.0	2.94	-	-	22825	20.3

Table 4.7. Comparison of branch-and-cut codes. Time limit of 10,000 seconds.

Name	FT-b&c + FP				(2-node) Concorde -s 123 -C 16					
	%Gap			Nodes	Time	%Gap			Nodes	Time
	Root	fLB	fUB			Root	fLB	fUB		
balas84	1.01	-	-	61	15.7	1.01	-	-	25	78.0
balas108	1.97	-	-	267	89.0	2.63	-	-	423	1416.0
balas120	1.05	-	-	1339	1276.3	1.05	-	-	755	7186.9
balas160	1.26	-	-	737	671.1	1.26	-	-	739	7848.0
balas200	1.24	-	-	1495	1712.8	0.74	-	-	239	2294.2
ftv180	1.20	-	-	939	366.0	0.69	-	-	29	236.2
uk66	2.29	-	-	47	5.7	1.47	-	-	17	66.1
ran1000.0	0.00	-	-	1	3.4	0.00	-	-	21	219.2
ran1000.1	0.00	-	-	3	23.5	0.00	-	-	19	191.7
ran1000.2	0.00	-	-	14	150.7	0.00	-	-	95	900.4
ran1000.3	0.01	-	-	11	62.2	0.01	-	-	343	3977.2
ran1000.4	0.01	-	-	18	148.7	0.01	-	-	379	3122.2
ran500.0	0.00	-	-	1	0.8	0.00	-	-	5	29.2
ran500.1	0.00	-	-	1	1.8	0.00	-	-	1	20.0
ran500.2	0.00	-	-	4	31.4	0.00	-	-	11	52.7
ran500.3	0.02	-	-	65	55.0	0.04	-	-	75	232.4
ran500.4	0.00	-	-	2	3.4	0.02	-	-	15	53.8
br17	0.00	-	-	1	0.0	0.00	-	-	1	0.2
ft53	0.00	-	-	1	0.1	0.00	-	-	1	0.6
ft70	0.02	-	-	5	0.2	0.01	-	-	3	3.2
ftv33	0.00	-	-	1	0.0	0.00	-	-	1	0.3
ftv35	0.88	-	-	9	0.4	0.68	-	-	5	9.0
ftv38	0.85	-	-	13	0.6	0.52	-	-	5	14.5
ftv44	0.37	-	-	9	0.5	0.12	-	-	3	9.1
ftv47	1.01	-	-	11	0.5	0.62	-	-	11	23.4
ftv55	0.81	-	-	35	1.4	0.44	-	-	3	9.0
ftv64	1.36	-	-	29	2.6	0.33	-	-	7	20.8
ftv70	0.92	-	-	11	1.1	0.26	-	-	3	17.8
ftv90	0.25	-	-	5	0.5	0.06	-	-	3	14.7
ftv100	0.39	-	-	21	2.2	0.00	-	-	1	12.6
ftv110	0.77	-	-	77	7.4	0.05	-	-	3	25.6
ftv120	0.97	-	-	123	13.1	0.28	-	-	7	54.4
ftv130	0.35	-	-	7	1.6	0.00	-	-	1	16.6
ftv140	0.25	-	-	9	2.1	0.00	-	-	3	25.6
ftv150	0.27	-	-	21	2.6	0.00	-	-	5	27.0
ftv160	0.67	-	-	17	3.8	0.30	-	-	7	55.7
ftv170	0.87	-	-	15	4.1	0.40	-	-	3	41.9
kro124p	0.04	-	-	3	1.0	0.00	-	-	1	9.9
p43	0.16	-	-	141	9.3	0.16	-	-	13	22.7
rbg323	0.00	-	-	1	0.4	0.00	-	-	3	23.9
rbg358	0.00	-	-	1	0.5	0.00	-	-	3	29.3
rbg403	0.00	-	-	1	1.3	0.00	-	-	5	49.3
rbg443	0.00	-	-	1	1.4	0.00	-	-	3	34.5
ry48p	0.53	-	-	7	0.8	0.35	-	-	5	22.9

of its *STSP* counterpart which is not captured adequately by the usual classes of *STSP* cuts (comb, clique tree inequalities, etc.).

As to the overall computing time, for the randomly generated instances (*ran*) the most effective code is CDT, which also performs very well on *shop* and *rbg* instances. Code FT-add has, on average, better performance than CDT (solving to optimality, within 1,000 CPU seconds, 8 more instances), never being, however, the best of the four codes on any instance.

Codes FT-b&c and *Concorde* turn out to be, in general, the most effective approaches, the first code being almost always faster than the second one, though it often requires more branching nodes. We believe this is mainly due to the faster (*ATSP*-specific) separation and pricing tools used in FT-b&c. Strangely enough, however, FT-b&c does not solve to optimality instance *crane316.10* which is, instead, solved by *Concorde* even in its pure *STSP* version “-C 0”; see Table 4.3. This pathological behavior of FT-b&c seems to derive from an unlucky sequence of wrong choices of the branching variable in the first levels of the branching tree.

Not surprisingly, the *Concorde* implementation proved very robust for hard instances of large size, as it has been designed and engineered to address very large *STSP* instances. On the other hand, FT-b&c was implemented by its authors to solve medium-size *ATSP* instances with up to 200 vertices, and exploits neither sophisticated primal heuristics nor optimized interfaces with the LP solver. In our view, the fact that its performance is comparable or better than that of *Concorde* “-C 16” (and considerably better than that of the pure *STSP* *Concorde* “-C 0”) is mainly due to the effectiveness of the *ATSP*-specific separation procedures used. This suggests that enriching the *Concorde* arsenal of *STSP* separation tools by means of *ATSP*-specific separation procedures would be the road to go for the solution of hard *ATSP* instances.

Acknowledgments

Work supported by Ministero dell’Istruzione, dell’Università e della Ricerca (M.I.U.R.) and by Consiglio Nazionale delle Ricerche (C.N.R.), Italy. The computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (Lab.O.R.).

Chapter 5

APPROXIMATION ALGORITHMS FOR GEOMETRIC TSP

Sanjeev Arora*

Department of Computer Science

Princeton University

Princeton, NJ 08544

arora@cs.princeton.edu

In the *Euclidean* traveling salesman problem, we are given n nodes in \mathbb{R}^2 (more generally, in \mathbb{R}^d) and desire the minimum cost salesman tour for these nodes, where the cost of the edge between nodes (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. The decision version of the problem (“Does a tour of cost $\leq C$ exist?”) is NP-hard [651, 346], but is not known to be in NP because of the use of square roots in computing the edge costs. Specifically, there is no known polynomial-time algorithm that, given integers a_1, a_2, \dots, a_n, C , can decide if $\sum_i \sqrt{a_i} \leq C$.

We are also interested in norms other than ℓ_2 , such as ℓ_p norms for any $p > 1$. All these are subcases of the more general notion of a geometric norm or *Minkowski norm*. We will refer to the version of the problem with a general geometric norm as *geometric TSP*. Since optimization is NP-hard, we will be interested in approximation. For $\alpha \geq 1$ we say that an algorithm *approximates* the problem within a factor α if it computes, for every instance I , a tour of cost at most $\alpha \cdot \text{OPT}(I)$, where $\text{OPT}(I)$ is the cost of the optimum tour in I . We emphasize that we are studying worst-case complexity in this chapter, in contrast to Chapter 7.

Geometric TSP is a subcase of metric TSP, so it inherits every algorithm for that general problem, including the *Christofides* heuristic [189], which approximates the problem within a factor 1.5 in polynomial time. For many years this was the best approximation algorithm for the problem. This changed rapidly in the last few years. Arora [37] designed an approximation scheme for geometric TSP that for every fixed $\epsilon > 0$

*Supported by a David and Lucile Packard Fellowship and NSF grant CCR-0098180.

could compute $(1 + \epsilon)$ -approximate solutions to the problem in polynomial time. The running time was initially $n^{O(1/\epsilon)}$, which later he improved to $n(\log n)^{O(1/\epsilon)}$. (A few months later, Mitchell independently discovered a similar $n^{O(1/\epsilon)}$ time approximation scheme [601].) The algorithm also applies to a host of other NP-hard geometric problems such as the *Minimum Steiner Tree*, *k-TSP* (“Given n points and a number k , find the shortest salesman tour that visits k points”), *k-MST* (“Given n points and a number k , find the shortest tree that contains k points”), etc. It also applies to TSP in \mathbb{R}^d for any constant d . If d is not constant but allowed to depend on n , the number of nodes, then the algorithm takes more than polynomial time that grows to exponential around $d = O(\log n)$. This dependence on dimension seems consistent with known complexity results, since Trevisan [794] has shown that the Euclidean TSP problem becomes MAX-SNP-hard in $O(\log n)$ dimensions, and hence the results of [658, 37] imply that there is an $\alpha > 1$ such that approximation within this factor is NP-hard.

One interesting feature of Arora’s algorithm is that it is fairly elementary, and some its ideas had already appeared in earlier algorithms such as Karp’s dissection heuristic [497] and Smith’s $2^{O(\sqrt{n})}$ time exact algorithm [766] and Blum, Chalasani and Vempala’s approximation algorithm for *k-MST* [113]. This chapter will describe the algorithm completely, and also survey a more recent algorithm of Rao and Smith [696] that has better running time. We will be concerned only with asymptotics and hence not describe the most practical version of the algorithm. The algorithms of this chapter have been implemented but they are currently not competitive with implementations of algorithms described elsewhere in this book. It would be interesting to use ideas in this chapter to improve some of those other implementations.

1. Background on Approximation

Ever since the discovery of NP-completeness, researchers have concentrated on finding near-optimal (or approximate) solutions for NP-hard problems. Sahni and Gonzalez [734] dashed such hopes for general TSP when they showed that computing good approximate solutions —of cost within even a factor 2^n of optimum— is NP-hard. This contrasts with metric TSP, for which we have Christofides’s heuristic.

The “ultimate” approximation algorithm is a *polynomial time approximation scheme* or *PTAS*, which is a sequence of polynomial-time algorithms, one for each $\epsilon > 0$, which can approximate the problem within a factor $1 + \epsilon$. Unfortunately, recent work on probabilistically checkable proofs [37] shows that if $P \neq NP$, then no PTAS exists for metric TSP.

(Actually, the same is known to be true for a large number of other problems, but that is the the topic of another story.) Trevisan has since shown that Euclidean TSP in $O(\log n)$ dimensions also has no PTAS if $P \neq NP$, as mentioned above. Thus the existence of a PTAS for Euclidean TSP in constant dimensions—the topic of the current chapter—is quite surprising². Before the discovery of this algorithm there was no indication that Euclidean TSP might be easier than metric TSP. Note that for most TSP heuristics (such as various local search heuristics) there are known Euclidean instances on which the heuristic does not approximate the optimum within a factor better than some constant (such as 1.5 or 2).

An important open question is whether we can approximate metric TSP within a factor better than 1.5? Approaches to this problem, including local search and linear programming relaxations, are discussed elsewhere in this book. There is some hope that the old Held-Karp relaxation for the TSP may help us approximate metric TSP within a factor $4/3$ (some results of Goemans [383] support this conjecture) but the best known bound on its approximation ratio is $3/2$ (Wolsey [825] and Shmoys and Williamson [761]).

2. Introduction to the Algorithm

In this chapter we only describe the algorithm for TSP in \mathbb{R}^2 with with ℓ_2 norm; the generalization to other norms is straightforward. The algorithm is reminiscent of Karp's dissection heuristic [497] (and a more recent algorithm for TSP in planar graphs due to Grigni et al. [395]) in that it uses geometric divide and conquer: it partitions the instance into smaller instances using squares. However, the algorithm differs from the dissection heuristic in two ways. First, it uses randomness to decide how to partition into smaller squares. Second, unlike the dissection heuristic, which treats the smaller squares as independent problem instances (to be solved separately and then linked together in a trivial way) this algorithm allows limited back-and-forth trips between the squares. Thus the subproblems inside adjacent squares are weakly interdependent. Specifically, the algorithm allows $O(1/\epsilon)$ entries/exits to each square of the dissection. The algorithm uses dynamic programming to find the best such tour. The correctness proof for the algorithm will show that the cost of this tour is within a factor $1 + \epsilon$ of the optimum.

²In fact, the discovery of this algorithm stemmed from the author's inability to extend the results of [37] to Euclidean TSP in constant dimensions.

Although the algorithm is described as randomized, it can be derandomized with some loss in efficiency (specifically, by trying all choices for the shifts used in the randomized dissection). Better derandomizations appear in Rao and Smith’s paper (see Section 4) and Czumaj and Lingas [237].

2.1. The Perturbation

First we perform a simple perturbation of the instance that, without greatly affecting the cost of the optimum tour, ensures that each node lies on the unit grid (i.e., has integer coordinates) and every internode distance is at least 2. Call the smallest axis-parallel square containing the nodes the *bounding box*. Our perturbation will ensure its sidelength is at most $n^2/2$.

We assume $\epsilon > 1/n^{1/3}$; a reasonable assumption since ϵ is a fixed constant independent of the input size. Let d be the maximum distance between any two nodes in the input. Then $2d \leq \text{OPT} \leq nd$. We lay down a grid in which the grid lines are separated by distance $\epsilon d/n^{1.5}$. Then we move each node to its nearest gridpoint. This may merge some nodes; we treat this merged node as a single node in the algorithm. (At the end we will extend the tour to these nodes in a trivial way by making excursions from the single representative.) Note that the perturbation moves each node by at most $\epsilon d/n^{1.5}$, so it affects the optimum tour cost by at most $\epsilon d/n^{0.5}$, which is negligible compared to ϵOPT when n is large. Finally, we rescale distances so that the minimum internode distance is at least 2. Then the maximum internode distance is at most $n^{1.5}/\epsilon$, which is asymptotically less than $n^2/2$, as desired.

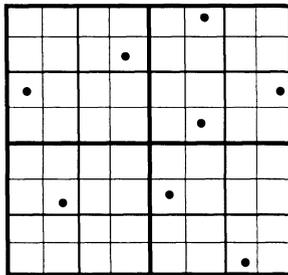


Figure 5.1. The dissection

2.2. Random Partition

A *dissection* of a square is a recursive partitioning into squares; see Figure 5.1. We view this partitioning as a tree of squares whose root

is the square we started with. Each square in the tree is partitioned into four equal squares, which are its children. The leaves are squares of sidelength 1.

A *randomized dissection* of the instance is defined as follows. Let $P \in \mathbb{R}^2$ be the lower left endpoint of the bounding box and let each side have length l . We enclose the bounding box inside a larger square — called the *enclosing box*—of sidelength $L = 2l$ and position the enclosing box such that P has distance a from the left edge and b from the lower edge, where integers $a, b \leq l$ are chosen randomly. We refer to a, b as the *horizontal* and *vertical shift* respectively; see Figure 5.2. The randomized dissection is defined to be the dissection of this enclosing box. Note that we are thinking of the input nodes and the unit grid as being fixed; the randomness is used only to determine the placement of the enclosing box (and its accompanying dissection).

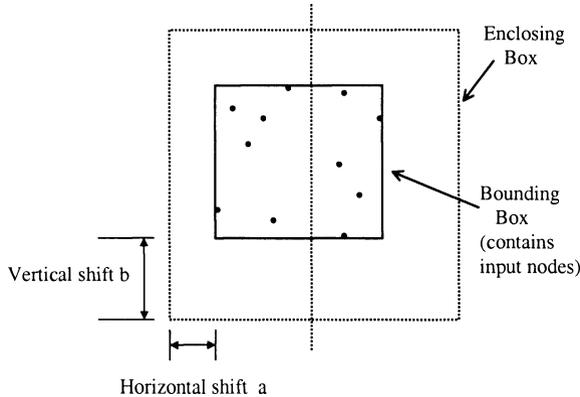


Figure 5.2. The enclosing box contains the bounding box and is twice as large, but is shifted by random amounts in the x and y directions.

We assume without loss of generality that L is a power of 2 so the squares in the dissection have integer endpoints and leaf squares have sides of length 1 (and hence at most one node in them). The *level* of a square in the dissection is its depth from the root; the root square has level 0. We also assign a *level* from 0 to $\log L - 1$ to each horizontal and vertical grid line that participated in the dissection. The horizontal (resp., vertical) line that divides the enclosing box into two has level 0. Similarly, the 2^i horizontal and 2^i vertical lines that divide the level i squares into level $i + 1$ squares each have level i .

Now we mention the property of a random dissection that will be crucial in the proof (see for example the proof of Lemma 1). Consider any fixed vertical grid line that intersects the bounding box of the instance. What is the chance that it becomes a level i line in the randomized

dissection? There are 2^i values of the horizontal shift a (see Figure 5.2 again) that cause this to happen, so

$$\Pr_a[\text{this line is at level } i] = \frac{2^i}{l} = \frac{2^{i+1}}{L} \quad (1)$$

Thus the randomized dissection treats—in an expected sense—all grid lines symmetrically.

2.3. Portal-Respecting Tours

Each grid line will have special points on it called *portals*. A level i line has $2^{i+1}m$ equally spaced portals, where m is the *portal parameter* (to be specified later). We require m to be a power of 2. In addition, we also refer to the corners of each a square as a portal. Since the level i line has 2^{i+1} level $i + 1$ squares touching it, we conclude that each side of the square has at most $m + 2$ portals (m usual portals, and the 2 corners), and a total of at most $4m + 4$ portals on its boundary. A *portal-respecting* tour is one that, whenever it crosses a grid line, does so at a portal. Of course, such a tour will not be optimal in general, since it may have to deviate from the straight-line path between nodes.

The optimum portal-respecting tour does not need to visit any portal more than twice; this follows by the standard observation that removing repeated visits can, thanks to triangle inequality, never increase the cost. Thus the optimum portal-respecting tour enters and leaves each dissection square at most $8m$ times. A simple dynamic programming can find the optimum portal respecting tour in time $2^{O(m)}L \log L$. Allowing $m = O(\log n/\epsilon)$ suffices (see Section 3), hence the running time is $n^{O(1/\epsilon)}$.

A more careful analysis in Section 4 shows that actually we only need to consider portal respecting tours that enter/exit each dissection square at most $4k = O(1/\epsilon)$ times. The dynamic programming described below can find the best such tour in $\text{poly}\left(\binom{m}{k}\right)2^{O(k)}L \log L$ time, which is $O(n(\log n)^{O(1/\epsilon)})$.

Dynamic programming. We sketch the simple dynamic programming referred to above. To allow a cleaner analysis later, the randomized dissection was described above as a regular 4-ary tree in which each leaf has the same depth. In an actual implementation, however, one can truncate this tree so that the partitioning stops as soon as a square has at most 1 input node in it. Then the dissection has at most $2n$ leaves and hence $O(n \log n)$ squares. Furthermore, the cost of the optimum portal respecting tour cannot increase since truncating the dissection can only reduce the number of times the tour has to make detours to pass through portals.

The truncated dissection (which is just the quadtree of the enclosing box) can be efficiently computed, for instance by sorting the nodes by x - and y -coordinates (for better algorithms, especially in higher dimensions, see Bern et al. [106]). The dynamic programming now is the obvious one. Suppose we are interested in portal-respecting tours that enter/exit each dissection square at most $4k$ times. The subproblem inside the square can be solved independently of the subproblem outside the square so long as we know the portals used by the tour to enter/exit the square, and the *order* in which the tour uses these portals. Note that given this *interface* information, the subproblems inside and outside the square involve finding not salesman tours but a set of up to $4k$ vertex-disjoint paths that visit all the nodes and visit portals in a way consistent with the interface. (See Figure 5.3.) We maintain a lookup table that, for each square and for each choice of the interface, stores the optimum way to solve the subproblem inside the square. The lookup table is filled up in a bottom-up fashion in the obvious way. Clearly, its size is $(\# \text{ of dissection squares}) \times m^{O(k)} k!$.

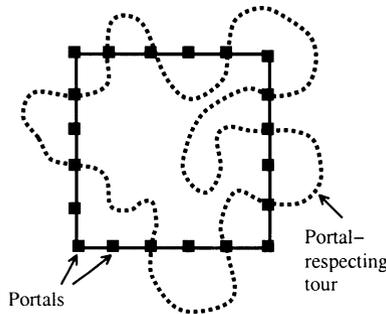


Figure 5.3. This portal-respecting tour enters and leaves the square 10 times, and the portion inside the square is a union of 5 disjoint paths.

As noted, k , the number of times the portal-respecting tour can enter/exit the square, is at most $8m = O(\log n/\epsilon)$. Then the term $m^{O(k)}$ in the running time is somewhat larger than any polynomial of n . However, one can change this term to $2^{O(m)} = n^{O(1/\epsilon)}$ by noticing that the dynamic programming need not consider all possible interfaces for a square. The optimum portal respecting tour in the plane does not cross itself, and thus it only needs to consider the corresponding interfaces. These correspond to well-matched parenthesis pairs, and the number of possibilities for these are given by the well-known *Catalan* numbers. We omit details since the better analysis given in Section 4 shows that $k = O(1/\epsilon)$, which greatly reduces the running time.

3. Simpler Algorithm

First we give a very simple analysis showing that if the portal parameter m is $O(\log n/\epsilon)$, then the best portal-respecting tour is likely to be near optimal. This tour may enter/leave each square $8m$ times.

The following simple lemma lies at the heart of this analysis. For any two nodes $u, v \in \mathbb{R}^2$ let the *portal-respecting distance* between u and v be the shortest distance between them when all intermediate grid lines have to be crossed at portals. We denote it by $d_{a,b}(u, v)$, where our notation stresses the dependence of this number upon shifts a and b . Clearly, $d_{a,b}(u, v) \geq d(u, v)$.

Lemma 1 *When shifts a, b are random, the expectation of $d_{a,b}(u, v) - d(u, v)$ is at most $\frac{2 \log L}{m} d(u, v)$, where L is the sidelength of the root square.*

Proof: The expectation, though difficult to calculate exactly, is easy to upperbound. The straight line path from u to v crosses the unit grid at most $2d(u, v)$ times. To get a portal-respecting path, we move each crossing to the nearest portal on that grid line (see Figure 5.4), which involves a detour whose length is at most the interportal distance. If the line in question has level i , the interportal distance is $L/m2^{i+1}$. By (1), the probability that the line is at level i is $2^{i+1}/L$. Hence the expected length of the detour is at most

$$\sum_{i=0}^{\log L-1} \frac{2^{i+1}}{L} \cdot \frac{L}{m2^{i+1}} = \frac{\log L}{m}.$$

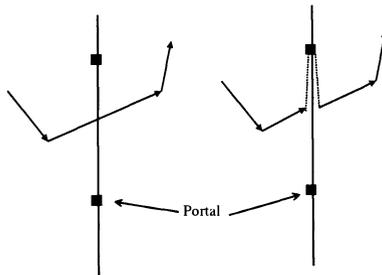


Figure 5.4. Every crossing is moved to the nearest portal by adding a “detour.”

The same upperbound applies to each of the $2d(u, v)$ crossings, so linearity of expectations implies that the expected increase in moving all crossings to portals is at most $2d(u, v) \log L/m$. This proves the lemma. ■

Let OPT denote the cost of the optimum salesman tour and $\text{OPT}_{m,a,b}$ denote the cost of the best portal-respecting tour when the portal parameter is m and the random shifts are a, b .

Corollary 2 *The expectation (over the choice of a, b) of $\text{OPT}_{m,a,b} - \text{OPT}$ is at most $2 \log L/m \text{OPT}$, where OPT is the cost of the best salesman tour.*

Proof: We make the optimum tour portal-respecting and upperbound the increase in cost. The upperbound from Lemma 1 applies to each edge in the optimum tour, so linearity of expectations implies that the expected cost increase is at most $\text{OPT} \cdot 2 \log L/m$. ■

With probability at least $1/2$ (over the choice of shifts a, b) the difference $\text{OPT}_{m,a,b} - \text{OPT}$ is at most twice its expectation, namely $4 \log L/m \cdot \text{OPT}$. When the root square has sides of length $L \leq n^2$ (as ensured by our perturbation) and m is at least $4 \log n/\epsilon$, this difference is at most $4 \log n/m \cdot \text{OPT} = \epsilon \cdot \text{OPT}$. Thus $\text{OPT}_{m,a,b} \leq (1 + \epsilon)\text{OPT}$ with probability at least $1/2$.

4. Better Algorithm

A portal-respecting tour is *k-light* if it crosses each side of each dissection square at most k times. We will show that if $k = 24/\epsilon + 4$ then the probability is at least $1/2$ (over the choice of the random shifts) that the best k -light tour has cost at most $(1 + \epsilon)\text{OPT}$. The analysis in this proof has a global nature, by which we mean that it takes all the edges of the tour into account simultaneously (see our charging argument below). By contrast, many past analyses of approximation algorithms for geometric problems reason in an edge-by-edge fashion; see for instance the algorithms surveyed in [105] and also our simpler analysis in Section 3.

We will use the following well-known fact about Euclidean TSP that is implicit in the analysis of the dissection heuristic, and is made explicit in [35].

Lemma 3 (Patching Lemma) *Let S be any line segment of length s and π be a closed path that crosses S at least thrice. Then we can break the path in all but two of these places, and add to it line segments lying on S of total length at most $3s$ such that π changes into a closed path π' that crosses S at most twice.*

Proof: For simplicity we give a proof using segments of length $6s$ instead of $3s$; the proof of $3s$ uses the Christofides heuristic and the reader may wish to work it out.

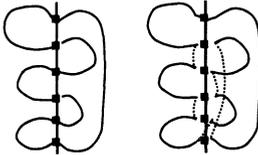


Figure 5.5. The tour crossed this line segment 6 times, but breaking it and reconnecting on each side (also called “patching”) reduced the number of crossings to 2.

Suppose π crosses S a total of t times. Let M_1, \dots, M_t be the points on which π crosses S . Break π at those points, thus causing it to fall apart into t paths P_1, P_2, \dots, P_t . In what follows, we will need two copies of each M_i , one for each side of S . Let M'_i and M''_i denote these copies.

Let $2j$ be the largest even number less than t . Let J be the multiset of line segments consisting of the following: (i) A minimum cost salesman tour through M_1, \dots, M_t . (ii) A minimum cost perfect matching among M_1, \dots, M_{2j} . Note that the line segments of J lie on S and their total length is at most $3s$. We take two copies J' and J'' of J and add them to π . We think of J' as lying on the left of S and J'' as lying on the right of S .

Now if $t = 2j + 1$ (i.e., t is odd) then we add an edge between M'_{2j+1} and M''_{2j+1} . If $t = 2j + 2$ then we add an edge between M'_{2j+1} and M''_{2j+1} and an edge between M'_{2j+2} and M''_{2j+2} . (Note that these edges have length 0.)

Together with the paths P_1, \dots, P_{2j} , these added segments and edges define a connected 4-regular graph on $\{M'_1, \dots, M'_t\} \cup \{M''_1, \dots, M''_t\}$. An Eulerian traversal of this graph is a closed path that contains P_1, \dots, P_t and crosses S at most twice. (See Figure 4.) Hence we have proved the theorem. ■

Another needed fact —implicit also in Lemma 1—relates the cost of a tour to the total number of times it crosses the lines in the unit grid. If l is one of these lines and π is a salesman tour then let $t(\pi, l)$ denote the number of times π crosses l . Then

$$\sum_{l:\text{vertical}} t(\pi, l) + \sum_{l:\text{horizontal}} t(\pi, l) \leq 2 \text{cost}(\pi) \tag{2}$$

Now we are ready to prove the main result of the section. Let $k \geq 24/\epsilon + 5$. The main idea is to transform an optimum tour π into a k -light tour. Whenever the tour enters/exits a square “too many” times, we use the Patching Lemma to reduce the number of crossings. This increases cost, but we will upperbound the expected cost increase by $\epsilon \cdot \text{OPT}/2$ using the relationship in (2). Then we will have shown that with probability at least $1/2$, the best k -light tour has cost at most $(1 + \epsilon)\text{OPT}$.

Let us describe this tour transformation process for a single vertical grid line, say l . (A similar transformation happens at every grid line.) Suppose l has level i . It is touched by 2^{i+1} level $i + 1$ squares, which partition it into 2^{i+1} segments of length $L/2^{i+1}$. For each $j > i$, line l is also touched by 2^j level j squares. In general, we will refer to the portion of l that lies in a level j square as a *level j segment*. The final goal is to reduce the number of crossings in each level i segment to k or less; we do this as follows.

Let $s = k - 4$. An *overloaded* segment of l is one which the tour crosses at least $s + 1$ times. The tour transformation proceeds as follows. For every level $\log L - 1$ segment that is overloaded, we apply the patching lemma and reduce the number of crossings to 2. In the transformed tour, we now look at segments of level $\log L - 2$ and for each overloaded segment, apply the patching lemma to reduce the number of crossings to 2. Continuing this way for progressively higher levels, we stop when no segment at level i is overloaded. At the end, we move all crossings to portals; we do this by adding vertical detours (i.e., vertical segments).

To analyze the cost increase in this transformation, we consider an imaginary procedure in which the tour transformation on this vertical grid line l proceeds to level 0, i.e., until the entire line is not overloaded. Let $X_{l,j}(b)$ be a random variable denoting the number of overloaded level j segments encountered in this imaginary procedure. We draw attention to the fact that $X_{l,j}(b)$ is determined by the vertical shift b alone, which determines the location of the tour crossings with respect to the segments on l . We claim that for every b ,

$$\sum_{j \geq 0} X_{l,j}(b) \leq \frac{t(\pi, l)}{s - 1}. \quad (3)$$

The reason is that the optimum tour π crossed grid line l only $t(\pi, l)$ times, and each application of the Patching Lemma counted on the left hand side of (3) replaces at least $s + 1$ crossings by at most 2, thus eliminating $s - 1$ crossings each time.

Since a level j segment has length $L/2^j$, the cost of the imaginary transformation procedure is, by the Patching Lemma, at most

$$\sum_{j \geq 1} X_{l,j}(b) \cdot \frac{3L}{2^j}. \tag{4}$$

(Note that in (4) we omit the case $j = 0$ because the level 0 square is just the bounding box, and the tour lies entirely inside it.)

The *actual* cost increase in the tour transformation at l depends on the level of l , which is determined by the horizontal shift a . When the level is i , the terms of (4) corresponding to $j \geq i + 1$ upperbound the cost increase:

$$\text{Increase in tour cost when } l \text{ has level } i \leq \sum_{j \geq i+1} X_{l,j}(b) \cdot \frac{3L}{2^j}, \tag{5}$$

We “charge” this cost to l . Of course, whether or not this charge occurs depends on whether or not i is the level of line l , which by (1) happens with probability at most $2^{i+1}/L$ (over the choice of the horizontal shift a). Thus for every vertical shift b

$$\begin{aligned} & E_a[\text{charge to } l \text{ when horizontal shift is } a] \\ &= \sum_{i \geq 1} \frac{2^{i+1}}{L} \cdot (\text{charge to } l \text{ when its level is } i) \\ &\leq \sum_{i \geq 1} \frac{2^{i+1}}{L} \cdot \sum_{j \geq i+1} X_{l,j}(b) \cdot \frac{3L}{2^j} \\ &= 3 \cdot \sum_{j \geq 1} \frac{X_{l,j}(b)}{2^j} \cdot \sum_{i \leq j-1} 2^i \\ &= 3 \cdot \sum_{j \geq 1} \frac{X_{l,j}(b)}{2^j} \cdot (2^j - 1) \\ &\leq 3 \cdot \sum_{j \geq 1} 2 \cdot X_{l,j}(b) \\ &\leq \frac{6 t(\pi, l)}{s - 1} \end{aligned}$$

We may now appear to be done, since linearity of expectations seems to imply that the total cost charged to all lines is

$$\begin{aligned} & \sum_l \frac{6t(\pi, l)}{s-1}, \\ \text{which from (2) is at most} & \leq 12 \frac{\text{cost}(\pi)}{s-1}. \end{aligned}$$

which is at most $\epsilon\text{OPT}/2$ since $s \geq 24/\epsilon + 1$.

However, we are not done. We have to worry about the issue of how the modifications at various grid lines affect each other. Fixing overloaded segments on vertical grid lines involves adding vertical segments to the tour, thus increasing the number of times the tour crosses horizontal grid lines; see Figure 4. Then we fix the overloaded segments on horizontal grid lines. This adds horizontal segments to the tour, which may potentially lead to some vertical lines becoming overloaded again. We need to argue that the process stops. To this end we make a simple observation: fixing the overloaded segments of the vertical grid line l adds at most 2 additional crossings on any horizontal line l' . The reason is that if the increase were more than 2, we could just use the Patching Lemma to reduce it to 2 and this would not increase cost since the Patching Lemma is being invoked for segments lying on l which have zero horizontal separation (that is, they lie on top of each other). Also, to make sure that we do not introduce new crossings on l itself we apply the patching separately on vertical segments of both sides of l . Arguing similarly about all intersecting pairs of grid lines, we can ensure that at the end of all our modifications, the tour crosses each side of each dissection square up to $s + 4$ times; up to s times through the side and up to 4 times through the two corners. Since $s + 4 = k$, we have shown that the tour is k -light.

The analysis of the cost incurred in moving all crossings to the nearest portal is similar to the one in Section 3. This completes our proof.

5. Faster Algorithm

Rao and Smith [696] describe an improvement of the above algorithm that runs in time $O(n \log n + n/\text{poly}(\epsilon))$. First they point out why the running time of the above algorithm is $n(\log n)^{O(1/\epsilon)}$. It is actually $nm^{O(k)}$ where m is the portal parameter and k is the number of times the tour can cross each side of each dissection square. The n comes from the number of squares in the dissection, and $m^{O(k)}$ from the fact that the dynamic programming has to enumerate all possible “interfaces,” which involves enumerating all ways of choosing k crossing points from among the $O(m)$ portals. The analysis given above seems to require m to be $\Omega(\log n)$ and k to be $\Omega(1/\epsilon)$, which makes the running time $\Omega(n(\log n)^{O(1/\epsilon)})$. Their main new idea is to reduce the $m^{O(k)}$ enumeration time by giving the dynamic programming more hints about which portals are used by the tour to enter/exit each dissection square.

The “hints” mentioned above are generated using a $(1 + \epsilon)$ -spanner of the input nodes. This is a connected graph with $O(n/\text{poly}(\epsilon))$ edges

in which the distance between any pair of nodes is within a factor $(1 + \epsilon)$ of the Euclidean distance. Such a spanner can be computed in $O(n \log n / \text{poly}(\epsilon))$ time (see Althoefer et al. [21]). Note that distances in the spanner define a metric space in which the optimum TSP cost is within a factor $(1 + \epsilon)$ of the optimum in the Euclidean space.

Rao and Smith notice that the tour transformation procedure of Section 4 can be applied to any connected graph, and in particular, to the spanner. The transformed graph is portal-respecting, as well as k -light for $k = O(1/\epsilon)$. The *expected* distance between any pair of points in the transformed graph is at most factor $(1 + \epsilon)$ more than what it was in the spanner. Note that some choices of the random shifts may stretch the distance by a much larger factor; the claim is only about the expectation. (In particular, it is quite possible that the transformed graph is not a spanner.) By linearity of expectation, the expected increase in the cost of the optimum tour in the spanner is also at most a factor $(1 + \epsilon)$.

Now the algorithm tries to find the optimum salesman tour with respect to distances in this transformed graph. Since the transformed graph is k -light, we know for each dissection square a set of at most $4k$ portals that are used by the tour to enter/exit the square. As usual, we can argue that no portal is crossed more than twice. Thus the dynamic programming only needs to consider $k^{O(k)}$ “interfaces” for each dissection square. For more details see Rao and Smith’s paper.

6. Generalizations to other Problems

Although this chapter is solely concerned with the TSP, the reader may also be interested in other geometric NP-hard problems. In the *Minimum Steiner Tree* problem, we are given n nodes in \mathbb{R}^d and desire the minimum-cost tree connecting them³. In general, the minimum spanning tree is not an optimal solution. In \mathbb{R}^2 (with distances measured in ℓ_2 norm) the cost of the MST can be as far as a factor $2/\sqrt{3}$ from the optimum. A spate of research activity in recent years (starting with the work of Zelikovsky[836]) has provided better algorithms, with an approximation ratio around 1.143. The metric case does not have an approximation scheme if $P \neq NP$ [107].

The algorithms from the previous sections generalize to this problem easily. They use very few properties of the TSP; only the Patching Lemma and the relationship in (2). Both are true for the Steiner tree problem.

³It appears that this problem was first posed by Gauss in a letter to Schumacher [393].

The algorithms also generalize to k -TSP, k -MST [35] the k -median problem [38], Euclidean min-cost k -connected subgraph [237], and the minimum latency problem [36] In some cases the algorithms are not as efficient as the TSP algorithms.

6.1. Higher-Dimensional Versions

Although we concentrated on the version of the Euclidean TSP, higher-dimensional Euclidean TSP in \mathbb{R}^2 , the algorithms also generalize to higher-dimensional versions of all these problems. The analysis is similar, with the obvious changes (such as replacing squares by higher dimensional cubes). For more details see Arora [35].

Note that the running time - even after using the ideas of Rao and Smith - has a doubly exponential dependence upon the dimension d . So the dimension should be $o(\log \log n)$ in order for the running time to be polynomial. We have reason to believe that this dependence is inherent, since Trevisan [794] has shown that there is an $\epsilon > 0$ such that $(1 + \epsilon)$ -approximation to Euclidean TSP in n dimensions is MAX-SNP hard. Using a general dimension-reduction due to Johnson and Lindenstrauss [468], it follows that if a polynomial-time approximation scheme exists even in $O(\log n)$ dimensions, then $P=NP$. A survey of these developments is currently under preparation and will soon be available from the author.

Chapter 6

EXPONENTIAL NEIGHBORHOODS AND DOMINATION ANALYSIS FOR THE TSP

Gregory Gutin

*Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
G.Gutin@rhul.ac.uk*

Anders Yeo

*Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
anders@cs.rhul.ac.uk*

Alexei Zverovitch

*Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
A.Zverovitch@rhul.ac.uk*

1. Introduction, Terminology and Notation

In this section, we overview the main results on the topics of this chapter and give basic terminology and notation used throughout the chapter.

1.1. Introduction

The purpose of this chapter is to introduce the reader to recently developed concepts and results on exponential (size) neighborhoods and domination analysis for the traveling salesman problem (TSP). Even though these topics are of certain practical relevance, we restrict our-

selves to the theoretical study. The body of computational experiments with exponential neighborhoods is insufficient yet to carry out meaningful comparisons between new and classical approaches; we refer the reader to the papers [80, 162] and Chapter 9, where certain computational experience with exponential neighborhoods is reported. The reader may consult Chapters 9 and 10 of this book for discussion of the experimental performance of some heuristics studied in the domination analysis part of this chapter.

It is worth noting that while the symmetric traveling salesman problem (STSP) can be considered, in many cases, as a subproblem of the asymmetric traveling salesman problem (ATSP), sometimes this view is too simplistic since the ATSP and STSP are defined on different graphs – complete directed and undirected. Thus, in particular, the number of tours in ATSP and STSP on n vertices is $(n - 1)!$ and $(n - 1)!/2$, respectively. Therefore, while we will mostly consider the ATSP in this chapter, we will provide a separate treatment of the STSP when needed. We will use the term TSP when it is not important whether the ATSP or STSP is under consideration.

Local search heuristics are among the main tools to compute near optimal tours in large instances of the TSP in relatively short time, see, e.g., Chapters 8, 9 and 10 of this book. In most cases the neighborhoods used in the local search algorithms are of polynomial cardinality. One may ask whether it is possible to have exponential size neighborhoods for the TSP such that the best tour in such a neighborhood can be computed in polynomial time. Fortunately, the answer to this question is positive. (This question is far from being trivial for some generalizations of the TSP, e.g. Deineko and Woeginger [248] conjecture that for the quadratic assignment problem there is no exponential neighborhood “searchable” in polynomial time.)

There are only a few papers on exponential neighborhoods published before the 1990s: Klyaus [507], Sarvanov and Doroshko [743, 744] and Gutin [414, 415]. In particular, [744] and [414] independently showed the existence of $(n/2)!$ -size neighborhood for the TSP with n vertices (n is even). In this neighborhood, the best tour can be computed in $O(n^3)$ time, i.e., asymptotically in at most the same time as a complete iteration of 3-Opt, which finds the best tour among only $\Theta(n^3)$ tours.

Punnen [680] showed how to generalize the neighborhood from [414, 744] and Gutin [416] proved that one of Punnen’s extensions provides neighborhoods of size $\Theta(\exp(\sqrt{n/2})(n/2)!/n^{1/4})$. We study basic results on exponential neighborhoods in Section 2. We use the definition of a neighborhood from [248], the only survey paper on the topic. Notice that while Deineko and Woeginger [248], in their own words, “only

scratched the surface” in their survey on the topic, we provide more detailed treatment of some exponential neighborhoods.

In Section 3, following Gutin and Yeo [421], we provide upper bounds on the size of ATSP neighborhood. In particular, we prove that there is no ATSP neighborhood of cardinality at least $\beta(n-k)!$ for any constant $\beta > 0$ and fixed integer k provided $\text{NP} \not\subseteq \text{P/poly}$. (We provide an informal description of the class P/poly in Section 3.2; for a formal introduction of the topic, see [82].)

While it is natural to study the possible cardinality of neighborhoods, it is clear that the size of a neighborhood is not the only parameter of importance. Indeed, the neighborhood introduced in [414, 744] does not perform well in computational practice. This may be a result of an unfortunate property of the neighborhood: many tours of the TSP are not reachable from each other under the structure imposed by this neighborhood. Carlier and Villon [162] showed that their neighborhood is much better in this respect: each tour can be reached from any other tour in at most logarithmic number (in n) of iterations if the choice of a tour at every iteration is “right”. Gutin and Yeo [418] introduced a neighborhood structure, which makes the tours much closer: for every pair of tours T_1, T_5 there are three tours T_2, T_3, T_4 such that every T_i is in the neighborhood of T_{i-1} , $i = 2, 3, 4, 5$. (The neighborhoods in [418] are polynomially searchable.) We study the “closeness” topic in Section 4.

Chapters 9 and 10 consider experimental performance of TSP heuristics. While experimental analysis is of certain importance, it cannot cover all possible families of TSP instances and, in particular, it normally does not cover the most hard ones. Experimental analysis provides little theoretical explanation why certain heuristics are successful while some others are not. This limits our ability to improve on the quality and efficiency of existing algorithms. It also limits our ability to extend approaches successful for the TSP to other combinatorial optimization (CO) problems.

Approximation analysis is a frequently used tool for theoretical evaluation of CO heuristics. Let \mathcal{H} be a heuristic for the TSP, and let \mathcal{I}_n be the set of instances of the TSP of size n . In approximation analysis, we use the approximation ratio $r_{\mathcal{H}}(n) = \max\{f(I)/f^*(I) : I \in \mathcal{I}_n\}$, where $f(I)$ ($f^*(I)$) is the cost of the heuristic (optimal) tour. Unfortunately, in most of cases, estimates for $r_{\mathcal{H}}(n)$ are not constants and provide only a vague picture of quality of heuristics.

Domination analysis provides an alternative to approximation analysis. In domination analysis, we are interested in the number of feasible solutions that are worse or equal in quality to the heuristic one, which is

called the domination number of the heuristic solution. In many cases, domination analysis is very useful. In particular, some heuristics have domination number 1 for the TSP. In other words, those heuristics, in the worst case, produce the unique worst possible solution. At the same time, the approximation ratio is not bounded by any constant. In this case, domination number provides a far better insight into the performance of the heuristics.

Results on domination number of TSP heuristics are considered in Section 5. Domination number was formally introduced in a 1996 version of [680] and [382]¹. Interestingly, the first important results on domination number can be traced back to the 1970s, see Rublineckii [733] and Sarvanov [738]. The *domination number* $\text{domn}(\mathcal{H}, \mathcal{I})$ of a TSP heuristic \mathcal{H} for a particular instance \mathcal{I} of the TSP with n vertices is the number of tours in \mathcal{I} which are at least as costly as the tour found by \mathcal{H} . The *domination number* $\text{domn}(\mathcal{H}, n)$ of \mathcal{H} is the minimum of $\text{domn}(\mathcal{H}, \mathcal{I})$ over all instances \mathcal{I} with n vertices. Since the ATSP on n vertices has $(n - 1)!$ tours, an algorithm for the ATSP with domination number $(n - 1)!$ is exact. The domination number of an exact algorithm for the STSP is $(n - 1)!/2$. Similarly, one can define the domination number of heuristics for other CO problems.

Glover and Punnen [382] asked whether there exists a polynomial time STSP heuristic with domination number at least $(n - 1)!/p(n)$, where $p(n)$ is a polynomial in n , provided $P \neq NP$. Answering this question, Gutin and Yeo [423] introduced polynomial time heuristics for the ATSP with domination number at least $(n - 2)!$. Two years after [423] was completed, we found out that Rublineckii [733] and Sarvanov [739] answered the above question already in the the 1970s by showing that certain polynomial time heuristics for the STSP and the ATSP are of domination number at least $(n - 2)!$ when n is odd and $(n - 2)!/2$ when n is even. Punnen, Margot and Kabadi [684] proved that the best improvement² versions of some well-known local search heuristics for the TSP after polynomial number of steps produce tours which are not worse than at least $\Omega((n - 2)!)$ other tours. Punnen and Kabadi [683] obtained an $O(n^2)$ time heuristic with domination number at least $\sum_{k=1}^{n-2} (k!)$. Gutin and Yeo [420] investigated the existence of polynomial time heuristics with domination number $\Theta((n - 1)!)$.

¹Actually, an equivalent concept of domination ratio was introduced, which is the ratio of the domination number and the number of tours.

²During every iteration, best improvement local search algorithms compute the best tour in the current neighborhood.

Some heuristics may have a small domination number (thus, indicating that they are not useful, in general). For example, the “anti-greedy” heuristic for the ATSP that starts by choosing an arc of maximum cost and proceeds by choosing the most expensive arc among remaining eligible ones, is of domination number 1 (consider an instance with $c(i, i+1) = 1$ for every $i = 1, 2, \dots, n-1$, $c(n, 1) = 1$, and $c(i, j) = 0$ for every $j \neq i+1$ and $(i, j) \neq (n, 1)$). While the fact that the domination number of the anti-greedy heuristic equals one is quite expected, in Section 5, we prove that the same is true for the greedy and nearest neighbor algorithms for both the ATSP and STSP (these results were obtained by Gutin, Yeo and Zverovich in [424]). Punnen, Margot and Kabadi [684] proved that some other TSP algorithms are of very small domination number. In particular, they showed that the double tree heuristic and some variations of the Christofides heuristic for the STSP are of domination number 1.

In this chapter we discuss approaches and results obtained mostly in the last decade. Despite limited time and effort in the areas of domination analysis and exponential neighborhoods, one can clearly see that significant progress has already been made. Although some of the existing results and approaches have already been used in practice (see Chapter 9 and [80, 162, 377]), it seems that much more research is required before the above mentioned areas can be used to design new high quality heuristics for the TSP and other CO problems. We hope that this chapter will provide motivation for scholars and practitioners to continue studying the domination analysis and exponential neighborhoods for the TSP and other CO problems.

1.2. Basic Terminology and Notation

Recall that the ATSP is stated as follows. Given a weighted complete digraph (\vec{K}_n, c) , find a Hamiltonian cycle in \vec{K}_n of minimum cost. Here the cost function c is a mapping from $A(\vec{K}_n)$ to the set of reals. The cost of an arc (x, y) of \vec{K}_n is $c(x, y)$. It is assumed that $V(\vec{K}_n) = \{1, 2, \dots, n\}$. The mapping c can be determined by the cost matrix $[c_{ij}]$. The STSP is defined similarly with the only difference that the graph under consideration is complete undirected (denoted by K_n). In this case, the matrix $[c_{ij}]$ is symmetric. Unless it is specified otherwise, n is the number of vertices in the instance of the TSP under consideration.

Let $C = x_1x_2\dots x_kx_1$ be a cycle in \vec{K}_n . The operation of *removal* of a vertex x_i ($1 \leq i \leq k$) results in the cycle $x_1x_2\dots x_{i-1}x_{i+1}\dots x_kx_1$ (thus, removal of x_i is not deletion of x_i from C ; *deletion* of x_i gives

the path $x_{i+1}x_{i+2} \dots x_kx_1x_2 \dots x_{i-1}$). Let y be a vertex of \vec{K}_n not in C . The operation of *insertion* of y into an arc (x_i, x_{i+1}) results in the cycle $x_1x_2 \dots x_ix_{i+1} \dots x_kx_1$. The *cost* of the insertion is defined as $c(x_i, y) + c(y, x_{i+1}) - c(x_i, x_{i+1})$. For a set $Z = \{z_1, \dots, z_s\}$ ($s \leq k$) of vertices not in C , an *insertion* of Z into C results in the tour obtained by inserting the nodes of Z into different arcs of the cycle. In particular, insertion of y into C involves insertion of y into one of the arcs of C .

For a path $P = x_1x_2 \dots x_m$ in (\vec{K}_n, c) , the *contraction*³ of P in (\vec{K}_n, c) , $(\vec{K}_n / P, c')$, is a complete digraph with vertex set

$$V(\vec{K}_n / P) = V(\vec{K}_n) \cup \{v_P\} - V(P),$$

where $v_P \notin V(\vec{K}_n)$, such that the cost $c'(u, w)$, for $u, w \in V(\vec{K}_n / P)$, is defined by $c(u, x_1)$ if $w = v_P$, $c(x_m, w)$ if $u = v_P$, and $c(u, w)$, otherwise. We can consider an arc $a = (x, y)$ as the path xy of length one; this allows us to look at \vec{K}_n / a as a special case of the above definition. The above definition has an obvious extension to a set of vertex-disjoint paths.

Further definitions on directed and undirected graphs can be found in the corresponding appendix of this book; see also [84].

2. Exponential Neighborhoods

We adapt the definition of a neighborhood for the TSP due to Deineko and Woeginger [248]. Let P be a set of permutations on n vertices. Then the *neighborhood* (with respect to P) of a tour $T = x_1x_2 \dots x_nx_1$ is defined as follows:

$$N_P(T) = \{x_{\pi(1)}x_{\pi(2)} \dots x_{\pi(n)}x_{\pi(1)} : \pi \in P\}.$$

A *neighborhood structure* consists of neighborhoods for every tour T . The above definition of a neighborhood is quite restrictive⁴ but reflects the very important “shifting” property of neighborhoods which distinguishes them from arbitrary sets of tours. Another important property usually imposed on a neighborhood $N(T)$ of a tour T is that the best among tours of $N(T)$ can be computed in time $p(n)$ polynomial in n . This is necessary to guarantee an efficient local search. Neighborhoods

³The operation is called path-contraction in [84], but since we do not consider any other type of contraction, we will use the shorter name.

⁴In particular, this definition implies that the neighborhood of every tour is of the same cardinality.

satisfying this property are called *polynomially searchable* or, more precisely, *p(n)-searchable*.

In the rest of this section and in Section 4, we only consider the ATSP: the neighborhoods we describe below can be readily adapted to the STSP.

2.1. The Pyramidal Neighborhood

In this subsection, we consider the pyramidal neighborhood introduced by Sarvanov and Doroshko [743]. Let $H = x_1x_2 \dots x_nx_1$ be a tour. Define the *pyramidal neighborhood* of H , denoted by $PY(x_1, H)$, as follows.

A tour $G = x_{i_1}x_{i_2}x_{i_3} \dots x_{i_n}x_{i_1}$, with $i_1 = 1$, belongs to $PY(x_1, H)$, if and only if there is an integer k , such that

$$i_1 < i_2 < \dots < i_k > i_{k+1} > i_{k+2} > \dots > i_n.$$

Observe that $i_k = n$. Note also that given $\{i_2, \dots, i_{k-1}\}$ (together with H and x_1) G is uniquely determined, and given G , the set

$$FORW(G, H) = \{i_2, i_3, \dots, i_{k-1}\}$$

is uniquely determined. The neighborhood structure is not symmetrical as if $H = x_1x_2x_3x_4x_1$ and $G = x_1x_3x_4x_2x_1$, then $G \in PY(x_1, H)$, but $H \notin PY(x_1, G)$. We first prove the well-known fact that the size of $PY(x_1, H)$ is exponential.

Theorem 1 $|PY(x_1, H)| = 2^{n-2}$.

Proof: As mentioned above the tours $G \in PY(x_1, H)$ are uniquely determined by $FORW(G, H)$, which is a subset of $\{2, 3, \dots, n - 1\}$ (of cardinality $n - 2$). Since any subset (including the empty set and the whole set) determines a tour in $PY(x_1, H)$, and there are 2^{n-2} such subsets, we are done. ■

The fact that the pyramidal neighborhood can be searched in time $O(n^2)$ was proved for the first time by Klyaus [507]; for proofs of this assertion and its extensions, see Section 4 of Chapter 11.

Theorem 2 *We can find an optimal tour in $PY(x_1, H)$ in $O(n^2)$ time.*

Since every tour in $PY(x_1, H)$, when $H = x_1x_2 \dots x_nx_1$, either uses the arc x_1x_2 or the arc x_2x_1 (and either $x_{n-1}x_n$ or x_nx_{n-1}), the algorithm of Theorem 2 will not produce a good tour if these arcs are

expensive. One way of avoiding this problem is to consider the neighborhood $PCV(H) = \cup_{j=1}^n PY(j, H)$ instead (PCV stands for *pyramidal Carlier-Villon* as Carlier and Villon [162] introduced this neighborhood). Clearly, by Theorem 2, we can find an optimal tour in $PCV(H)$ in $O(n^3)$ time, by just running the algorithm of Theorem 2 n times.

It is not difficult to show that, for example, the well-known $2 - Opt$ neighborhood is a subset of PCV (i.e., $2 - Opt(H) \subset PCV(H)$) for the STSP. Deineko and Woeginger [248] proved that PCV covers at least 75 % of tours in 3-Opt. For some experimental results using PCV we refer the reader to [162].

2.2. The Assign Neighborhood and Its Variations

For the special case of $|Z| = \lfloor n/2 \rfloor$ (see the definition of Z below), this neighborhood was introduced in [414, 744]. Punnen [680] introduced the general definition of this neighborhood as well as its further extension (see the last paragraphs of this subsection).

Let $T = x_1x_2\dots x_nx_1$ be a tour and let $Z = \{x_{i_1}, x_{i_2}, \dots, x_{i_s}\}$ be a set of non-adjacent vertices of T , i.e., $2 \leq |i_k - i_r| \leq n - 2$ for all $1 \leq k < r \leq s$. The *assign neighborhood* of T with respect to Z , $N(T, Z)$, consists of the tours that can be obtained from T by removal of the vertices in Z one by one followed by an insertion of Z into the cycle derived after the removal. (Recall that, by the definition of insertion of several vertices into a cycle C in Subsection 1.2, the vertices of Z are inserted into different arcs of C .) For example,

$$N(x_1x_2x_3x_4x_5x_1, \{x_1, x_3\}) = \\ \{x_2x_4x_3x_5x_1x_2, x_2x_4x_1x_5x_3x_2, x_2x_4x_3x_5x_1x_2 : \{i, j\} = \{1, 3\}\}.$$

Theorem 3 [416, 680] *The neighborhood $N(T, Z)$ is $O(n^3)$ -searchable.*

Proof: Let $C = y_1y_2\dots y_{n-s}y_1$ be the cycle obtained from T after removal of Z and let $Y = \{z_1, z_2, \dots, z_s\}$. By the definition of insertion, we have $n - s \geq s$. Let ϕ be an injective mapping from Z to $Y = \{y_1, y_2, \dots, y_{n-s}\}$. (The requirement that ϕ is injective means that $\phi(z_i) \neq \phi(z_j)$ if $i \neq j$.) If we insert some z_i into an arc (y_j, y_{j+1}) , then the weight of C will be increased by $c(y_j, z_i) + c(z_i, y_{j+1}) - c(y_j, y_{j+1})$. Therefore, if we insert every z_i , $i = 1, 2, \dots, s$, into $(y_{\phi(i)}, y_{\phi(i)+1})$, the weight of C will be increased by

$$g(\phi) = \sum_{i=1}^s c(y_{\phi(i)}, z_i) + c(z_i, y_{\phi(i)+1}) - c(y_{\phi(i)}, y_{\phi(i)+1}).$$

Clearly, to find a tour of $N(T, Z)$ of minimum weight, it suffices to minimize $g(\phi)$ on the set of all injections ϕ from Z to Y . This can be done using the following weighted complete bipartite graph B . The partite sets of B are Z and Y . The weight of an edge $z_i y_j$ is set to be $c(y_j, z_i) + c(z_i, y_{j+1}) - c(y_j, y_{j+1})$.

By the definition of B , every maximum matching M of B corresponds to an injection ϕ_M from Z to Y . Moreover, the weights of M and ϕ_M coincide. A minimum weight maximum matching in B can be found by solving the assignment problem. Therefore, in $O(n^3)$ time, we can find the best tour in $N(T, Z)$. ■

Let $\text{ins}(n, s)$ be the number of tours in $N(T, Z)$, $s = |Z|$, and let $n \geq 5$. Since there are $k = n - s$ ways to insert x_1 in C , $k - 1$ ways to insert x_2 in C when x_1 has been inserted, etc., we obtain that $\text{ins}(n, s) = (n - s)(n - s - 1) \dots (n - 2s + 1)$. It is natural to ask what is the largest possible size of the assign neighborhood for the ATSP with n vertices. This question is answered in the following theorem. For a real r , $[r]_0$ ($[r]_1$, resp.) is the maximum integer (semi-integer, respectively) that does not exceed r (a semi-integer is a number of the form $p/2$, where p is an odd integer); for an integer m , $\sigma(m) = m \bmod 2$.

Theorem 4 [416] *For a fixed $n \geq 5$, the maximum size of the assign neighborhood equals*

$$\text{maxins}(n) = \frac{(n/2 + p_0)!}{(2p_0)!},$$

where $p_0 = \left\lceil \sqrt{\frac{1}{8}(n + \frac{9}{8})} + \frac{3}{8} \right\rceil_{\sigma(n)}$.

Proof: Assume first that n is even. Consider $f(p) = \text{ins}(n, n/2 - p)$, where p is a non-negative integer smaller than $n/2$. For $p \geq 1$, the difference $\Delta f(p) = f(p) - f(p - 1) = b(-2p(2p - 1) + (n/2 + p)) = bq(p)/2$, where $q(p) = -8p^2 + 6p + n$, $b = (n/2 + p - 1)(n/2 + p - 2) \dots (2p + 1)$. Clearly, $\text{sign}(\Delta f(p)) = \text{sign}(q(p))$. Therefore, $f(p)$ increases when $q(p) > 0$, and $f(p)$ decreases when $q(p) < 0$. For $p \geq 1$, $q(p)$ decreases and has a positive root $r = \sqrt{\frac{1}{8}(n + \frac{9}{8})} + \frac{3}{8}$. Thus, $f(p)$, where $p \in \{1, \dots, n/2\}$ is maximum for $p = [r]_0$.

Analogously, when n is odd, we obtain that $f(p)$ is maximum for $p = [r]_1$. ■

The following asymptotic formula provides us with an estimate on how large $\text{maxins}(n)$ is. Note that, for $2m \leq n \leq 2m + 1$, $\text{ins}(n, m) = \left\lfloor \frac{n+1}{2} \right\rfloor_0!$.

Theorem 5 [416] We have $\text{maxins}(n) = \Theta\left(\frac{e^{\sqrt{n/2}} \lfloor \frac{n+1}{2} \rfloor!}{n^{\frac{1}{4} + \lfloor \frac{1}{2} \rfloor \sigma(n)}}\right)$.

The value of $\text{maxins}(n)$ is the maximum known size of a neighborhood searchable in time $O(n^3)$. We can combine several neighborhoods $N(T, Z)$ of T for various sets Z and construct a polynomially searchable neighborhood of size $\Theta(e^{\sqrt{n/2}} \lfloor \frac{n+1}{2} \rfloor! n^k)$ for every natural number k [416]. Do there exist larger polynomially searchable neighborhoods? Some stronger question is raised in Section 6.

For large values of n , the time $O(n^3)$ appears to be too high to be used in local search algorithms. Thus, the following result is of interest (observe that $(n-1)! = 2^{\Theta(n \log n)}$):

Theorem 6 [416] 1. For every β , $0 < \beta \leq 2$, there is an $O(n^{1+\beta})$ -algorithm for finding the best among $2^{\Theta(n \log n)}$ tours.

2. For every positive integer r there exists an $O(r^5 n)$ -time algorithm for constructing the best among $\Omega(r^n)$ tours.

Corollary 12 in Subsection 3.1 of this chapter implies that the first part of this theorem cannot be, in some sense, improved.

Punnen [680] suggested an extension of the assign neighborhood. There we allow one to remove paths rather than vertices and insert them back. The rationale behind this extension is to preserve “good” parts of the current tour T . For example, one can use the following strategy: the cheaper an arc a in T the larger the probability to preserve a . The reader can easily add his/her own details to this approach. In practice this more general approach seems to be more promising.

A small number of computational experiments on a fairly straightforward implementation of a local search heuristic using Punnen’s extension of the assign neighborhood were performed by Gutin, Punnen and Zverovich (unpublished). In general, the results appeared to be too modest in comparison to those of the state-of-the-art heuristics. To improve the results, one should probably combine Punnen’s extension of the assign neighborhood with some “classical” neighborhoods.

2.3. The Balas-Simonetti Neighborhood

The following neighborhood, was introduced by Balas [66] and studied computationally by Balas and Simonetti in [80]. Although this neighborhood has been defined for both ATSP and STSP, we consider here only the ATSP case. Let k be any integer with $2 \leq k \leq n$, let

$H = x_1x_2 \dots x_nx_1$ be a tour, and define the neighborhood of H , denoted by $BS_k(x_1, H)$, as follows (see [80]).

A tour $G = x_{\pi(1)}x_{\pi(2)}x_{\pi(3)} \dots x_{\pi(n)}x_{\pi(1)}$ (with $\pi(1) = 1$) belongs to $BS_k(x_1, H)$ if and only if for all integers i and j with $j \geq i + k$ we have $\pi(i) < \pi(j)$.

In other words if a vertex, x_j , lies k or more places after a vertex x_i in H , then x_j must lie after x_i in G (when one walks along the tour, starting at x_1). Furthermore, the inequality $j \geq i + k$ is not taken modulo n , which is why the vertex x_1 has a special function in the above definition. The above neighborhood is not symmetric, as seen by the following example with $n = 5$ and $k = 3$. Let $H = x_1x_2x_3x_4x_5x_1$, $G = x_1x_4x_2x_5x_3x_1$, and note that $G \in BS_k(x_1, H)$, but $H \notin BS_k(x_1, G)$ as x_3 does not come after x_4 in H .

In the proof of Theorem 8, we will illustrate how to find an optimal solution in $BS_k(x_1, H)$ in $O(nk^22^k)$ time, by reducing the problem to a shortest path problem in an auxiliary digraph, G^* , with at most $nk(k + 1)2^{k-2}$ arcs. Note that for a fixed k this implies a linear algorithm, which turns out to be quite effective in practice [80].

To the best of our knowledge, the following theorem that provides bounds for the size of $BS_k(x_1, H)$ is a new result.

Theorem 7 For $n \geq k(k + 1)$, $\left(\frac{k}{e}\right)^{n-1} < |BS_k(x_1, H)| \leq k^{n-1}$. Furthermore, we have that $|BS_2(x_1, H)| = Fib(n)$, where $Fib(n)$ is the n th Fibonacci number.

Proof: We will start by proving that $\left(\frac{k}{e}\right)^n \leq |BS_k(x_1, H)|$. First assume that $n = ik + 1$, where i is an integer, and without loss of generality let $H = x_1x_2 \dots x_nx_1$. Define \mathcal{F} to be the set of all tours of the form $x_{\pi(1)}x_{\pi(2)}x_{\pi(3)} \dots x_{\pi(n)}x_{\pi(1)}$ with $\pi(1) = 1$ and $\{\pi(jk + 2), \pi(jk + 3), \dots, \pi(jk + k + 1)\} = \{jk + 2, jk + 3, \dots, jk + k + 1\}$, for all $j = 0, 1, \dots, i - 1$. This means that we only allow tours that permute the first k vertices (not including x_1), the next k vertices, etc, but not vertices between these sets. Clearly the number of tours in \mathcal{F} is $(k!)^i$ and $\mathcal{F} \subset BS_k(x_1, H)$. Using Stirling’s formula we get $(k!)^i > (\sqrt{2\pi k}k^k e^{-k})^{(n-1)/k} > (k/e)^{n-1}$. This proves the case when $n - 1$ is divisible by k .

If $n = ik + j$, where $1 < j \leq k$, then we proceed as follows. We still have i sets of size k we may permute, but now we have $j - 1$ vertices left over (we do not count x_1). We choose the i sets as follows: $\{x_2, x_3, \dots, x_{k+1}\}$ is the first set, x_{k+2} is a left-over vertex,

$$\{x_{k+3}, x_{k+4}, \dots, x_{2k+2}\}$$

is the second set, x_{2k+3} is a left-over vertex, etc. After all $j - 1$ left-over vertices have been used, the sets will not have any vertices between them. Since $n \geq k(k + 1)$ this can be done.

We can obtain a tour in $BS_k(x_1, H)$ by permuting the sets and then retaining every left-over vertex or inserting it in one of the places available. Since we have, on average, at least k possibilities for every left-over vertex, we obtain that

$$|BS_k(x_1, H)| > (k/e)^{ik} k^j > (k/e)^{n-1}.$$

We will now prove that $|BS_k(x_1, H)| \leq k^{n-1}$. We will build the tour starting at x_1 , and show that we have at most k choices at each position. Assume that we have built a partial tour (i.e. a path), $x_1 x_{\pi(2)} x_{\pi(3)} \dots x_{\pi(i)}$, and let l be the smallest index not used yet (i.e., $l = \min(\{1, 2, \dots, n\} - \{1, \pi(2), \pi(3), \dots, \pi(i)\})$). Clearly we can only place x_j in the $(i + 1)$ th position if $l \leq j \leq l + k - 1$. Therefore we get that $|BS_k(x_1, H)| \leq k^{n-1}$.

Finally we prove that $|BS_2(x_1, H)| = Fib(n)$ by induction. Note that $BS_2(x_1, H)$ contains all tours where we only have swapped positions of neighbors in H (and x_1 stays fixed). Observe that $|BS_2(x_1, H)| = Fib(n)$ holds for $n = 2$ and $n = 3$, and assume that it holds for $n - 1$ and $n - 2$, $n \geq 4$. Let $H = x_1 x_2 \dots x_n x_1$, and note that by the induction hypothesis there are $Fib(n - 2)$ tours starting with $x_1 x_3 x_2$, and that there are $Fib(n - 1)$ tours starting with $x_1 x_2$. Since there are no other possibilities we get that $|BS_2(x_1, H)| = Fib(n - 2) + Fib(n - 1) = Fib(n)$

■

Note that $Fib(n)$ is approximately $0.7236 \times 1.618^{n-1}$.

Theorem 8 [66] *We can find an optimum in $BS_k(x_1, H)$ in $O(nk^2 2^k)$ time.*

Proof: We transform the problem to a minimum cost path problem, in an auxiliary digraph $D_k(x_1, H)$, which we will simply denote by D_k . The vertices of D_k are tuples (i, j, S^-, S^+) , such that there exists some tour $R = x_{\pi(1)} x_{\pi(2)} \dots x_{\pi(n)} x_{\pi(1)} \in BS_k(x_1, H)$ ($\pi(1) = 1$) such that the following holds:

1. $\pi(i) = j$;
2. $S^- = \{\pi(1), \pi(2), \dots, \pi(i - 1)\} \cap \{i, i + 1, \dots, n\}$;
3. $S^+ = \{\pi(i), \pi(i + 1), \dots, \pi(n)\} \cap \{1, 2, \dots, i - 1\}$.

We furthermore say that the tuple (i, j, S^-, S^+) is compatible with the tour R . Note that $|S^-| = |S^+| (= i - 1 - |\{\pi(1), \pi(2), \dots, \pi(i - 1)\} \cap$

$\{1, 2, \dots, i - 1\}$). An arc (x, y) is in D_k if $x = (i, j_x, S_x^-, S_x^+)$ and $y = (i + 1, j_y, S_y^-, S_y^+)$, and there exists some tour $R = \pi(1)\pi(2) \dots \pi(n)\pi(1)$, for which both x and y are compatible. Furthermore, if this is the case, then $S_y^- = S_x^- \cup (\{j_x\} \cap \{i + 1, i + 2, \dots, n\}) - \{i\}$ and $S_y^+ = S_x^+ \cup (\{i\} \cap (V - S_x^-)) - \{j_x\}$, where $V = \{1, 2, \dots, n\}$. Since the last two formulas can be proved similarly, we will show only the second one. It is straightforward to see that $S_y^+ = S_x^+ \cup (\{\pi(i), \pi(i + 1), \dots, \pi(n)\} \cap \{i\}) - \{j_x\}$. However,

$$\begin{aligned} \{\pi(i), \pi(i + 1), \dots, \pi(n)\} \cap \{i\} &= \\ (\{\pi(i), \pi(i + 1), \dots, \pi(n)\} \cup \{1, 2, \dots, i - 1\}) \cap \{i\} &= \\ (V - S_x^-) \cap \{i\}. & \end{aligned}$$

We will use the fact that S_y^- and S_y^+ are totally determined by S_x^- , S_x^+ , i and j_x several times below.

We will now show that there is a one-to-one correspondence between tours in $BS_k(x_1, H)$ and paths from $(1, 1, \emptyset, \emptyset)$ to $(n + 1, 1, \emptyset, \emptyset)$ in D_k . For an example, see Figure 6.1. Clearly any tour in $BS_k(x_1, H)$ has a corresponding path from $(1, 1, \emptyset, \emptyset)$ to $(n + 1, 1, \emptyset, \emptyset)$ in D_k , so now let P be a path from $(1, 1, \emptyset, \emptyset)$ to $(n + 1, 1, \emptyset, \emptyset)$ in D_k . Let $P = (1, 1, \emptyset, \emptyset)(2, \pi(2), S_2^-, S_2^+) \dots (n, \pi(n), S_n^-, S_n^+)(n + 1, 1, \emptyset, \emptyset)$. Now we show that $Q = x_1x_{\pi(2)}x_{\pi(3)} \dots x_{\pi(n)}x_1$ is a tour in $BS_k(x_1, H)$.

Note that if R is a tour compatible with $(i, \pi(i), S_i^-, S_i^+)$, then one can uniquely determine the first i elements in R (but not their order), as they are the ones with the following indices, $S_i^- \cup (\{1, 2, \dots, i - 1\} - S_i^+) \cup \pi(i)$. We will now show by induction that $(i, \pi(i), S_i^-, S_i^+)$ is compatible with Q and $1, \pi(2), \pi(3), \dots, \pi(i)$ are distinct. Clearly this is true for $i = 2$. So assume that it is true for $i - 1$ ($i \geq 3$). As $1, \pi(2), \pi(3) \dots, \pi(i - 1)$ are uniquely determined by $(i - 1, \pi(i - 1), S_{i-1}^-, S_{i-1}^+)$, and there is an arc from $(i - 1, \pi(i - 1), S_{i-1}^-, S_{i-1}^+)$ to $(i, \pi(i), S_i^-, S_i^+)$ we must have that $\pi(i)$ is distinct from $1, \pi(2), \pi(3) \dots, \pi(i - 1)$ (as there is a tour that is compatible with both $(i - 1, \pi(i - 1), S_{i-1}^-, S_{i-1}^+)$ and $(i, \pi(i), S_i^-, S_i^+)$). Furthermore S_i^- and S_i^+ are totally determined by $S_{i-1}^-, S_{i-1}^+, i - 1$ and $\pi(i - 1)$, so therefore $(i, \pi(i), S_i^-, S_i^+)$ is compatible with Q . This completes the inductive proof. It now follows that all $(1, 1, \emptyset, \emptyset), (2, \pi(2), S_2^-, S_2^+) \dots (n + 1, 1, \emptyset, \emptyset)$ are compatible with Q and Q is a tour. Therefore Q is a tour in $BS_k(x_1, H)$.

Now by setting the cost of the arc xy , where $x = (i, j_x, S_x^-, S_x^+)$ and $y = (i + 1, j_y, S_y^-, S_y^+)$, to the cost of the arc $x_{j_x}x_{j_y}$ the cost of a path from $(1, 1, \emptyset, \emptyset)$ to $(n + 1, 1, \emptyset, \emptyset)$ is equal to the cost of the corresponding tour given by this path. For an example, see the path illustrated by the thick rectangles in Figure 6.1, which corresponds to the tour $x_1x_3x_4x_2x_6x_5x_1$.

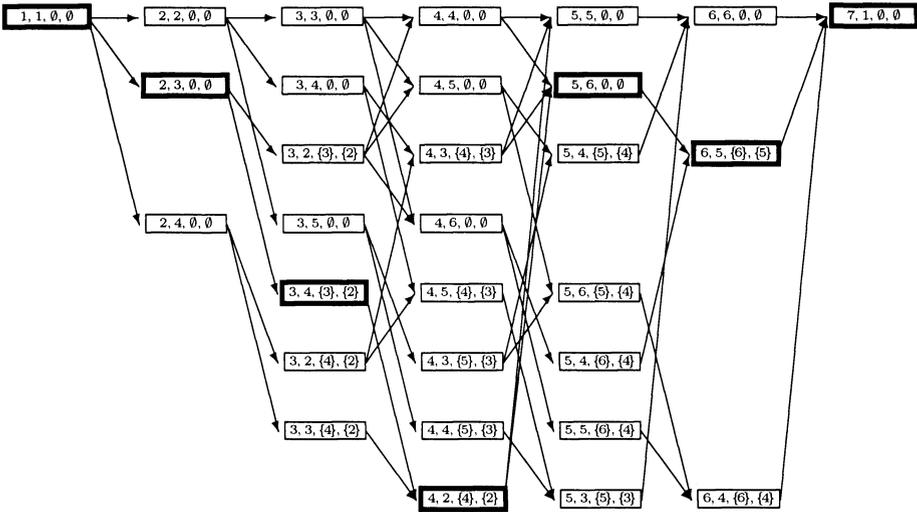


Figure 6.1. Example when $n = 6$ and $k = 3$. The path connecting the thick nodes correspond to the tour $x_1x_3x_4x_2x_6x_5x_1$.

In [66] Balas proved that the number of vertices in D_k of the form (i, j_x, S_x^-, S_x^+) is equal to $(k + 1)2^{k-2}$, for any given i , with $k + 1 \leq i \leq n - k + 1$. It is easy to see that for every remaining i there are at most $(k + 1)2^{k-2}$ such vertices. This implies that the total number of vertices in D_k is at most $n(k + 1)2^{k-2}$.

We will now prove that the out-degree of any vertex in D_k is at most k . Let $x = (i, j_x, S_x^-, S_x^+)$ be some vertex in D_k , and let $y = (i + 1, l, S_y^-, S_y^+)$ be an out-neighbor of x . As mentioned above, S_y^- and S_y^+ are totally determined by S_x^-, S_x^+, i and j_x . Let $p = \min\{S_y^+ \cup \{i + 1\}\}$, (or $\min\{S_x^+ \cup \{i, i + 1\} - j_x\}$, which is equivalent), and note that p is the smallest index, such that x_p is not used in the path $x_{\pi(1)}x_{\pi(2)} \dots x_{\pi(i)}$. Now it is not difficult to see that $p \leq l \leq p + k - 1$. Therefore, the out-degree of x is at most k .

This implies that the number of arcs in D_k is bounded by $nk(k + 1)2^{k-2}$. So finding a cheapest path of length n in D_k can be done in $O(nk(k + 1)2^{k-2}) = O(nk^22^k)$ time. Since any tour in $BS_k(x_1, H)$ corresponds to a path of length n in D_k and any path of length n in D_k corresponds to a tour in $BS_k(x_1, H)$, we are done. ■

The above algorithm can be generalized, such that the constant k depends on the position on the tour. That is, given a set of integers $\{k(i) : i = 1, 2, \dots, n\}$, in the definition of the neighborhood, we have

that, if $j \geq i + k(i)$ then $\pi(i) < \pi(j)$. This generalization is not too difficult to implement and a description of this can be found in [80].

Furthermore, using the above generalizations the algorithm can be extended to time window problems as well as the time target problems. We refer the reader to [80] for more details.

The algorithm of this subsection has been tested in [80], and seems to work well as a local search algorithm. One may compare the algorithm in this subsection with k -Opt as they both perform local changes. However, the neighborhood described here has exponential size, and can be searched in linear time (for constant k), whereas k -Opt has polynomial size neighborhoods, and non-linear running time (in n). The algorithms of this subsection seem to perform particularly well on TSP instances that model actual cities and distances between cities. One reason for this could be that cities tend to cluster in metropolitan areas. For a more detailed discussion of this topic we refer the reader to [80].

Finally we note that the digraphs D_k , described in the proof of Theorem 8, can be computed using the values n and k , independently of the input (\vec{K}_n, c) . Then it remains to add the costs, when the input becomes known. This preprocessing may, in many cases, save considerable time as actually constructing the digraphs D_k is more time consuming than computing the shortest path in D_k .

3. Upper Bounds for Neighborhood Size

The aim of this section is to provide upper bounds for ATSP neighborhood sizes. In Subsection 3.1 we prove upper bounds depending on the time to search the neighborhood. In Subsection 3.2 we obtain an upper bound of the size of polynomially searchable neighborhoods.

3.1. General Upper Bounds

This subsection is based on [421]. The next theorem provides an upper bound on the size of an ATSP neighborhood depending on the time to search the neighborhood. It is realistic to assume that the search algorithm spends at least one unit of time on every arc that it considers.

Theorem 9 *Let N_n be an ATSP neighborhood that can be searched in time $t(n)$. Then $|N_n| \leq \max_{1 \leq n' \leq n} (t(n)/n')^{n'}$.*

Proof: Let $D = (\vec{K}_n, c)$ be an instance of the ATSP and let H be the tour that our search algorithm returns, when run on D . Let E denote the set of arcs in D , which the search algorithm actually examine; observe that $|E| \leq t(n)$ by the assumption above. Let the arcs of $A(H) - E$ have high enough cost and the arcs in $A(D) - E - A(H)$ have low enough

cost, such that all tours in N_n must use all arcs in $A(H) - E$ and no arc in $A(D) - E - A(H)$. This can be done as H has the lowest cost of all tours in N_n . Now let D' be the digraph obtained by contracting the arcs in $A(H) - E$ and deleting the arcs not in E , and let n' be the number of vertices in D' . Note that every tour in N_n corresponds to a tour in D' and, thus, the number of tours in D' is an upper bound on $|N_n|$. In a tour of D' , there are at most $d^+(i)$ possibilities for the successor of a vertex i , where $d^+(i)$ is the out-degree of i in D' . Hence we obtain that

$$|N_n| \leq \prod_{i=1}^{n'} d^+(i) \leq \left(\frac{1}{n'} \sum_{i=1}^{n'} d^+(i) \right)^{n'} \leq \left(\frac{t(n)}{n'} \right)^{n'},$$

where we applied the arithmetic-geometric mean inequality. ■

Corollary 10 *Let N_n be an ATSP neighborhood that can be searched in time $t(n)$. Then $|N_n| \leq \max\{e^{t(n)/e}, (t(n)/n)^n\}$, where e is the basis of natural logarithms.*

Proof: Let $U(n) = \max_{1 \leq n' \leq n} (t(n)/n')^{n'}$. By differentiating $f(n') = (t(n)/n')^{n'}$ with respect to n' we can readily obtain that $f(n')$ increases for $1 \leq n' \leq t(n)/e$, and decreases for $t(n)/e \leq n' \leq n$. Thus, if $n \leq t(n)/e$, then $f(n')$ increases for every value of $n' < n$ and $U(n) = f(n) = (t(n)/n)^n$. On the other hand, if $n \geq t(n)/e$ then the maximum of $f(n')$ is for $n' = t(n)/e$ and, hence, $U(n) = e^{t(n)/e}$. ■

It follows from the proof of Corollary 10 that

Corollary 11 *For $t(n) \geq en$, we have $|N_n| \leq (t(n)/n)^n$.*

Note that the restriction $t(n) \geq en$ is important since otherwise the bound of Corollary 11 can be invalid. Indeed, if $t(n)$ is a constant, then for n large enough the upper bound implies that $|N_n| = 0$, which is not correct since there are neighborhoods of constant size that can be searched in constant time: consider a tour T , delete three arcs in T and add three other arcs to form a new tour T' . Clearly, the best of the two tours can be found in constant time by considering only the six arcs mentioned above. Notice that this observation was not taken into account in [248], where the bound $|N_n| \leq (2t(n)/n)^n$ was claimed. That bound is invalid for $t(n) \leq n/2$.

Corollary 10 immediately implies that linear-time algorithms can be used only for neighborhoods of size at most $2^{O(n)}$. Using Corollary 10, it is also easy to show the following:

Corollary 12 *The time required to search an ATSP neighborhood of size $2^{\Theta(n \log n)}$ is $\Omega(n^{1+\alpha})$ for some positive constant α .*

3.2. Upper Bounds for Polynomial Time Searchable Neighborhoods

Deineko and Woeginger [248] conjectured that there is no ATSP neighborhood of cardinality at least $\beta(n-1)!$ for any positive constant β provided $P \neq NP$. In this subsection based on [421] we prove that there is no ATSP neighborhood of cardinality at least $\beta(n-k)!$ for any constant $\beta > 0$ and fixed integer k provided $NP \not\subseteq P/poly$.

$P/poly$ is a well-known complexity class in structural complexity theory, see e.g. [82], and it is widely believed that $NP \not\subseteq P/poly$ for otherwise, as proved in the well-known paper by Karp and Lipton [499], it would imply that the so-called polynomial hierarchy collapses on the second level, which is thought to be very unlikely. The idea that defines $P/poly$ is that, for each input size n , one is able to compute a polynomial-sized “key for size n inputs”. This is called the “advice for size n inputs”. It is allowed that the computation of this “key” may take time exponential in n (or worse). $P/poly$ stands for the class of problems solvable in polynomial time (in input size n) given the poly-sized general advice for inputs of size n . For formal definitions of $P/poly$ and related non-uniform complexity classes, consult [82].

Let S be a finite set and \mathcal{F} be a family of subsets of S such that \mathcal{F} is a *cover* of S , i.e., $\cup\{F : F \in \mathcal{F}\} = S$. The well-known *covering problem* is to find a cover of S containing the minimum number of sets in \mathcal{F} . While the following *greedy covering algorithm* (GCA) does not always produce a cover with minimum number of sets, GCA finds asymptotically optimal results for some wide classes of families, see e.g. [526]. GCA starts by choosing a set F in \mathcal{F} of maximum cardinality, deleting F from \mathcal{F} and initiating a “cover” $\mathcal{C} = \{F\}$. Then GCA deletes the elements of F from every remaining set in \mathcal{F} and chooses a set H of maximum cardinality in \mathcal{F} , appends it to \mathcal{C} and updates \mathcal{F} as above. The algorithm stops when \mathcal{C} becomes a cover of S . The following lemma have been obtained independently by several authors, see Proposition 10.1.1 in [47].

Lemma 13 *Let $|S| = s$, let \mathcal{F} contain f sets, and let every element of S be in at least δ sets of \mathcal{F} . Then the cover found by GCA is of cardinality at most $1 + f(1 + \ln(\delta s/f))/\delta$.*

Using this lemma we can prove the following:

Theorem 14 *Let \mathcal{T} be the set of all tours of the ATSP on n vertices. For every fixed integer $k \geq 1$ and constant $\beta > 0$, unless $\text{NP} \subseteq \text{P/poly}$, there is no set Π of permutations on $\{1, 2, \dots, n\}$ of cardinality at least $\beta(n-k)!$ such that every neighborhood $N_\Pi(T)$, $T \in \mathcal{T}$, is polynomial time searchable.*

Proof: Assume that, for some $k \geq 1$ and $\beta > 0$, there exists a set Π of permutations on $\{1, 2, \dots, n\}$ of cardinality at least $\beta(n-k)!$ such that every neighborhood $N_\Pi(T)$, $T \in \mathcal{T}$, is polynomial time searchable. Let $\mathcal{N} = \{N_\Pi(T) : T \in \mathcal{T}\}$. Consider the covering problem with $S = \mathcal{T}$ and $\mathcal{F} = \mathcal{N}$. Observe that $|S| = |\mathcal{F}| = (n-1)!$. To see that every tour is in at least $\delta = (n-k)!$ neighborhoods of \mathcal{N} , consider a tour $Y = y_1 y_2 \dots y_n y_1$ and observe that for every $\pi \in \Pi$,

$$Y \in N_\Pi(y_{\pi^{-1}(1)} y_{\pi^{-1}(2)} \dots y_{\pi^{-1}(n)} y_{\pi^{-1}(1)}).$$

By Lemma 13 there is a cover \mathcal{C} of S with at most $O(n^k \ln n)$ neighborhoods from \mathcal{N} . Since every neighborhood in \mathcal{C} is polynomial time searchable and \mathcal{C} contains only polynomial number of neighborhoods, we can construct the best tour in polynomial time provided \mathcal{C} is found. To find \mathcal{C} (which depends only on n , and not on the instance of the ATSP) we need exponential time and, thus, the fact that the best tour can be computed in polynomial time implies that $\text{NP} \subseteq \text{P/poly}$. ■

4. Diameters of Neighborhood Structure Digraphs

The *distance* from a vertex x to a vertex y of an unweighted digraph D is 0 if $x = y$, the length of the shortest path from x to y , if D has one, and ∞ , otherwise. The *diameter* of a digraph D is the maximum distance in D . Given neighborhood $N(T)$ for every tour T in \vec{K}_n (i.e., a neighborhood structure), the corresponding *neighborhood digraph* (of order $(n-1)!$) is a directed graph with vertex set consisting of all tours in \vec{K}_n and arc set containing a pair (T', T'') if and only if $T'' \in N(T')$. The diameter of the neighborhood graph is one of the most important characteristics of the neighborhood structure and the corresponding local search scheme [162, 248, 372]. Clearly, a neighborhood structure with a neighborhood digraph of smaller diameter seems to be more powerful than one with a neighborhood digraph of larger diameter, let alone a neighborhood structure whose digraph has infinite diameter (in the last case, some tours are not “reachable” from the initial tour during local search procedure).

4.1. Diameters of Pyramidal and the Balas-Simonetti Neighborhood Digraphs

If the diameter of the pyramidal neighborhood digraph is d_{PY} , then Theorem 1 implies that $(2^{n-2})^{d_{PY}} \geq (n-1)!$ and, thus, $d_{PY} = \Omega(\log n)$. The next theorem implies that $d_{PY} = \Theta(\log n)$.

Theorem 15 [162] *The diameter of the neighborhood digraph corresponding to $PY(x_1, H)$ is at most $\lceil \log_2 n \rceil$.*

Observe that this theorem implies that the diameter d_{PCV} of the pyramidal Carlier-Villon neighborhood introduced in the end of Subsection 2.1 is also $\Theta(\log n)$. Indeed, $|PCV(H)| \leq n2^{n-2}$ and, thus, $d_{PCV} = \Omega(\log n)$. On the other hand, $PY(x_1, H) \subseteq PCV(H)$ and, hence, $d_{PCV} \leq d_{PY}$. The next theorem is a new result.

Theorem 16 *The neighborhood digraph of $BS_k(x_1, H)$ is of diameter $O(n)$.*

Proof: Since $BS_k(x_1, H)$ includes $BS_2(x_1, H)$ for every $k \geq 2$, it suffices to prove this theorem for $k = 2$. Let $H = x_1x_2 \dots x_nx_1$ and $G = x_{\pi(1)}x_{\pi(2)} \dots x_{\pi(n)}x_{\pi(1)}$. We will show that there is a sequence of tours $G = G_1, G_2, \dots, G_{n+1} = H$, such that $G_{i+1} \in BS_2(x_1, G_i)$, $i = 1, 2, \dots, n$.

We find G_i as follows. When i is even, and $G_{i-1} = x_1x_{z_2} \dots x_{z_n}x_1$ then let $G_i = x_1x_{w_2} \dots x_{w_n}x_1$ such that the following holds. The first two vertices on G_i are a sorted version of the first two vertices on G_{i-1} (i.e., $\{x_{z_1}, x_{z_2}\} = \{x_{w_1}, x_{w_2}\}$ and $w_1 < w_2$), the next two vertices are a sorted version of the next two vertices on G_{i-1} , etc. When i is odd, we leave the first vertex unchanged, but then the next two vertices are a sorted version of the next two vertices on G_{i-1} , etc.

Observe that $G_i \in BS_k(x_1, G_{i-1})$ holds. The claim that $G_{n+1} = H$ is equivalent to the assertion that tours G_1, G_2, \dots, G_{n+1} “sort” numbers $\pi(1), \pi(2), \dots, \pi(n)$ (to $1, 2, \dots, n$). It remains to observe that this assertion follows from Part (c) of Problem 28-1 in [218], p. 651, i.e., from the fact that every odd-even sorting network is a sorting network. The details are left to the interested reader. ■

The result of this theorem can be improved to $O(n/k)$. We leave details to the interested reader.

4.2. Diameter of Assign Neighborhood Digraphs

For a positive integer $k \leq n/2$, the neighborhood digraph $\Gamma(n, k)$ of the assign neighborhood has vertex set formed by all tours in \vec{K}_n . An

arc (T, R) is in $\Gamma(n, k)$ if there exists a set Z of k non-adjacent vertices of T such that $R \in N(T, Z)$. Clearly, (T, R) is in $\Gamma(n, k)$ if and only if (R, T) is in $\Gamma(n, k)$, i.e., $\Gamma(n, k)$ is symmetric. We denote by $\text{dist}_k(T, R)$ the distance (i.e., the length of a shortest path) from T to R in $\Gamma(n, k)$.

For a tour T in \vec{K}_n , let \mathcal{I}_{nk} denote the family of all sets of k non-adjacent vertices in T . Clearly, the neighborhood $N_k(T)$ of a tour T in $\Gamma(n, k)$ equals

$$\cup_{Z \in \mathcal{I}_{nk}} N(T, Z).$$

Thus if, for some k , $i(n, k) = |\mathcal{I}_{nk}|$ is polynomial in n , then from the fact that $N(T, Z)$ is polynomially searchable it follows that $N_k(T)$ is polynomially searchable. Otherwise, $N_k(T)$ may be non-polynomially searchable. Since polynomially searchable $N_k(T)$ are of our interest, we start with evaluating $i(n, k)$ in Theorem 17. It follows from Theorem 17 that, for fixed k , $i(n, k)$ and $i(n, n - k)$ are polynomial.

Theorem 17 [418] $i(n, k) = \binom{n-k}{k} + \binom{n-k-1}{k-1}$.

Corollary 18 [418] *If p is a non-negative fixed integer, then $N_{p+1}(T)$ and $N_{\lfloor (n-p)/2 \rfloor}(T)$ are polynomially searchable ($p < \lfloor n/2 \rfloor$).*

Proof: This follows from Theorem 17 taking into consideration that $\binom{m}{k} = \binom{m}{m-k}$. ■

One can easily prove that if n is even, then $\Gamma(n, n/2)$ consists of an exponential number of strongly connected components and, thus, its diameter is infinite (for example, $x_1x_2\dots x_nx_1$ and $x_1\dots x_{n-2}x_nx_{n-1}x_1$ belong to different strong components of this digraph). Therefore, below we consider $\Gamma(n, k)$ for $k < n/2$ only.

Theorem 19 $\text{diam}(\Gamma(n, \lfloor (n - 1)/2 \rfloor)) \leq 4$.

Proof: We assume that $n \geq 5$, as for $2 \leq n \leq 4$ this claim can be verified directly. Let $C = x_1x_2\dots x_nx_1$ and $T = y_1y_2\dots y_ny_1$ be a pair of distinct tours in \vec{K}_n . Put $k = \lfloor (n - 1)/2 \rfloor$. We will prove that $\text{dist}_k(T, C) \leq 4$, thus showing that $\text{diam}(\Gamma(n, k)) \leq 4$.

We call a vertex v *even* (*odd*) with respect to C if $v = x_j$, where $1 \leq j \leq n$ and j is even (odd). For a set of vertices X of \vec{K}_n , let X_{odd} (X_{even}) be the set of odd (even) vertices in X .

First we consider the case of even n , i.e. $k = n/2 - 1$. The proof in this case consists of two steps. At the first step, we show that there exists a tour T'' whose vertices alternate in parity and such that $\text{dist}_k(T, T'') \leq 2$. Moreover, T'' has a pair of consecutive vertices which are also consecutive

in C . At the second step, we will see that $\text{dist}_k(T'', C) \leq 2$ as the odd and even vertices of T'' (except for the vertices of the above pair) can be separately reordered to form C . Thus, we will conclude that $\text{dist}_k(T, C) \leq 4$. Now, we proceed with the proof.

Clearly, T has a pair y_j, y_{j+1} such that y_{j+1} is odd and y_j is even. Let

$$Z = \{y_{j+2}, y_{j+4}, \dots, y_{j+2k}\}$$

and let $|Z_{\text{odd}}| = s$. Remove the vertices of Z from T and then insert the s odd vertices of Z into the arcs $y_{j+1}y_{j+3}, \dots, y_{j+2s-1}y_{j+2s+1}$ and $k - s$ even vertices of Z into the arcs

$$y_{j+2s+1}y_{j+2s+3}, y_{j+2s+3}y_{j+2s+5}, \dots, y_{j+2k-1}y_{j+2k+1}.$$

We have obtained a tour

$$T' = y_j y_{j+1} v_{j+2} y_{j+3} v_{j+4} y_{j+5} \dots y_{j+2k-1} v_{j+2k} y_{j+2k+1} y_j,$$

where $\{v_{j+2}, \dots, v_{j+2k}\} = Z$.

Let $Z' = \{y_{j+3}, y_{j+5}, \dots, y_{j+2k+1}\}$ and let $|Z'_{\text{even}}| = t$. Since the number of odd vertices in $V(\vec{K}_n) - \{y_j, y_{j+1}\}$ is equal to $k = |Z_{\text{odd}}| + |Z'_{\text{odd}}| = s + k - t$, we obtain that $s = t$. Remove Z' from T' and insert the t even vertices of Z' into the arcs $y_{j+1}v_{j+2}, v_{j+2}v_{j+4}, v_{j+6}v_{j+8}, \dots, v_{j+2s-2}v_{j+2s}$ and the $k - s$ odd vertices of Z' into the arcs

$$v_{j+2s+2}v_{j+2s+4}, \dots, v_{j+2k-2}v_{j+2k}, v_{j+2k}y_j.$$

We have derived a tour $T'' = u_1 u_2 \dots u_n u_1$. Clearly, the vertices of T'' alternate in parity, i.e., for every m , if u_m is odd, then u_{m+1} is even.

Now we prove that the processes of insertion of Z and Z' can be performed in such a way that T'' contains a pair of consecutive vertices which are also consecutive in C (i.e. there exist indices p and q such that $u_p = x_q$ and $u_{p+1} = x_{q+1}$). Since $1 < |Z'| < n$, there exists a pair of distinct indices i, m such that $x_i, x_m \in Z'$ and $x_{i+1}, x_{m-1} \notin Z'$. Without loss of generality, we assume that i is odd. We consider two cases.

Case 1: $|Z'_{\text{odd}}| \geq 2$. We prove that we may choose index $q = i$. Since $x_{i+1} \notin Z'$ and $i + 1$ is even, either $y_j = x_{i+1}$ or $x_{i+1} \in Z_{\text{even}}$. If $x_{i+1} \in Z_{\text{even}}$, in the process of insertion of Z , we insert x_{i+1} into $y_{j+2k-1}y_{j+2k+1}$, i.e. $x_{i+1} = y_{j+2k}$. In the process of insertion of Z' , we insert x_i into $v_{j+2k}y_j$ if $x_{i+1} = y_j$ or into $v_{j+2k-2}v_{j+2k}$, otherwise (i.e. $x_{i+1} = y_{j+2k}$).

Case 2: $|Z'_{\text{odd}}| = 1$. Thus, m is even. Since $n \geq 6$, it follows that $|Z'_{\text{even}}| \geq 2$. Analogously to Case 1, one may take $q = m - 1$.

Therefore, without loss of generality, we assume that $u_{n-1} = x_i$, $u_n = x_{i+1}$. Since $\{u_2, u_4, \dots, u_{2k}, x_{i+1}\} = C_{\text{even}}$, we can delete $\{u_2, \dots, u_{2k}\}$ from T'' and insert it into the obtained cycle to get the tour C' given by $C' = u_1x_{i+3}u_3x_{i+5}u_5 \dots u_{2k-1}x_{i-1}u_{n-1}x_{i+1}u_1$. Analogously, we can delete $\{u_1, u_3, \dots, u_{2k-1}\}$ from C' and insert it into the obtained cycle to get C . We conclude that $\text{dist}_k(T, C) \leq 4$.

Now let n be odd; then $k = (n - 1)/2$. Notice that, without loss of generality, we may assume that $x_n = y_n$ (to fix the initial labelings of T and C). Consider tours $X = x_1x_2 \dots x_nx_{n+1}x_1$ and $Y = y_1y_2 \dots y_{n-1}y_ny_{n+1}y_1$ in $\overleftrightarrow{K}_{n+1}$, where $y_n = x_n$, $y_{n+1} = x_{n+1}$. If we assume that $j = n$, $j + 1 = n + 1$, we can obtain, analogously to the case of even n , a tour Y'' such that the vertices of Y'' alternate in parity (with respect to their indices in X), x_{n+1} follows x_n in Y'' and $\text{dist}_k(Y, Y'') \leq 2$. Now if $i = n$ and $i + 1 = n + 1$, then we can show, similarly to the case of even n , that $\text{dist}_k(Y'', X) \leq 2$ and, thus, $\text{dist}_k(Y, X) \leq 4$. Notice that, in the whole process of constructing X from Y , we have never removed x_n and x_{n+1} or inserted any vertex into the arc x_nx_{n+1} . Thus, we could contract the arc x_nx_{n+1} to x_n and obtain C from T in four “steps”. This shows that $\text{dist}_k(T, C) \leq 4$. ■

We can extend Theorem 19 using the following:

Theorem 20 [418] *Let $\text{dist}_k(T, C) = 1$ for tours T and C and let m be an integer smaller than k . Then, $\text{dist}_m(T, C) \leq \lceil k/m \rceil$.*

Corollary 21 *For every positive m ,*

$$\text{diam}(\Gamma(n, m)) \leq 4\lceil (n - 1)/2 \rceil / m.$$

In particular, if p is a positive integral constant, then $\text{diam}(\Gamma(n, \lfloor (n - p)/2 \rfloor)) \leq 8$ for every $n \geq 2p + 1$.

Proof: The first inequality follows directly from the above two theorems and the triangle inequality for distances in graphs. The first inequality implies the second one. Indeed, $n \geq 2p + 1$ implies

$$\frac{(n - 1)/2}{(n - p - 1)/2} \leq 2, \quad \frac{\lfloor (n - 1)/2 \rfloor}{\lfloor (n - p)/2 \rfloor} \leq 2.$$

■

5. Domination Analysis

Recall that the domination number, $\text{domn}(\mathcal{H}, n)$, of a heuristic \mathcal{H} for the TSP is the maximum integer $k = k(n)$ such that, for every instance

\mathcal{I} of the TSP on n vertices, \mathcal{H} produces a tour T which is not worse than at least k tours in \mathcal{I} including T itself.

In this section, we describe some important results in domination analysis of TSP heuristics. In Subsection 5.1, domination numbers of ATSP and STSP heuristics are compared. In Subsection 5.2, we consider TSP heuristics of large domination number, at least $\Omega((n - 2)!)$. It turns out that several well-known heuristics have a large domination number. In Subsection 5.3 we briefly discuss bounds on the largest possible domination number of a polynomial time TSP heuristic. TSP heuristics of small domination number are considered in Subsection 5.4. It is somewhat surprising that such heuristics as the greedy, nearest neighbor and double tree algorithms are all of domination number 1.

5.1. Domination Number of Heuristics for the STSP and ATSP

In this subsection we observe that, in certain cases (e.g., for lower bounds on domination number), it is enough to study heuristics for the ATSP since one can readily obtain similar results on heuristics for the STSP from the corresponding ones for the ATSP. This justifies that we mostly study ATSP heuristics in this section. We also prove an assertion that relates the maximum possible domination numbers of polynomial time heuristics for the ATSP and $\overleftrightarrow{\text{STSP}}$.

For a tour $H = x_1x_2 \dots x_nx_1$ in \overleftrightarrow{K}_n , the tour $x_nx_{n-1} \dots x_1x_n$ will be denoted by \overleftarrow{H} .

Since an instance of the STSP can be transformed into an “equivalent” instance of the ATSP by replacing every edge xy of K_n by the pair xy, yx of arcs of costs equal to the cost of the edge xy , every heuristic for the ATSP can be used for the STSP⁵. Observe that a polynomial time heuristic \mathcal{A} for the ATSP with domination number $d(n)$ has domination number at least $d(n)/2$ for the STSP. The factor $\frac{1}{2}$ is due to the fact that a pair Q, \overleftarrow{Q} of tours in \overleftrightarrow{K}_n is indistinguishable in K_n .

One of the central natural questions on the domination number is to determine the maximum domination number of a polynomial time heuristic for the ATSP. We call it the *maximum domination number of the ATSP*. We can introduce the similar parameter for the STSP. The STSP being, in a sense, a special case of the ATSP, one may suspect that the maximum domination number of the STSP is larger than that of the ATSP. We will now show that this is not true.

⁵This, in particular, allows one to apply ATSP heuristics to the STSP without redefining them, see, e.g. Subsection 5.4.

Theorem 22 [417] *For every polynomial heuristic \mathcal{H} for the STSP, there is a polynomial heuristic \mathcal{H}' for the ATSP such that $\text{domn}(\mathcal{H}', n) \geq \text{domn}(\mathcal{H}, n)$.*

Proof: To an instance of ATSP with cost function c assign an instance of STSP defined on the same set of vertices and with cost function c' defined by $c'(x, y) = \frac{1}{2}(c(x, y) + c(y, x))$ for every $x \neq y$. Let $T = x_1x_2\dots x_nx_1$ be a tour found by the heuristic \mathcal{H} applied to (K_n, c') and let \mathcal{S} be the set of all tours R in (K_n, c') such that $c'(T) \leq c'(R)$. The cycle $T = x_1x_2\dots x_nx_1$ can be considered as a tour in $(\overleftrightarrow{K}_n, c)$. For a tour Q in $(\overleftrightarrow{K}_n, c)$, let Q^-, Q^+ be defined as follows:

$$\{Q^-, Q^+\} = \{Q, \overline{Q}\}, c(Q^-) = \min\{c(Q), c(\overline{Q})\}.$$

This theorem now follows from the fact that for every $Z \in \mathcal{S}$, $c(T^-) \leq c(Z^+)$ as $c(T^-) \leq c'(T) \leq c'(Z) \leq c(Z^+)$. ■

5.2. Heuristics of Domination Number $\Omega((n - 2)!)$

While the assertion of the next theorem for odd n was already known to Rev Kirkman (see [104], p. 187), the even case result was only established by Tillson [792] as a solution to the corresponding conjecture by J.C. Bermond and V. Faber (who observed that the decomposition does not exist for $n = 4$ and $n = 6$).

Theorem 23 *For every $n \geq 2$, $n \neq 4$, $n \neq 6$, there exists a decomposition of $A(\overleftrightarrow{K}_n)$ into tours.*

Let $T(\overleftrightarrow{K}_n)$ ($\tau(n, c)$) be the total cost of all tours (the average cost of a tour) in $(\overleftrightarrow{K}_n, c)$. Since every arc of \overleftrightarrow{K}_n is contained in $(n - 2)!$ tours, $\tau(n, c) = T(\overleftrightarrow{K}_n)/(n - 1)! = (n - 2)!c(\overleftrightarrow{K}_n)/(n - 1)!$, and hence, $\tau(n, c) = c(\overleftrightarrow{K}_n)/(n - 1)$. This formula can also be shown using linearity of expectation. For the STSP, it is easy to see that $\tau(n, c) = 2c(K_n)/(n - 1)$, where as above $\tau(n, c)$ is the average cost of a tour.

The following result was first obtained by Sarvanov [739] when n is odd, and Gutin and Yeo [423] when n is even. As we see below Theorem 24 allows us to show that certain heuristics are of domination number at least $(n - 2)!$.

Theorem 24 *Consider any instance of the ATSP and a tour H such that $c(H) \leq \tau(n, c)$. If $n \neq 6$, then H is not worse than at least $(n - 2)!$ tours.*

Proof: The result is trivial for $n = 2, 3$. If $n = 4$, the result follows from the simple fact that the most expensive tour T in \vec{K}_n has cost $c(T) \geq c(H)$.

Assume that $n \geq 5$ and $n \neq 6$. Let $D_1 = \{C_1, \dots, C_{n-1}\}$ be a decomposition of the arcs of \vec{K}_n into tours (such a decomposition exists by Theorem 23). Given a tour R in \vec{K}_n , clearly there is an automorphism of \vec{K}_n that maps C_1 into R . Therefore, if we consider D_1 together with the decompositions $(D_1, \dots, D_{(n-1)!})$ of \vec{K}_n obtained from D_1 using all automorphisms of \vec{K}_n which map the vertex 1 into itself, we will have every tour of \vec{K}_n in one of D_i 's. Moreover, every tour is in exactly $n - 1$ decompositions D_i 's (by mapping a tour C_i into a tour C_j $1 \leq i \neq j \leq n - 1$) we fix the automorphism).

Choose the most expensive tour in each of D_i and form a set \mathcal{E} from all distinct tours obtained in this manner. Clearly, $|\mathcal{E}| \geq (n - 2)!$. As $\sum_{i=1}^{n-1} c(C_i) = c(\vec{K}_n)$, every tour T of \mathcal{E} has cost $c(T) \geq \tau(n, c)$. Therefore, $c(H) \leq c(T)$ for every $T \in \mathcal{E}$. ■

To see that the assertion of Theorem 24 is almost best possible, choose a tour H and an arc a not in H . Let every arc in H be of cost one, let $c(a) = n(n - 1)$ and let every arc not in $A(H) \cup \{a\}$ be of cost zero. Clearly the cost of H is less than the average (which is $n^2/(n - 1)$), but only tours using the arc a have higher cost. Thus, H is not worse than exactly $(n - 2)! + 1$ tours (including itself).

The first remark in Subsection 5.1 and Theorem 24 imply that, for the STSP, the assertion similar to Theorem 24 holds with $(n - 2)!$ replaced by $(n - 2)!/2$. However, Rublineckii [733] proved the following stronger result.

Theorem 25 *Consider an instance (K_n, c) of the STSP and a tour H such that $c(H) \leq \tau(n, c)$. Then H is not worse than at least $(n - 2)!$ tours when n is odd and $(n - 2)!/2$ tours when n is even.*

The ideas in the proof of Theorem 25 are similar to those used in the proof of Theorem 24. Instead of Theorem 23, Rublineckii [733] used a much simpler result that the edges K_n ($2K_n$) can be decomposed in edge-disjoint tours when n is odd (even), where $2K_n$ is the complete multigraph with 2 edges between every pair of distinct vertices.

The vertex insertion algorithm for the ATSP work as follows. First, we fix some ordering v_1, \dots, v_n of the vertices of \vec{K}_n . Then, we perform

$n - 1$ steps. On the first step we form the cycle $v_1v_2v_1$. On step k , $2 \leq k \leq n - 1$, given the k -cycle $v_{\pi(1)}v_{\pi(2)} \cdots v_{\pi(k)}v_{\pi(1)}$ from the previous step, we find the value j_0 of j , which minimizes the expression

$$c(v_{\pi(j)}, v_{k+1}) + c(v_{k+1}, v_{\pi(j+1)}) - c(v_{\pi(j)}, v_{\pi(j+1)}),$$

$1 \leq j \leq k$, and insert v_{k+1} between $v_{\pi(j_0)}$ and $v_{\pi(j_0+1)}$ forming a $(k + 1)$ -cycle. Clearly, the vertex insertion algorithm for the STSP differs from the ATSP one in the fact that it starts from a cycle with three vertices. The following theorem was first proved by E.M. Lifshitz (see [733]) for the STSP.

Theorem 26 *Let H_n be a tour constructed by the vertex insertion algorithm \mathcal{A} for the TSP with n vertices. Then $c(H_n) \leq \tau(n, c)$.*

Proof: We prove this result only for the ATSP by induction on n . The theorem is trivially true for $n = 2$. Let $H_{n-1} = v_{\pi(1)}v_{\pi(2)} \cdots v_{\pi(n-1)}v_{\pi(1)}$ be the cycle constructed in Step $n - 2$ of the algorithm and assume that in Step $n - 1$, it was decided to insert v_n between $v_{\pi(j_0)}$ and $v_{\pi(j_0+1)}$ in order to obtain H_n . Let V be the vertex set of \vec{K}_n and, for a partition $X \cup Y = V$, let $(X, Y) = \{(x, y) : x \in X, y \in Y\}$. Then, we have

$$\begin{aligned} c(H_n) &= \\ c(H_{n-1}) + c(v_{\pi(j_0)}, v_n) + c(v_n, v_{\pi(j_0+1)}) - c(v_{\pi(j_0)}, v_{\pi(j_0+1)}) &\leq \\ c(H_{n-1}) + \frac{\sum_{i=1}^{n-1} c(v_{\pi(i)}, v_n) + c(v_n, v_{\pi(i+1)}) - c(v_{\pi(i)}, v_{\pi(i+1)})}{n - 1} &= \\ c(H_{n-1}) + \frac{c(V - v_n, v_n) + c(v_n, V - v_n) - c(H_{n-1})}{n - 1} &\leq \\ \frac{(n - 2)\tau(n - 1, c) + c(v_n, V - v_n) + c(V - v_n, v_n)}{n - 1} &= \\ \frac{c(\vec{K}_n - v_n) + c(v_n, V - v_n) + c(V - v_n, v_n)}{n - 1} &= \tau(n, c), \end{aligned}$$

where $\tau(n - 1, c)$ is the average cost of a tour in $\vec{K}_n - v_n$. ■

Theorems 24 and 26 imply the following result (similar result holds for the STSP, see Theorem 25).

Theorem 27 [683] *For the ATSP vertex insertion algorithm \mathcal{A} and $n \neq 6$ we have $\text{domn}(\mathcal{A}, n) \geq (n - 2)!$.*

Gutin and Yeo [423] proved that the following ATSP algorithm always produces a tour of cost at most the average cost: choose an arc e such that the average cost of a tour through e is minimum, contract e and repeat the above choice and contraction until only two arcs remain. The output is the tour obtained from the two arcs together with the contracted ones. A similar algorithm was described by Vizing [810].

Given neighborhood structure N , the *best improvement* local search (LS) algorithm starts from an arbitrary tour; at every iteration it finds the best tour T' in the neighborhood $N(T)$ of the current tour T and replaces T by T' . The algorithm stops when $c(T') = c(T)$, in which case T is a *local optimum* with respect to N . Normally practical LS codes do not use the best improvement strategy; instead they find a better (than T) tour T' at every iteration as long as it is possible. This strategy saves running time and often yields better practical results, but the *first improvement* LS is difficult to formalize since the way to find the first improvement varies from code to code. Thus, let us restrict ourselves to the best improvement versions of 2-Opt and 3-Opt.

The k -Opt, $k \geq 2$, neighborhood of a tour T consists of all tours that can be obtained by deleting a collection of k edges (arcs) and adding another collection of k edges (arcs). Rublineckii [733] showed that every local optimum for 2-Opt and 3-Opt for the STSP is of cost at least the average cost of a tour and, thus, by Theorem 25 is of domination number at least $(n - 2)!/2$ when n is even and $(n - 2)!$ when n is odd. Observe that this result is of restricted interest since, to reach a k -Opt local optimum, one may need exponential time (see Section 3 in [463]). However, Punnen, Margot and Kabadi [684] managed to prove the following result.

Theorem 28 *For the STSP the best improvement 2-Opt algorithm produces a tour of cost at most $\tau(n, c)$ in at most*

$$O(\min\{n^3 \log n, n \log(c(H_0) - \tau(c, n))\})$$

iterations, where H_0 is the initial tour.

Punnen, Margot and Kabadi observed that Theorem 28 holds also for 3-Opt and the pyramidal Carlier-Villon neighborhood. The last result can be extended to the ATSP because of Theorem 22. It is pointed

out in [684] that analogous results hold also for the well-known Lin-Kernighan algorithm [563] and shortest path ejection chain algorithm of Glover [372, 682] (see also Chapter 8).

5.3. Bounds on Maximum Domination Number of Polynomial Heuristics

Clearly, unless $P=NP$, there is no polynomial time ATSP algorithm with domination number $(n-1)!$. Punnen, Margot and Kabadi [684] proved that unless $P=NP$, there is no polynomial time ATSP algorithm with domination number at least $(n-1)! - k$ for any constant k . This result can be extended from constant k to some function in n [684].

Gutin and Yeo [420] showed that, if there is a constant $r > 1$ such that for every sufficiently large k a k -regular digraph of order at most $rk-1$ can be decomposed into Hamiltonian cycles in polynomial time in n , then the maximum domination number of the ATSP is $\Theta((n-1)!)$. This result is of interest due to the fact that Häggkvist [428, 429] announced (not published) that the above Hamiltonian decomposition exists for every $1 < r \leq 2$, see also Alspach et al. [19]. His approach is constructive and implies a polynomial algorithm to find such a decomposition. If Häggkvist's result holds, the main theorem in [420] implies that, in polynomial time, one can always find a tour, which is not worse than 50% of all tours.

Notice that the 50% threshold may seem to be easily achievable at first glance: just find the best in a large sample \mathcal{S} of randomly chosen tours. A random tour has approximately a 50% chance of being better than 50% of all tours. However, in this approach the probability that the best tour of \mathcal{S} is more expensive than 50% of all tours is always positive (if we consider only polynomial size samples of random tours). The difficulty of the problem by Glover and Punnen is well illustrated by the problem [605] to find a tournament on n vertices with the number of Hamiltonian cycles exceeding the average number of Hamiltonian cycles in a tournament of order n . This problem formulated long time ago has not been solved yet.

5.4. Heuristics with Small Domination Numbers

Chapters 9 and 10 describe experimental results indicating that the greedy algorithm performs rather badly in the computational practice of the ATSP and STSP, see also [200, 377, 425, 463]. The aim of this subsection is to show that greedy-type algorithms are no match, with respect to the domination number, to heuristics considered in Subsection 5.2. This provides some theoretical explanation why “being greedy” is

not so good for the TSP. This subsection is based on Gutin, Yeo and Zverovich [424].

Before considering greedy-type algorithms in detail, we would like to notice that Punnen, Margot and Kabadi [684] recently constructed STSP instances for which the well-know double tree heuristic produces the unique worst tour. Note that these instances even satisfy the triangle inequality, i.e., for them the double tree heuristic computes a tour which is at most only twice more expensive than the cheapest tour. The authors of [684] also showed that the famous Christofides heuristic is of domination number at most $\lceil n/2 \rceil!$.

The *greedy algorithm* (GR) builds a tour in (\vec{K}_n, c) by repeatedly choosing the cheapest eligible arc until the chosen arcs form a tour; an arc $a = uv$ is *eligible* if the out-degree of u in D and the in-degree of v in D equal zero, where D is the digraph induced by the set S of chosen arcs, and a can be added to S without creating a non-Hamiltonian cycle. The *nearest neighbor algorithm* (NN) starts its tour from a fixed vertex i_1 , goes to the nearest vertex i_2 (i.e., $c(i_1, i_2) = \min\{c(i_1, j) : j \neq i_1\}$), then to the nearest vertex i_3 (from i_2) distinct from i_1 and i_2 , etc. Computational experience with NN for the ATSP and STSP is discussed in Chapters 9 and 10, and [200, 463]. We will also consider a stronger version of NN, the *repetitive NN algorithm* (RNN), which starts NN from each of the vertices in turn and chooses the best tour. In the rest of the chapter we assume that NN starts from vertex 1.

The following theorem was first proved in [424]. We give a different proof by adapting the proof of a much more general result from [419]. The result holds for a wide family of CO problems including the assignment problem, i.e., the domination number of the greedy algorithm for the assignment problem is proved to be 1.

Theorem 29 *The domination number of GR for the TSP is 1.*

Proof: This proof holds for both ATSP and STSP, but for simplicity we assume that we deal with the STSP. We will consider tours of STSP as sets of their edges. For a set $S = \{e_1, \dots, e_s\}$ of edges forming a partial tour in K_n (i.e., this set of edges can be extended to a tour), $Z(e_1, \dots, e_s)$ denotes the set of edges not in S such that each edge from $Z(e_1, \dots, e_s)$ can be added to S to form a (larger) partial tour.

Let $T' = \{e'_1, e'_2, \dots, e'_n\}$ be an arbitrary fixed tour and let T be an arbitrary tour distinct from T' . It is easy to see that

$$\sum_{j=0}^{n-1} |Z(e'_1, e'_2, \dots, e'_j) \cap T| < n(n+1)/2. \tag{1}$$

Let $M > n$, let $c(e'_i) = iM$ for each $e'_i \in T'$ and, for $e \notin T'$, let $c(e) = 1 + jM$ if $e \in Z(e'_1, e'_2, \dots, e'_{j-1})$ but $e \notin Z(e'_1, e'_2, \dots, e'_j)$. Clearly, GR constructs T' and $c(T') = Mn(n+1)/2$.

Let $T = \{e_1, e_2, \dots, e_k\}$. Assume that $c(e_i) \in \{aM, aM + 1\}$. Then clearly

$$e_i \in Z(e'_1, e'_2, \dots, e'_{a-1}),$$

but $e_i \notin Z(e'_1, e'_2, \dots, e'_a)$, so e_i lies in $Z(e'_1, e'_2, \dots, e'_j) \cap T$, provided $j \leq a - 1$. Thus, e_i is counted a times in the sum in (1). Hence,

$$\begin{aligned} c(T) &= \sum_{i=1}^n c(e_i) \leq n + M \sum_{j=0}^{n-1} |Z(\{e'_1, e'_2, \dots, e'_j\}) \cap T| \\ &\leq n + M(n(n+1)/2 - 1) = n - M + c(T'), \end{aligned}$$

which is less than the cost of T' as $M > n$. Since GR finds T' , and T is arbitrary, we see that GR finds the unique most expensive tour. ■

The proof of Theorem 29 implies that the domination number of NN for TSP is also 1 (indeed, NN will construct the same tour as GR). However, the following two theorems show that the situation is slightly better for RNN.

Theorem 30 [424] *Let $n \geq 4$. The domination number of RNN for the ATSP is at least $n/2$ and at most $n - 1$.*

Proof: We first consider the following instance of the ATSP, which proves that RNN for the ATSP has domination number at most $n - 1$. Let $N > 2n$. Let all arcs $(i, i + 1)$, $1 \leq i < n$, have cost iN , all arcs $(i, i + 2)$, $1 \leq i \leq n - 2$, cost $iN + 1$, and all remaining forward arcs (i, j) cost $iN + 2$. Let a backward arc (i, j) have cost $(j - 1)N$.

When NN tour T starts at $i \notin \{1, n\}$, it has the form $(i, 1, 2, \dots, i - 1, i + 1, i + 2, \dots, n, i)$ and cost

$$\ell = \sum_{k=1}^{n-1} kN - N + 1.$$

When T starts at 1 or n , we simply have $T = (1, 2, \dots, n, 1)$ of cost $\sum_{k=1}^{n-1} kN > \ell$. Let \mathcal{F} denote the set of all tours T described above (note that $|\mathcal{F}| = n - 1$). Observe that any tour in \mathcal{F} has cost at least ℓ . Let C be any tour not in \mathcal{F} . Let B denote the set of backward arcs in C , and define the length of a backward arc (i, j) by $i - j$. Let q denote the sum of the lengths of the arcs in B . Since C is a tour (and therefore there is a path from n to 1) we have $q \geq n - 1$. The cost of C is at

most $\sum_{i=1}^n (iN + 2) - qN - |B|N$, since if (i, j) is an arc in B , then the corresponding term $iN + 2$ in the sum can be replaced by the real cost $(j - 1)N = iN + 2 - (i - j + 1)N - 2$ of the arc. We have

$$\sum_{i=1}^n (iN + 2) - qN - |B|N \leq \ell + 2n + N(n + 1 - q - |B|) - 1.$$

Since C is not in \mathcal{F} we have $|B| \geq 2$, implying that $2n + N(n + 1 - q - |B|) - 1$ is negative except for the case of $q = n - 1$ and $|B| = 2$. We may conclude that the cost of C is less than ℓ , as $q = n - 1$ and $|B| = 2$ would imply that C belongs to \mathcal{F} . Therefore all cycles not in \mathcal{F} have cost less than those in \mathcal{F} .

In order to prove that RNN has domination number at least $n/2$, assume that this is false, and proceed as follows. RNN constructs n tours, but several of them may coincide. By the assumption, there exist at least three tours that coincide. Let $F = x_1x_2 \dots x_nx_1$ be a tour such that $F = F_i = F_j = F_k$, where F_s is the tour obtained by starting NN at x_s and x_i, x_j and x_k are distinct. Without loss of generality, we may assume that $i = 1$ and $2 < j \leq 1 + (n/2)$. For every m , with $j < m \leq n$, let C_m be the tour obtained by deleting the arcs $(x_i, x_{i+1}), (x_j, x_{j+1}), (x_m, x_{m+1})$ and adding the arcs

$$(x_i, x_{j+1}), (x_m, x_{i+1}), (x_j, x_{m+1}).$$

Note that $c(C_m) \geq c(F)$, since $c(x_i, x_{i+1}) \leq c(x_i, x_{j+1})$ (because we used NN from x_i to construct F_i), $c(x_j, x_{j+1}) \leq c(x_j, x_{m+1})$ (since we used NN from x_j to construct F_j) and $c(x_m, x_{m+1}) \leq c(x_m, x_{i+1})$ (since NN chose the arc x_mx_{m+1} on F_j , when the arc x_mx_{i+1} was available). Therefore the cost of F is at most that of $F, C_{j+1}, C_{j+2}, \dots, C_n$, implying that the domination number is at least $n - j + 1 \geq n/2$, a contradiction. ■

We call a tour $x_1x_2 \dots x_nx_1$, $x_1 = 1$, of the STSP *pyramidal* if $x_1 < x_2 < \dots < x_k > x_{k+1} > \dots > x_n$ for some index k . Since every pyramidal tour $x_1x_2 \dots x_nx_1$, $x_1 = 1$, is determined by the set $\{x_2, x_3, \dots, x_{k-1}\}$ or the set $\{x_{k+1}, x_{k+2}, \dots, x_n\}$ (clearly, $x_k = n$), we obtain that the number of pyramidal tours of the STSP is 2^{n-3} .

The next theorem gives an upper bound for the domination number of RNN for the STSP. Even though the theorem leaves a possibility that this domination number is exponential, it is still much smaller than $\Theta((n - 2)!)$.

Theorem 31 [424] *Let $n \geq 4$. The domination number of RNN for the STSP is at most 2^{n-3} .*

Proof: We consider the following instance of the STSP, which proves that RNN for the STSP has domination number at most 2^{n-3} . Let $N > 2n$. Let all edges $(i, i + 1)$, $1 \leq i < n$, have cost iN , all edges $(i, i + 2)$, $1 \leq i \leq n - 2$, cost $iN + 1$, and all remaining edges (i, j) , $i < j$, cost $iN + 2$.

Let c_{RNN} be the cost of the cheapest tour constructed by RNN. It is straightforward to verify that

$$c_{\text{RNN}} = c(12 \dots n1) = \sum_{i=1}^{n-1} iN + N + 2. \tag{2}$$

Let $T = x_1x_2 \dots x_nx_1$ be a tour in K_n , $x_1 = 1$; we orient all edges of T such that T becomes a directed cycle T' . Some of arcs in T' are forward, others are backward. For a backward arc $e = (j, i)$, we define its length as $q(e) = j - i$. We denote the sum of the lengths of backward arcs in T' by $q(T')$. (By the definition of a backward arc the length of every backward arc is positive.) Let c_{max} be the cost of the most expensive non-pyramidal tour T . Since the number of pyramidal tours is 2^{n-3} , to prove this theorem it suffices to show that $c_{\text{max}} < c_{\text{RNN}}$.

Observe that $q(T') \geq n$ for every T' corresponding to a non-pyramidal tour T . Let H be a non-pyramidal tour of cost c_{max} , and let $e_i = (i, j)$ be an arc of H' . If e_i is forward, then $c(e_i) \leq iN + 2$, and if e_i is backward, then $c(e_i) \leq jN + 2 = iN + 2 - q(e_i)N$. Thus,

$$c_{\text{max}} \leq \sum_{i=1}^n (iN + 2) - q(H')N \leq \sum_{i=1}^{n-1} iN + 2n$$

as $q(H') \geq n$. Since $N > 2n$ and by (2), we conclude that indeed $c_{\text{max}} < c_{\text{RNN}}$. ■

By the observation in the first paragraph of Subsection 5.1 and the lower bound in Theorem 30, the domination number of RNN for the STSP is at least $n/4$. It would be interesting to find the exact values of the domination number of RNN for the ATSP and STSP.

6. Further Research

Exponential neighborhoods can be included into a quite general approach in combinatorial optimization (CO): restrict the feasible set of solutions of a CO problem such that one can find the best solution of the restricted problem in polynomial time. This method, which we suggest to call the *polynomial restriction approach* (PRA) is somewhat dual to the analysis of polynomial solvable cases of the TSP: while in the latter

one restricts instances to consider, in the PRA we restrict the solution set for all instances of the TSP. There is some interaction between the two approaches, see e.g. Glover and Punnen [382], but in essence they are quite different. Notice that PRA may be of interest not only for exponential neighborhoods; non-neighborhood type sets of exponential size, where the best tour can be computed in polynomial time, may be used in exact algorithms (see below) or in certain meta-heuristics.

The following approach is obviously hardly practical, but perhaps its modifications may be of interest to practical exact algorithms. All tours of the ATSP can be enumerated and represented as leaves of a special rooted tree \mathcal{T} as follows. The root of \mathcal{T} (e.g. the first level of \mathcal{T}) is the vertex 1. Every node of the t th level of \mathcal{T} corresponds to a path $i_1 i_2 \dots i_t$ in \vec{K}_n such that $i_1 = 1$, and every edge of \mathcal{T} is of the form $\{i_1 i_2 \dots i_{t-1}, i_1 i_2 \dots i_t\}$ and has weight $c(i_{t-1} i_t)$ (except for $t = n$ when the weight is $c(i_{n-1} i_n) + c(i_n i_1)$). It is clear how to develop a simple branch-and-bound algorithm using \mathcal{T} : search \mathcal{T} by the means of the depth first search. The well-known Held-Karp dynamic programming algorithm [443] solves the ATSP to optimality in time $O(n^2 2^n)$. A simple modification of this algorithm can be used to find a cheapest Hamiltonian path between a pair of given vertices in $\vec{K}_{\log n}$ in time $O(n \log^2 n)$. This modification can be applied to cut $\log n$ last levels of \mathcal{T} , i.e., visit in the worst case less than $n^{\log n} (n - \log n)!$ leaves instead of $(n - 1)!$ leaves.

Our study of exponential neighborhoods for the ATSP suggests the following natural question.

Problem 32 *Do there exist polynomially searchable neighborhoods of size more than $\Theta(e^{\sqrt{n/2}} \lfloor \frac{n+1}{2} \rfloor! n^k)$ for any positive integer k ?*

The following question is stronger in a sense; it was raised by Deineko and Woeginger [248], who conjectured that the answer to Problem 33 is yes (under the assumption that $P \neq NP$).

Problem 33 *Do there exist polynomially searchable neighborhoods of size at least $\lfloor \alpha(n - 1) \rfloor!$ for some fixed $\alpha > \frac{1}{2}$?*

While one can see certain progress in the theoretical study of exponential neighborhoods, their use in computational algorithms has been less successful so far. We hope that this chapter will motivate extensive experimental study of various exponential neighborhoods.

The following problem, which we raised earlier, is one of the central questions in domination analysis for the TSP.

Problem 34 *Determine the maximum domination number of a polynomial heuristic for the ATSP (STSP).*

We provided exact values and bounds for the domination number of various heuristics for the ATSP and STSP. Observe that, for a given TSP instance \mathcal{I} , most of TSP heuristics will retain domination number if we increase the cost of every arc (edge) in \mathcal{I} by the same positive constant M . This implies that such heuristics will have the same domination number even if we restrict the set of instances from all ATSP (STSP) instances to those for which the triangle inequality holds.

The Euclidean TSP is a special class of STSP instances with triangle inequality and it is of great importance to practice. It would be quite interesting to obtain domination number results for Euclidean TSP heuristics (where the set of instances is restricted to the Euclidean TSP ones).

The domination number reflects the worst case behavior of a heuristic. If the worst case instances of the TSP are rather untypical for some heuristic, the domination number may not indicate the true value of the heuristic. Perhaps, certain probabilistic parameters, such as the average domination number, may provide further indication of the quality of the heuristic.

Acknowledgments We would like to thank Santosh Kabadi, Denis Naddef and Abraham Punnen for very helpful remarks and suggestion. The research of the first author was partially supported by an EPSRC grant.

Chapter 7

PROBABILISTIC ANALYSIS OF THE TSP

A. M. Frieze

Department of Mathematics, Carnegie Mellon University, USA

alan@random.math.cmu.edu

J. E. Yukich

Department of Mathematics, Lehigh University, USA

jey0@lehigh.edu

1. Introduction

In this chapter we study the Hamiltonian cycle and Traveling Salesman problem from a probabilistic point of view. Here we try to elucidate the properties of typical rather than worst-case examples. Structurally, one hopes to bring out the surprising properties of typical instances. Algorithmically the hope is that one can in some way explain the successful solution of large problems, much larger than that predicted by worst-case analysis. This of course raises the question of what do we mean by typical? The mathematical view of this is to define a probability space Ω of instances and study the expected properties of ω drawn from Ω with a given probability measure.

Our discussion falls naturally into two parts: the independent case and the Euclidean case. The independent case will include a discussion of the existence of Hamiltonian cycles in various classes of random graphs and digraphs. We will then discuss algorithms for finding Hamiltonian cycles which are both fast and likely to succeed. We include a discussion of extension-rotation constructions and the variance reduction technique of Robinson and Wormald. Following this we consider Traveling Salesman Problems where the coefficients are drawn independently. We describe both exact and approximate algorithms. We include a section on open problems.

After this we survey stochastic results for the total edge length of the Euclidean TSP. Here the cities are assumed to be points in \mathbb{R}^d and the distance between points is the usual Euclidean distance. We describe ways to prove a.s. limit theorems and concentration inequalities for the total edge length of the shortest tour on a random sample of size n , as $n \rightarrow \infty$. The focus is on presenting probabilistic techniques which not only describe the total edge length of the random Euclidean TSP, but which also describe (or have the potential to describe) the behavior of heuristics. The approach centers around the boundary functional method as well as the isoperimetric methods of Rhee and Talagrand. We conclude with a section on open problems.

1.1. Probabilistic Preliminaries

We first collect some basic probabilistic tools which will be needed in the sequel.

Jensen's inequality. A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called *convex* if for all $x, y \in \mathbb{R}$ and all $0 \leq \lambda \leq 1$ we have

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

In geometric terms this says that each point on the chord between $(x, f(x))$ and $(y, f(y))$ is above the graph of f . Jensen's inequality says that if X is a random variable with finite mean $\mathbb{E}[X]$ and if f is convex, then

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X]). \tag{1}$$

Generalized Chebyshev inequality. Let $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ be an increasing function and let X be a positive random variable. Then for all $t > 0$

$$\mathbb{P}[X \geq t] \leq \mathbb{E}[f(X)]/f(t). \tag{2}$$

If we replace X by $|X - \mathbb{E}X|$ and let $f(x) = x^2$ we obtain a special case of (2), namely

$$\mathbb{P}[|X - \mathbb{E}X| > t] \leq \text{Var}X/t^2. \tag{3}$$

With high probability. A sequence of events \mathcal{E}_n , $n \geq 1$, is said to occur with high probability (whp) if $\lim_{n \rightarrow \infty} \mathbb{P}[\mathcal{E}_n] = 1$.

Stochastic convergence. Let X_1, X_2, \dots, X be random variables on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We say that

- (a) $X_n \rightarrow X$ almost surely, written $\lim_{n \rightarrow \infty} X_n = X$ a.s., if

$$\mathbb{P}[\omega \in \Omega : X_n(\omega) \rightarrow X(\omega) \text{ as } n \rightarrow \infty] = 1.$$
- (b) $X_n \rightarrow X$ in L^1 or in mean if

$$\mathbb{E}|X_n - X| \rightarrow 0 \text{ as } n \rightarrow \infty.$$
- (c) $X_n \rightarrow X$ completely, written $\lim_{n \rightarrow \infty} X_n = X$ c.c., if for all $\epsilon > 0$,

$$\sum_{n=1}^{\infty} \mathbb{P}[|X_n - X| > \epsilon] < \infty.$$

Binomial random variable. For all $n \in \mathbb{N}$ and $0 < p < 1$, $B(n, p)$ denotes a binomial random variable with parameters n and p . Chernoff's bound for the binomial says that for all $0 \leq \epsilon \leq 1$

$$\mathbb{P}[|B(n, p) - np| \geq \epsilon np] \leq 2 \exp(-\epsilon^2 np/3). \tag{4}$$

See e.g. Alon and Spencer [18].

2. Hamiltonian Cycles in Random Graphs

2.1. Models of Random Graphs

In this section we describe two simple models of a random graph and their relationship. The random graph $G_{n,m}$ is chosen uniformly at random from the collection $\mathcal{G}_{n,m}$, where $\mathcal{G}_{n,m}$ is the set of graphs with vertex set $[n] = \{1, 2, \dots, n\}$ and exactly m edges. Since $|\mathcal{G}_{n,m}| = \binom{N}{m}$, $N = \binom{n}{2}$, each graph in $\mathcal{G}_{n,m}$ has probability $1/\binom{N}{m}$ of being selected. This model was intensely studied in a seminal sequence of papers by Erdős and Rényi [277, 278] and has since become a well established branch of combinatorics, see e.g. the book of Bollobás [124]. The related model $G_{n,p}$ (here $0 \leq p \leq 1$) is defined as follows: It has vertex set $[n]$ and each of the N possible edges is independently included with probability p and excluded with probability $1 - p$. Thus if $G = ([n], E)$,

$$\mathbb{P}[G_{n,p} = G] = p^{|E|}(1 - p)^{N - |E|}.$$

Thus for example if $p = \frac{1}{2}$ then all graphs with vertex set $[n]$ are equally likely.

The next thing to observe is that *conditional on having m edges*, $G_{n,p}$ is distributed as $G_{n,m}$. Thus if \mathcal{P}_n is any graph property for the set of graphs with vertex set $[n]$ then

$$\mathbb{P}[G_{n,p} \in \mathcal{P}_n] = \sum_{m=0}^N \binom{N}{m} p^m (1 - p)^{N - m} \mathbb{P}[G_{n,m} \in \mathcal{P}_n]. \tag{5}$$

The number of edges of $G_{n,p}$ is the binomial random variable $B(N, p)$ and if its mean Np is large, then $B(N, p)$ is concentrated around it. More precisely, Chernoff's bound on the binomial (4) implies that if $Np/\log n \rightarrow \infty$ then

$$\mathbb{P}[|B(N, p) - Np| \geq \sqrt{KNp \log n}] \leq 2n^{-K/3} \quad (6)$$

which tends to zero as $n \rightarrow \infty$. Thus, plausibly, with (5) and (6), one can argue simultaneously about properties of $G_{n,p}$ and $G_{n,m}$. This is not true all the time, but it is in many cases. Much of the interest in the theory of random graphs concerns properties that occur *with high probability* (whp). Erdős and Rényi proved many beautiful results about random graphs from this viewpoint. Suppose $p = p(n)$ depends on n :

- If $np - \log n \rightarrow \infty$ then $G_{n,p}$ is connected whp.
- If $np = c > 1$, c constant, then $G_{n,p}$ has a unique (giant) component of size order n , whp.

They also observed that for many properties \mathcal{P}_n there is a threshold probability $p_0 = p_0(n)$ such that if $p/p_0 \rightarrow 0$ then $G_{n,p} \notin \mathcal{P}_n$ whp and if $p/p_0 \rightarrow \infty$ then $G_{n,p} \in \mathcal{P}_n$ whp. For example if $\mathcal{A}_n = \{G \text{ contains a copy of } K_4\}$ then

$$\begin{aligned} \mathbb{P}[G_{n,p} \in \mathcal{A}_n] &= o(1) && \text{if } n^{2/3}p \rightarrow 0. \\ \mathbb{P}[G_{n,p} \in \mathcal{A}_n] &= 1 - o(1) && \text{if } n^{2/3}p \rightarrow \infty. \end{aligned}$$

Thus, one line of research in random graphs is the determination of thresholds for graph properties.

2.2. Existence Results

Erdős and Rényi did not manage to establish the threshold for the existence of a Hamiltonian cycle in a random graph. It took about twenty years to solve this problem.

2.2.1 The Threshold for Hamiltonicity. A breakthrough came when Pósa [672] proved the following theorem.

Theorem 1 *If $K > 16$ is constant and $p \geq \frac{K \log n}{n}$ then $G_{n,p}$ is Hamiltonian whp.*

Proof: We give the details of the proof because it is elegant, technically unchallenging, and illustrative of the most basic methods of proof in this area.

We first show that $G_{n,p}$ has a Hamiltonian path whp. For a graph $G = (V, E)$ we let $\lambda(G)$ denote the length of the longest path in G . Observe that

G has a Hamiltonian path iff $\lambda(G) > \lambda(G - v)$ for all $v \in V$.

For all $1 \leq i \leq n$, let \mathcal{E}_i be the event $\{\lambda(G_{n,p}) = \lambda(G_{n,p} - i)\}$. Then

$$\mathbb{P}[G_{n,p} \text{ has no Hamiltonian path}] \leq \sum_{i=1}^n \mathbb{P}[\mathcal{E}_i] = n\mathbb{P}[\mathcal{E}_n]. \tag{7}$$

We will show that

$$\mathbb{P}[\mathcal{E}_n] \leq n^{3-K/4} + n^{-K/4}. \tag{8}$$

Let $H = G_{n,p} - n$ and notice that H is distributed as $G_{n-1,p}$. Let $P = (x_0, x_1, \dots, x_k)$ be any longest path of H and let

$$X = \{x : H \text{ contains a path of length } k \text{ from } x_0 \text{ to } x\}.$$

We will show that

$$\mathbb{P}[|X| \leq \frac{n}{4}] \leq n^{3-K/4}. \tag{9}$$

Assuming (9) and noting that \mathcal{E}_n implies there is no edge joining X to n in $G_{n,p}$, it follows that

$$\begin{aligned} \mathbb{P}[\mathcal{E}_n] &\leq \mathbb{P}[|X| \leq \frac{n}{4}] + \mathbb{P}[\mathcal{E}_n \mid |X| > \frac{n}{4}] \\ &\leq n^{3-K/4} + (1-p)^{n/4} \\ &\leq n^{3-K/4} + n^{-K/4}, \end{aligned}$$

which is (8). We now have to prove (9). For each $i \in \{0, 1, \dots, k-2\}$ such that (x_k, x_i) is an edge of H we can define a new path (see Figure 7.1)

$$P' = \text{ROTATE}(P; x_k, x_i) = (x_0, x_1, \dots, x_i, x_k, x_{k-1}, \dots, x_{i+1}).$$

We say that P' is obtained from P by a *rotation* with x_0 as the fixed endpoint. Note that P' is also of length k and so $x_{i+1} \in X$. Let $\text{rot}(X) \subseteq X$ be the set of endpoints obtainable by doing a sequence of rotations, starting at P , and always using x_0 as the fixed endpoint. For $S \subseteq V(H)$ let

$$N_H(S) = \{w \in V(H) \setminus S : \exists v \in S \text{ such that } (v, w) \in E(H)\}.$$

The following lemma is perhaps the key idea behind Posá's result.

Lemma 2

$$|N_H(\text{rot}(X))| < 2|\text{rot}(X)|.$$

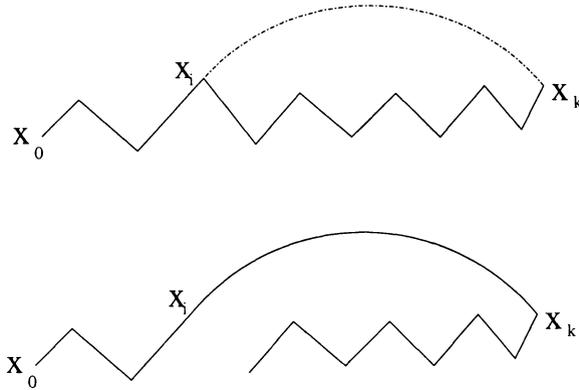


Figure 7.1. A rotation with x_0 as fixed end point.

Proof: We show that if $y \in N_H(\text{rot}(X)) \setminus \{x_0\}$ then there exists $z \in \text{rot}(X)$ such that (y, z) is an edge of P . The lemma follows immediately.

Since $y \in N_H(\text{rot}(X))$ there exists $x \in \text{rot}(X)$ such that $(x, y) \in E(H)$. So there exists a path Q of length k from x_0 to x with y as an internal vertex, obtainable from P by a sequence of rotations. Let z_1 be the neighbor of y on Q between y and x . Then $z_1 \in \text{rot}(X)$ because it is the endpoint of $\text{ROTATE}(Q; x, y)$. So if (y, z_1) is an edge of P we are done. If (y, z) is not an edge of P then one of the edges $e = (y, z_2)$ incident with y in P has been deleted in the sequence of rotations which produced Q . But when an edge e is deleted by a rotation, one of its endpoints is placed in $\text{rot}(X)$. Since $y \notin \text{rot}(X)$ we have $z_2 \in \text{rot}(X)$ and the lemma follows. ■

Now for a calculation:

$$\begin{aligned}
 \mathbb{P}[\exists S, 1 \leq |S| \leq \frac{n}{4}, |N_H(S)| < 2|S|] &\leq \sum_{s=1}^{n/4} \binom{n-1}{s} \binom{n-1}{2s} (1-p)^{s(n-3s)} \\
 &\leq \sum_{s=1}^{n/4} n^s n^{2s} n^{-Ks/4} \\
 &\leq 2n^{3-K/4}.
 \end{aligned}$$

So inequality (9) follows from this and Lemma 2. Applying (7) we see that $G_{n,p}$ has a Hamiltonian path whp.

To finish the proof consider the graph $\Gamma = G_{n,p_1=5p/6} \cup G_{n,p_2=p/6}$. Each of the N possible edges appears independently with probability

$p_1 + p_2 - p_1p_2 < p$ and so

$$\mathbb{P}[\Gamma \text{ is Hamiltonian}] \leq \mathbb{P}[G_{n,p} \text{ is Hamiltonian}] \tag{10}$$

Now by the previous analysis, whp G_{n,p_1} has a Hamiltonian path, P say. Fix one of its endpoints x_0 and let X be the set of endpoints of Hamiltonian paths with one endpoint x_0 . By the above analysis, $|X| \geq \frac{n}{4}$ whp. Now given X , the set of edges of G_{n,p_2} which join x_0 and X is independent of G_{n,p_1} and if there is one, Γ is Hamiltonian. So

$$\begin{aligned} \mathbb{P}[\Gamma \text{ is not Hamiltonian}] &\leq \\ \mathbb{P}[G_{n,p_1} \text{ has no Hamilton path}] &+ \mathbb{P}[|X| \leq \frac{n}{4}] + (1 - p_2)^{n/4} = o(1). \end{aligned}$$

This together with (10) proves Theorem 1. ■

Now a graph with a vertex of degree 0 or 1 is not Hamiltonian. Let $m = \frac{n}{2}(\log n + \log \log n + c_n)$. Erdős and Rényi [279] showed that

$$\lim_{n \rightarrow \infty} \mathbb{P}[\delta(G_{n,m}) \geq 2] = \begin{cases} 0 & c_n \rightarrow -\infty \\ e^{-e^{-c}} & c_n \rightarrow c \\ 1 & c_n \rightarrow +\infty \end{cases} \tag{11}$$

(There is no mystery to the right hand side of the above when $c_n = c$. The expected number of vertices of degree 0 or 1 is $\approx e^{-c}$ and the distribution of this number is asymptotically Poisson.)

The above provides a lower bound for the asymptotic probability of a graph being Hamiltonian. Komlós and Szemerédi [512] proved that this was tight, essentially saying that the best constant in Theorem 1 is $K = 1$.

Theorem 3

$$\lim_{n \rightarrow \infty} \mathbb{P}[G_{n,m} \text{ is Hamiltonian}] = \lim_{n \rightarrow \infty} \mathbb{P}[\delta(G_{n,m}) \geq 2].$$

2.2.2 Graph Process. There is an alternative stronger version of Theorem 3 which is quite remarkable at first sight. The graph process is a random sequence $G_0, G_1, \dots, G_i = ([n], E_i), \dots, G_N$ where $E_i = \{e_1, e_2, \dots, e_i\}$ and e_i is chosen randomly from \bar{E}_{i-1} . Thus starting from the empty graph G_0 , we randomly add new edges until we have a complete graph. Note the G_m has the same distribution as $G_{n,m}$.

For a graph property \mathcal{G} , the *hitting time* $\tau(\mathcal{G})$ is given by

$$\tau(\mathcal{G}) = \min\{i : G_i \in \mathcal{G}\}.$$

Let $\mathcal{H} = \{G \text{ is Hamiltonian}\}$ and $\mathcal{D}_k = \{\delta(G) \geq k\}$. Clearly $\tau(\mathcal{H}) \geq \tau(\mathcal{D}_2)$. Bollobás [123] and Ajtai, Komlós and Szemerédi [10] showed

Theorem 4

$$\tau(\mathcal{H}) = \tau(\mathcal{D}_2) \quad \text{whp.}$$

In other words, if we randomly append edges one by one, whp the first edge that raises the minimum degree to 2, also makes the graph Hamiltonian.

Other properties

The results of Theorems 3 and 4 have been generalized in a number of ways. We now know, for example, that the following statements hold whp:

- $G_{\tau(\mathcal{D}_k)}$ has $\lfloor k/2 \rfloor$ edge disjoint Hamilton cycles plus a further edge disjoint (near) perfect matching if k is odd, $k = O(1)$ [129].
- $G_{\tau(\mathcal{D}_2)}$ has $(\log n)^{n-o(n)}$ distinct Hamiltonian cycles [211].
- $G_{\tau(\mathcal{D}_2)}$ contains k vertex disjoint cycles of size n/k , $k = O(1)$ [330].
- $G_{\tau(\mathcal{D}_2)}$ contains a cycle of every size, $3 \leq k \leq n$, [212, 573, 210]

See also [213] and the surveys [331], [334].

2.2.3 Regular Graphs. In this section we discuss the existence of Hamiltonian cycles in random regular graphs. We use $G_{n,r}$ to denote a graph chosen uniformly at random from the set of r -regular graphs with vertex set $[n]$.

It is easy to show that whp $G_{n,2}$ is a collection of $O(\log n)$ vertex disjoint cycles. Thus $G_{n,3}$ or random cubic graphs is where the real interest starts. There was some success in applying the extension-rotation methods of Section 2.2, [122],[289] and [329]. In the last mentioned paper it was shown that $G_{n,r}$ is Hamiltonian whp for each fixed $r \geq 85$.

A breakthrough came with the work of Robinson and Wormald [726, 727] who used a completely different approach to solve the problem. They proved that $G_{n,r}$ is Hamiltonian whp for each fixed $r \geq 3$.

2.2.4 Model of Random Regular Graphs. We describe the *configuration model* of Bollobás [121]. This is a probabilistic interpretation of a counting formula of Bender and Canfield [99].

Thus let $W = [n] \times [r]$ ($W_v = v \times [r]$ represents r half edges incident with vertex $v \in [n]$.) The elements of W are called *points* and a 2-element subset of W is called a *pairing*. A *configuration* F is a partition of W into $rn/2$ pairings. We associate with F a multigraph $\mu(F) = ([n], E(F))$

where, as a multi-set,

$$E(F) = \{(v, w) : \{(v, i), (w, j)\} \in F \text{ for some } 1 \leq i, j \leq r\}.$$

(Note that $v = w$ is possible here.)

Let Ω denote the set of possible configurations. Thus

$$|\Omega| = \Lambda(rn)$$

where

$$\Lambda(2m) = \frac{(2m)!}{m!2^m}.$$

We say that F is *simple* if the multigraph $\mu(F)$ has no loops or multiple edges. Let Ω_0 denote the set of simple configurations.

We turn Ω into a probability space by giving each element the same probability. We need the following two main properties of this model:

P1 Each $G \in \mathcal{G}(n, r)$ is the image (under μ) of exactly $(r!)^n$ simple configurations.

P2 $\mathbb{P}[F \in \Omega_0] \approx e^{-(r^2-1)/4}$.

(Here $\alpha \approx \beta$ means that $\alpha/\beta \rightarrow 1$ as $n \rightarrow \infty$.)

Suppose now that \mathcal{A}^* is a property of configurations and \mathcal{A} is a property of graphs such that when $F \in \Omega_0, \mu(F) \in \mathcal{A}$ implies $F \in \mathcal{A}^*$. Then P1 and P2 imply

$$\mathbb{P}[G \in \mathcal{A}] \leq (1 + o(1))e^{(r^2-1)/4}\mathbb{P}[F \in \mathcal{A}^*]$$

where G is chosen randomly from \mathcal{G} and F is chosen randomly from Ω . So if r is a constant independent of n then we can use this to see that

$$\mathbb{P}[F \in \mathcal{A}^*] = o(1) \text{ implies } \mathbb{P}[G \in \mathcal{A}] = o(1).$$

2.2.5 The Robinson-Wormald Approach. Suppose we wanted to use Chebyshev's inequality (3) to prove that the existence of a Hamiltonian cycle. For $F \in \Omega$ let

$$Z_H = Z_H(F) = \text{the number of Hamiltonian cycles in } H.$$

A reasonably straightforward calculation shows that

$$\mathbb{E}[Z_H] \approx \sqrt{\frac{\pi}{2n}} \left(\left(\frac{r-2}{r} \right)^{(r-2)/2} (r-1) \right)^n,$$

and a much more difficult calculation [332] gives

$$\mathbb{E}[Z_H^2] \approx \frac{\pi r}{2(r-2)n} \left(\left(\frac{r-2}{r} \right)^{r-2} (r-1)^2 \right)^n,$$

immediately yielding

$$\mathbb{P}[Z_H \neq 0] \geq \frac{\mathbb{E}(Z_H)^2}{\mathbb{E}(Z_H^2)} \approx \frac{r-2}{r}.$$

So when $r = 3$ we see already that $\mathbb{P}[Z_H \neq 0] \geq 1/3$. We need to boost this inequality to 1-o(1).

Let C_l denote the number of l -cycles of $\mu(F)$ for $l \geq 1$. We will be concerned mainly with C_l where l is odd. For $\mathbf{c} = (c_1, c_2, \dots, c_b) \in N^b$, where $N = \{0, 1, 2, \dots\}$, let group $\Omega_{\mathbf{c}} = \{F \in \Omega : C_{2k-1} = c_k, 1 \leq k \leq b\}$. Surprisingly, there is much less variation in the number of Hamiltonian cycles *within* groups, than there is *between* groups. In fact, almost all of the variance can be “explained” by variation between group means.

For $\mathbf{c} \in N^b$ let

$$\pi_{\mathbf{c}} = \mathbb{P}[F \in \Omega_{\mathbf{c}}], \quad \mathbb{E}_{\mathbf{c}} = \mathbb{E}[Z_H \mid F \in \Omega_{\mathbf{c}}] \text{ and } V_{\mathbf{c}} = \text{Var}[Z_H \mid F \in \Omega_{\mathbf{c}}].$$

Then by conditioning on \mathbf{c} we have

$$\mathbb{E}[Z_H^2] = \sum_{\mathbf{c} \in N^b} \pi_{\mathbf{c}} \mathbb{E}[Z_H^2 \mid F \in \Omega_{\mathbf{c}}] = \sum_{\mathbf{c} \in N^b} \pi_{\mathbf{c}} V_{\mathbf{c}} + \sum_{\mathbf{c} \in N^b} \pi_{\mathbf{c}} \mathbb{E}_{\mathbf{c}}^2.$$

Robinson and Wormald [726, 727] prove that

$$\sum_{\mathbf{c} \in N^b} \pi_{\mathbf{c}} V_{\mathbf{c}} \leq \delta \mathbb{E}[Z_H]^2, \tag{12}$$

for some small value δ .

The rest is an application of Chebyshev’s inequality (3). Define the random variable \hat{Z}_H by

$$\hat{Z}_H = \mathbb{E}_{\mathbf{c}}, \text{ if } F \in \Omega_{\mathbf{c}}.$$

Then for any $t > 0$

$$\begin{aligned} \mathbb{P}[|Z_H - \hat{Z}_H| \geq t] &\leq \mathbb{E}[(Z_H - \hat{Z}_H)^2/t^2] \\ &= \sum_{\mathbf{c} \in N^b} \pi_{\mathbf{c}} V_{\mathbf{c}}/t^2 \\ &\leq \delta \mathbb{E}[Z_H]^2/t^2 \end{aligned} \tag{13}$$

where the last inequality follows from inequality (12).

Robinson and Wormald argue that for likely values of \mathbf{c} we find that $\mathbb{E}_{\mathbf{c}} > \epsilon \mathbb{E}(Z_H)$ where δ/ϵ^2 can be chosen arbitrarily small, though fixed as $n \rightarrow \infty$. Putting $t = \epsilon \mathbb{E}(Z_H)$ in (13) gives the required result

$$\mathbb{P}[Z_H = 0] \leq \delta/\epsilon^2 + \theta$$

where θ is the small probability that $G_{n,r} \in \mathbb{E}_{\mathbf{c}}$ where $\mathbb{E}_{\mathbf{c}} \leq \epsilon \mathbb{E}(Z_H)$.

2.3. Polynomial Time Algorithms

Here we turn our attention to algorithms for finding Hamiltonian cycles which work whp.

2.3.1 Sparse Case. Angluin and Valiant [24] described a randomized algorithm which whp finds a Hamiltonian cycle in $G_{n,p}$ provided $p \geq K \log n/n$ for sufficiently large K . Shamir [759] improved this to $p \geq (\log n + (2 + \epsilon) \log \log n)/n$ which is almost best possible.

We discuss the algorithm HAM of Bollobás, Fenner and Frieze [128] which yields a constructive proof of Theorem 3. We work in $G_{n,m}$ where $m = \frac{n}{2}(\log n + \log \log n + c_n)$ and either $c_n \rightarrow c$ or $c_n \rightarrow \infty$. Let $d = 2m/n$ be the average degree of G . We note that G is connected whp and we check for minimum degree at least 2 before running HAM.

Algorithm HAM works in stages. At the start of stage k we have a path P_k of length k . Suppose that its endpoints are w_0, w_1 . We start the algorithm in Stage 0 with $P_0 = \{1\}$. We first grow a tree of paths of length k , all with endpoints w_0 . (There are places in this procedure where we can jump to stage $k + 1$. When k is small we are likely to do this immediately, see below.) The tree is grown in breadth first fashion to a depth T , where $T = \lceil \log n / (\log d - \log \log d) \rceil + 1$. The children of a node P are all those paths which can be obtained from P by a rotation with w_0 as the fixed endpoint.

Let $END(w_0, G)$ denote the set of endpoints, other than w_0 , of the paths produced in this manner. Then for each $x \in END(w_0, G)$ we start with the first path Q_x produced with endpoints w_0, x and grow a tree of depth T , with root Q_x and this time use rotations with x as fixed endpoint. Let $END(x, G)$ denote the set of endpoints, other than x , of the paths produced in this manner.

We may not need to do this much work in a stage. If ever we find a path Q with endpoints x, y such that $(y, z) \in E$ and $z \notin Q$ then clearly we have a path Q, z of length $k + 1$ and we go immediately to the next stage by a *simple extension*. If $(x, y) \in E$ then because G is connected, there is an edge (u, v) joining the cycle $Q + (x, y)$ to the rest of G . We then have a path $Q + (x, y) + (u, v) - (z, u)$ of length $k + 1$ where z is

adjacent to u on Q . This is called a *cycle extension* and we call (x, y) the *closing edge*. If neither of these two possibilities occurs during stage k then HAM fails.

HAM stops successfully in stage $n - 1$ if it manages to close one of the Hamiltonian paths that it creates.

In the event that HAM fails, if

$$END = END(G) = \{w_0\} \cup END(w_0, G)$$

then we know that

$$x \in END, y \in END(x, G) \text{ implies } (x, y) \notin G. \tag{14}$$

We will show that $|END|$ is of order n whp and so if there were no conditioning, this would be unlikely to happen.

Suppose now that HAM fails in stage k . Let $W = W(G)$ denote the edges of the paths $P^{(0)}, P^{(1)}, \dots, P^{(M)} = P_k$ where $P^{(i+1)}$ is obtained from $P^{(i)}$ by a rotation, simple or cycle extension plus the closing edges of the cycle extensions in the sequence. Clearly

$$|W| \leq nT. \tag{15}$$

For $X \subset E$ let $G_X = ([n], E \setminus X)$. The following should be clear.

Lemma 5 *Suppose that HAM terminates unsuccessfully in stage k on input G . If $X \subseteq E \setminus W$ then HAM will terminate unsuccessfully on input G_X . Furthermore, on G_X , HAM will generate P_k at the start of stage k through the same sequence of rotations and extensions.*

A vertex of G is *small* if its degree is at most $d/20$ and *large* otherwise. The following lemma is proved in [128].

Lemma 6 *Assume $c_n \not\rightarrow \infty$. Then*

- (a) *G contains no more than $n^{1/2}$ small vertices.*
- (b) *G does not contain 2 small vertices at a distance of 4 or less apart.*
- (c) *G contains no vertex of degree $5d$ or more.*
- (d) *There does not exist a set of large vertices S such that $|S| \leq n/d$ and $|N_G(S)| \leq d|S|/300$.*

Now let $\mathcal{G}_0 = \mathcal{G}_{n,m}$. Let $\mathcal{G}_1 = \{G : G \text{ is connected, has minimum degree at least } 2 \text{ and satisfies the conditions of Lemma 6}\}$. Then, by (11),

$$|\mathcal{G}_1| \approx e^{-e^{-c_n}} \binom{N}{m}. \tag{16}$$

Note that the running time of HAM on a member of \mathcal{G}_1 is

$$O(n^2(5d)^T \log n) = O(n^{3+o(1)})$$

assuming [24] that we can do a rotation in $O(\log n)$ time. We can make HAM always run in polynomial time by not trying on graphs with maximum degree $\geq 5d$. Note also that HAM is a *deterministic* algorithm.

Suppose that HAM terminates unsuccessfully in stage k on input G . Let $X \subseteq E$ be *deletable* if the following three properties hold:

- 1 No edge of X is incident with a small vertex,
- 2 No large vertex meets more than $d/1000$ edges of X , and
- 3 $X \cap W = \emptyset$.

The following lemma is proved in [128].

Lemma 7 *Suppose that HAM terminates unsuccessfully in stage k on input $G \in \mathcal{G}_1$. Suppose $X \subseteq E$ is deletable. Then for large n ,*

$$|END(G_X)| \geq n/1000 \tag{17}$$

$$|END(x, G_X)| \geq n/1000 \quad \text{for } x \in END(G_X). \tag{18}$$

(Idea of proof: Let S_t denote the set of large endpoints of paths at depth t in a tree. The conditions in Lemma 6 imply that $|S_t| \leq n/d$ implies $|S_{t+1}| \geq d|S_t|/1000$. This is basically because $|S_{t+1}| \geq (\frac{1}{2} - o(1))|N(S_t)|$. The $o(1)$ accounts for small neighbors and the $\frac{1}{2}$ comes from two neighbors giving the same new endpoint.)

Now let $\mathcal{G}_2 = \{G : G \in \mathcal{G}_1 \text{ and HAM terminates unsuccessfully on } G\}$. We use a ‘‘coloring’’ argument of Fenner and Frieze [289] to prove

$$\frac{|\mathcal{G}_2|}{|\mathcal{G}_0|} \rightarrow 0 \text{ as } n \rightarrow \infty. \tag{19}$$

Combining (16), our assumption on c_n , and (19) we obtain:

Theorem 8

$$\lim_{n \rightarrow \infty} \mathbb{P}[\text{HAM finds a Hamilton cycle}] = \lim_{n \rightarrow \infty} \mathbb{P}[G_{n,m} \text{ is Hamiltonian}].$$

Proof of (19): Let $\omega = \lceil \lambda d \rceil$ for some (arbitrary) positive constant λ . For $G \in \mathcal{G}_1$ let $(G, j), j = 1, 2, \dots, \nu = \binom{m}{\omega}$ enumerate all the possible ways of choosing ω edges of G and coloring them green and the remaining $m - \omega$ edges blue. Let $X = X(G, j)$ denote the set of green edges. Let $a(G, j) = 1$ if HAM terminates unsuccessfully on G and G_X and

there does not exist $e = (x, y) \in X$ such that $x \in \text{END}(G_X)$ and $y \in \text{END}(x, G_X)$ and

$$\min\{|\text{END}(G_X)|, |\text{END}(x, G_X)|\} \geq n/1000$$

for all $x \in \text{END}(G_X)$. Otherwise, $a(G, j) = 0$.

We show next that for $G \in \mathcal{G}_2$

$$\sum_{j=1}^{\nu} a(G, j) \geq (1 - o(1)) \binom{m - nT}{\omega}. \tag{20}$$

To see this let $G \in \mathcal{G}_2$ and let HAM terminate unsuccessfully in stage k on G . It follows from (14) and Lemmas 5 and 7 that if $X = X(G, j)$ is deletable then $a(G, j) = 1$. Let $G' = (V', E')$ be the subgraph of G induced by the large vertices and the edges not in $W(G)$. Then $|V'| \geq n - n^{1/2}$ and $|E'| \geq m - nT$. The number of deletable sets is the number of ways of choosing ω edges from E' subject to the condition that no vertex in V' is incident with more than $d/1000$ edges. Almost all choices of ω edges satisfy this and (20) follows.

On the other hand let H be a fixed graph with vertex set $[n]$ and $m - \omega$ edges. Let $b(H) = |\{(G, j) : H = G_X, G \in \mathcal{G}_1 \text{ and } a(G, j) = 1\}|$. Then

$$b(H) \leq \binom{N' - m + \omega}{\omega} \text{ where } N' = N - \binom{\lceil n/1000 \rceil}{2}. \tag{21}$$

Then, by (20),

$$\begin{aligned} (1 - o(1)) \binom{m - nT}{\omega} |\mathcal{G}_2| &\leq \sum_{G \in \mathcal{G}_2} \sum_{j=1}^{\nu} a(G, j) \\ &\leq \sum_{G \in \mathcal{G}_0} \sum_{j=1}^{\nu} a(G, j) = \sum_H b(H) \\ &\leq \binom{N' - m + \omega}{\omega} \binom{N}{m - \omega} \end{aligned}$$

on using (21).

It follows that

$$\frac{|\mathcal{G}_2|}{|\mathcal{G}_0|} \leq (1 + o(1)) \frac{\binom{N' - m + \omega}{\omega} \binom{N}{m - \omega}}{\binom{m_1}{\omega} \binom{N}{\omega}} \leq e^{-\lambda d/1000001},$$

which proves (19).

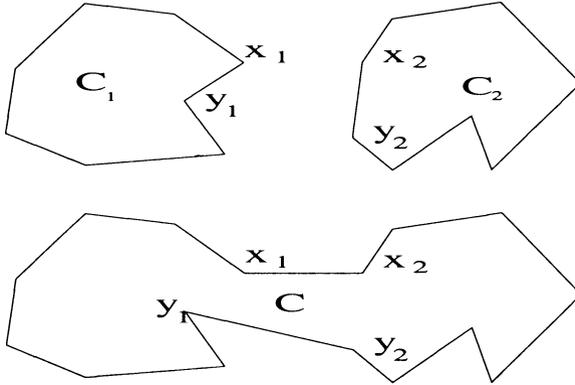


Figure 7.2. Patching C_1 and C_2 together

2.3.2 Dense Case. Algorithm HAM was designed to work at the threshold for the existence of Hamiltonian cycles. At the other end of the scale we have the dense case $G_{n,p}$, p constant, where our graph has $\Omega(n^2)$ edges whp. For the case $p = 1/2$ (i.e. each graph equally likely), one can show that HAM fails with probability $o(2^{-n})$ and if in the rare occasions where it fails to find a Hamiltonian cycle we use the $O(n^2 2^n)$ dynamic programming algorithm of Held and Karp [444] then we have a deterministic algorithm which (i) runs in polynomial expected time and (ii) *always* determines whether or not a graph has a Hamiltonian cycle.

Gurevich and Shelah [412] and Thomason [791] constructed randomized algorithms which run in $O(n^2)$ expected time which is *linear* in the number of edges whp.

It is in fact quite easy to construct an algorithm which finds a Hamiltonian cycle in $G_{n,1/2}$ whp. It is more difficult to make the failure probability small enough to make it run in polynomial expected time. Here is a very simple algorithm for the former case. It is based on the idea of *patching* cycles. Given two cycles C_1, C_2 we look for edges $e_i = (x_i, y_i)$, $i = 1, 2$ such that G contains the edges $f_1 = (x_1, x_2)$ and $f_2 = (y_1, y_2)$. We then create a new cycle $C = C_1 + C_2 + f_1 + f_2 - e_1 - e_2$ which covers the vertices of C_1 and C_2 – see Figure 7.2.

Patching Algorithm

- (a) Divide $[n]$ into $s = \lfloor n/\omega \rfloor$ sets X_1, X_2, \dots, X_s of size ω or $\omega + 1$ where $\omega = \lceil 2 \log_2 n \rceil$.

- (b) Use the algorithm of [444] to find a Hamiltonian cycle C_i through the vertices of each X_i , $i = 1, 2, \dots, s$.
Step (b) takes $O(n\omega 2^\omega) = O(n^3 \log n)$ time and it fails with probability $O(2^{-\omega n}) = o(1)$.
- (c) For $i = 1, 2, \dots, s$ divide each cycle into 2 paths $P_{i,1}, P_{i,2}$ of lengths $\approx \omega/2$ and then for $i = 1, 2, \dots, s - 1$ try to patch C_i into C_{i+1} using edges from $P_{i,2}$ and $P_{i+1,1}$ which do not contain endpoints of the paths (to avoid dependencies).
The probability that Step (c) fails is $O(s2^{-\Omega(\omega^2)}) = o(1)$.

The method is easily parallelizable and a more complicated parallel version runs in $O(\log \log n)^2$ time [327]. Mackenzie and Stout [574] and Van Wieren and Stout [808] found $O(\log^* n)$ parallel expected time algorithms. Parallel algorithms for random graphs with $Kn \log n$ edges were considered in Coppersmith, Raghavan, and Tompa [217].

2.3.3 Regular Graphs. In this section we consider the problem of finding a Hamiltonian cycle in a random regular graph. The paper [329] provides an $O(n^{3+o(1)})$ time extension-rotation algorithm for finding a Hamiltonian cycle in $G_{n,r}$ whp provided $r \geq 85$, r constant. Frieze, Jerrum, Molloy, Robinson and Wormald [332] used a completely different approach for $r \geq 3$, r constant.

The idea is very simple: Use the algorithm of Jerrum and Sinclair [457] to generate a (near) random 2-factor of $G_{n,r}$. The paper [329] then argues that whp the number of Hamiltonian cycles in $G_{n,r}$ is at least a proportion $1/(2n^{5/2})$ of the number of 2-factors. Thus after $O(n^{5/2} \log n)$ applications of the Jerrum-Sinclair algorithm we will whp produce a Hamiltonian cycle.

A similar idea was used in Frieze and Suen [337] for finding Hamiltonian cycles in a random digraph with m edges, $m/n^{3/2} \rightarrow \infty$.

What happens if $r \rightarrow \infty$ with n . The above results only hold for r constant. nevertheless, two recent papers, Cooper, Frieze and Reed [216] and Krivelevich, Sudakov, Vu and Wormald [522] show that random regular graphs are Hamiltonian, $3 \leq r \leq n - 1$.

2.3.4 Digraphs. Analogously to $G_{n,m}$ the random digraph $D_{n,m}$ has vertex set $[n]$ and m random edges. The model $D_{n,p}$ is defined similarly. A natural conjecture would be that $D_{n,m}$ is Hamiltonian whp when it has enough edges to ensure that both the minimum in-degree and out-degree are both at least one whp. This was proved in Frieze [328]. Let $m = n(\log n + c_n)$. Then

Theorem 9

$$\lim_{n \rightarrow \infty} \mathbb{P}[D_{n,m} \text{ is Hamiltonian}] = \begin{cases} 0 & c_n \rightarrow -\infty \\ \epsilon^{-2e^{-c}} & c_n \rightarrow c \\ 1 & c_n \rightarrow +\infty \end{cases}$$

This was proved algorithmically. The cycle is found whp by an $O(n^{3/2})$ time algorithm. In Section 3.2 we will discuss a result, in some detail, which implies Theorem 9.

Earlier, Angluin and Valiant [24] had given an $O(n(\log n)^2)$ time algorithm which works whp for $m \geq Kn \log n$, K sufficiently large. McDiarmid [589] proved the interesting inequality

$$\mathbb{P}[D_{n,p} \text{ is Hamiltonian}] \geq \mathbb{P}[G_{n,p} \text{ is Hamiltonian}]. \tag{22}$$

So putting $p = (\log n + \log \log n + \omega)/n$, $\omega \rightarrow \infty$ (22) and Theorem 3 implies that $D_{n,p}$ is Hamiltonian whp. This is not quite as strong as Theorem 9.

2.4. Other Models

We briefly consider some results pertaining to the existence of Hamiltonians in other models of a random graph.

Random regular digraphs. Let $D_{n,r}$ be chosen uniformly from the set of digraphs with vertex set $[n]$ in which each vertex has in-degree and out-degree r .

Theorem 10 [215] *Assume r is constant independent of n . Then*

$$\lim_{n \rightarrow \infty} \mathbb{P}[D_{n,r} \text{ is Hamiltonian}] = \begin{cases} 0 & r \leq 2 \\ 1 & r \geq 3 \end{cases}$$

k -out model. In this model G_{k-out} the vertex set is $[n]$ and then each vertex $v \in [n]$ independently chooses k neighbors. This is a graph, not a digraph, the average degree is $2k$ and multiple edges are possible. What is known is summarized in the following. The case $k = 3$ is an important open question.

Theorem 11 [214]

$$\lim_{n \rightarrow \infty} \mathbb{P}[G_{k-out} \text{ is Hamiltonian}] = \begin{cases} 0 & k \leq 2 \\ 1 & k \geq 4 \end{cases}$$

k -in, k -out model. In this model $D_{k-in,k-out}$ the vertex set is $[n]$ and then each vertex $v \in [n]$ independently chooses k in-neighbors and k out-neighbors.

Theorem 12 [214]

$$\lim_{n \rightarrow \infty} \mathbb{P}[D_{k-in, k-out} \text{ is Hamiltonian}] = \begin{cases} 0 & k = 1 \\ 1 & k \geq 2 \end{cases}$$

Hidden Hamiltonian cycles. In this model we start with a cycle of size n and add either (i) cn random edges to it, c constant or (ii) a random perfect matching. This has some cryptographic significance in relation to authentication schemes [138]. [138] shows that in case (i), if c is sufficiently large then one can find a Hamiltonian cycle whp in polynomial time. This may not be the original cycle, but it nevertheless kills the authentication scheme. Similarly, [332] shows that whp we can find a Hamiltonian cycle in case (ii). See Wormald [828] for a recent survey which explains the relation between graphs generated as in (ii) and random regular graphs.

This ends our discussion of the Hamiltonian cycle problem in random graphs. We turn to the Traveling Salesman Problem (TSP).

3. Traveling Salesman Problem: Independent Model

We consider problems where the coefficients of the cost matrix $C = [c_{i,j}]$ are either completely independent (asymmetric model) or constrained by $c_{i,j} = c_{j,i}$ (symmetric model). Sections 3.1, 3.2 consider exact solutions and then Section 3.3 considers approximate solutions. Section 3.4 considers an enumerative exact algorithm for the asymmetric case.

3.1. Symmetric Case: Exact Solution

Let us first consider a symmetric model in which the coefficients $c_{i,j}$ are random integers in the range $[0..B]$. Frieze [326] described an algorithm which finds an exact solution whp provided $B = o(n/\log \log n)$.

The strategy of the algorithm is to first find a set X_0 of *troublesome* vertices and then to find a set of vertex disjoint paths $\mathcal{P} = \{P_1, \dots, P_t\}$ which cover X_0 as cheaply as possible. By cover we mean that each vertex of X_0 is an interior vertex of one of the paths of \mathcal{P} . We choose \mathcal{P} to minimize $c(\mathcal{P}) = \sum_{i=1}^t c(P_i)$ where $c(P_i)$ is the weight of the edges of P_i .

Having found \mathcal{P} we then find a Hamiltonian cycle H which contains the paths of \mathcal{P} as sub-paths and which otherwise only contains zero length edges. It is easy to see that H solves the associated traveling salesman problem.

For $B = o(n/(\log n)^{1/2})$ it suffices to take $X_0 = \{v \in [n] : d_0(v) \leq d/2\}$ where $d_0(v)$ is the number of zero length edges incident with v and $d = n/B$ is (close to) the expected number of zero length edges incident with a vertex. So in the construction of H we essentially have to find a Hamiltonian cycle in a random graph with minimum degree $\geq d/2$. A modification of the algorithm HAM of Section 2.3.1 will suffice. For larger B we have to augment X_0 by other vertices. There are many details best left to the interested reader of [326].

3.2. Asymmetric Case: Exact Solution

In this section we consider randomly generated asymmetric problems. In particular we discuss a result of Frieze, Karp and Reed [333]. The main result of [333] is the following: For an $n \times n$ matrix C let $AP(C)$ be the minimum value for the assignment problem with matrix C and let $ATSP(C)$ be the minimum cost of a traveling salesman tour with costs C . We always have $AP(C) \leq ATSP(C)$. The following provides sufficient conditions for equality whp.

Theorem 13 *Let $\{X_n\}$ be a sequence of random variables over the non-negative reals. Let $p = p_n = \mathbb{P}[X_n = 0]$ and let $\omega = \omega(n) = np$. Let $C = C(n)$ be an $n \times n$ matrix whose entries are drawn independently from the same distribution as X_n . If $\omega \rightarrow \infty$ as $n \rightarrow \infty$ then $AP(C) = ATSP(C)$ whp.*

We see easily that this implies Theorem 9 for the case $c_n \rightarrow +\infty$. Let $X_n = 0$ or 1 . Now $np = \log n + c_n \rightarrow \infty$ and it is known from Erdős and Rényi [280] that $AP(C) = 0$ whp. Thus $ATSP(C) = 0$ whp i.e. whp there is a Hamiltonian cycle in the graph induced by zero length edges.

Let H be the weighted bipartite graph with vertex set $X \cup Y$, where $X = \{x_1, x_2, \dots, x_n\}$ and $Y = \{y_1, y_2, \dots, y_n\}$, and with an edge of weight $c_{i,j}$ between x_i and y_j . Let D be the complete digraph on vertex set $[n]$, in which each edge (i, j) has weight $c_{i,j}$. A cycle cover is a subgraph of D in which each of the n vertices has in-degree 1 and out-degree 1. The Assignment Problem can be stated in any of the following equivalent forms:

- Find a perfect matching of minimum weight in H .
- Find a cycle cover of minimum weight in D .
- Find a permutation σ (of $[n]$) to minimize $\sum_{i=1}^n c_{i,\sigma(i)}$.

Let the indicator variable $z_{i,j}$ be 1 if $c_{i,j} = 0$ and 0 otherwise. Then the $z_{i,j}$ are independent, and each $z_{i,j}$ is equal to 1 with probability

p . Emulating a useful trick due to Walkup [817] we view the $z_{i,j}$ as being generated in the following way. Let $h = h(n)$ be defined by the equation $1 - p = (1 - h)^5$ and let $z_{i,j}^k$, for $i = 1, 2, \dots, n$, $j = 1, 2, \dots, n$ and $k = 1, 2, 3, 4, 5$, be independent indicator variables, each of which is equal to 1 with probability h . Let $z_{i,j} = \max_{k=1}^5 z_{i,j}^k$. Then the $z_{i,j}$ are independent, and each is equal to 1 with probability p . For $k = 1, 2$, let H_k be the bipartite graph with vertex set $X \cup Y$, and with an edge between x_i and y_j if and only if $z_{i,j}^k = 1$. For $k = 3, 4, 5$, let D_k be the digraph with vertex set $[n]$ and an edge from i to j if and only if $z_{i,j}^k = 1$. The edges of D_3 , D_4 and D_5 , respectively, will be called *out-edges*, *in-edges* and *patch edges*. Each type of edge will play a special role in the construction of a Hamiltonian circuit of weight $AP(C)$. It will be important that the random graphs H_1 and H_2 , and the random digraphs D_3 , D_4 and D_5 , are completely independent. Also, let $s = s(n) = nh$; s is the expected degree of a vertex in H_1 or H_2 , and the expected out-degree of a vertex in D_3 , D_4 or D_5 . Clearly, $s \geq \omega/5$, and thus s tends to infinity if ω does.

The construction of the desired Hamiltonian circuit proceeds in the following stages:

- (a) (Identification of *troublesome vertices*). By considering the edges of $H_1 \cup H_2$ identify a set $A \subset X$ and a set $B \subset Y$, $|A| = |B|$. The cardinality of $A \cup B$ is small whp.

The set $A \cup B$ contains the vertices of exceptionally small degree plus certain other vertices that are likely to be incident with edges of nonzero weight in an optimal assignment. At the same time construct a matching in H which is of minimum weight, subject to the condition that it covers the vertices in $A \cup B$ and no other vertices. (Compare with the algorithm of the previous section.)

- (b) Consider the subgraph of $H_1 \cup H_2$ induced by $(X \setminus A) \cup (Y \setminus B)$. This bipartite graph has a perfect matching whp. Combining that perfect matching with the matching constructed in the previous step, obtain an optimal assignment for H in which every non-zero-weight edge is incident with a vertex in $A \cup B$.
- (c) The optimal assignment just constructed has the properties of a random permutation. In particular it has $O(\log n)$ cycles.
- (d) Using the out-edges and in-edges, attempt to convert the original optimal assignment into a permutation with no short cycles. This process succeeds whp.

- (e) Using the patch edges, patch the long cycles together into a single cycle, thus solving the *ATSP*, much as in the patching algorithm of Section 2.3.2. The patching process succeeds whp.

The overall strategy of the proof is to construct an optimal assignment while keeping the in-edges, out-edges and patch edges (except those incident with $A \cup B$) in reserve for use in converting the optimal assignment to a tour without increasing cost.

In this summary we focus on Step (d). The reason we want to get large cycles is that if we have two cycles C_1 and C_2 , then we get $\Omega(|C_1||C_2|)$ opportunities to create patches and we need to be sure that this is significantly greater than the inverse of the probability s^2/n^2 of making a single patch in D_5 . So if C_1 and C_2 are large then $|C_1||C_2| \geq n^2/\omega$ and this is large compared with n^2/s^2 .

3.2.1 Elimination of Small Cycles. Call a cycle in a permutation *small* if it contains fewer than $n/\sqrt{\omega}$ vertices. We now discuss how the out-edges and in-edges are used to convert the original optimal assignment into an optimal assignment in which no cycle is small. Our procedure is to take each small cycle of the original optimal assignment σ in turn and try to remove it without creating any new small cycles.

We now describe the *rotation-closure algorithm* that is used to eliminate one small cycle. Let C be a small cycle. We make a number of separate attempts to remove C . The i th attempt consists of an Out Phase and an In Phase. We will ignore some technical points which are necessary to maintain some independence.

The Out Phase

Define a *near-cycle-cover* as a digraph θ consisting of a directed path P_θ plus a set of vertex-disjoint directed cycles covering the vertices not in P_θ . We obtain an initial near-cycle-cover by deleting an edge of C from the current (optimal) assignment, thus converting the small cycle C into a path. We then attempt to obtain many near-cycle-covers by a rotation process. The state of this process is described by a rooted tree whose nodes are near-cycle-covers, with the original near-cycle-cover at the root. (Compare with HAM of Section 2.3.1.) Consider a typical node θ consisting of a path P_θ directed from a_θ to b_θ plus a cycle cover of the remaining vertices. We obtain descendants of θ by looking at out-edges directed from b_θ . Consider an edge that is directed from b_θ to a vertex y with predecessor x . Such an edge is *successful* if either y lies on a large cycle or y lies on P_θ and the sub-paths of P_θ from a_θ to x and from y to b_θ are both of length at least $n/\sqrt{\omega}$. In such cases a descendant of θ is created by deleting (x, y) and inserting (b_θ, y) . The

tree of near-cycle-covers is grown in a breadth-first manner until the number of leaves reaches $m = \sqrt{n \ln n}$.

Assuming that the number of vertices on short cycles is less than $n/\omega^{1/3}$ (this is true whp) and ignoring some technicalities, the number of descendants of node θ is a binomial random variable $B(n - o(n), s/n)$, and the random variables associated with distinct nodes are independent. Suppose that level t of the rooted tree describing the Out Phase has a vertices. Then, applying the Chernoff bound (4) on the tails of the binomial, the number of nodes at level $t + 1$ lies between $as/2$ and $2as$, with probability greater than or equal to $1 - e^{-as/10}$. Hence the probability that the Out Phase fails to produce m leaves is (quite conservatively) at most $\sum_{k=1}^{\infty} e^{-ks/10} \leq e^{-s/20}$.

The In Phase

The tree produced by an Out Phase has m terminal nodes. Each of these is a near-cycle-cover in which the directed path begins at the same vertex v . Let the j th terminal node be denoted G_j , and let the directed path in G_j run from v to x_j . During the In Phase we grow rooted trees independently from all the G_j , $j = 1, 2, \dots, m$. The process is like the Out Phase, except that, in computing the descendants of a node θ , we fan backwards along in-edges, rather than forwards along out-edges.

If all goes according to plan, we end the In Phase with at least $m^2/2$ near-cycle-covers. The conditional probability that we cannot close any of the paths of these covers into large cycles is at most $(1 - s/n)^{m^2/2} \leq n^{-s/2}$. Closing a path creates a new optimal assignment with at least one less small cycle.

3.3. Asymmetric Case: Approximation Algorithms

An early Russian paper by Gimadi and Perepelitsa [366] analyzes the “nearest neighbor” algorithm when the cost $c_{i,j} = a_n + \xi_{i,j}(b_n - a_n)$, $b_n \geq a_n$. In the nearest neighbor algorithm, at the k th stage we have a path P_k of length k and if $k < n - 1$ this is increased to length $k + 1$ by adding the shortest edge leaving P_k . When $k = n - 1$ there is a unique completion to a tour. It is assumed that the $\xi_{i,j}$ are drawn independently from some common continuous distribution function F over $[0, 1]$. Assuming $s_n = \int_{t_n}^1 \frac{dx}{F(x)} \rightarrow \infty$ with n , $F(t_n) = \frac{1}{n}$, they use the Chebyshev inequality to show whp the length Z of the tour produced by the greedy heuristic is at most $na_n + (b_n - a_n)(nt_n + s_n)$. In particular if $F(x) = x$, i.e. the $\xi_{i,j}$ are uniform $[0, 1]$ then $t_n = \frac{1}{n}$ and $s_n = \log n$. Since the optimum tour has length at least na_n we see that if $\frac{b_n}{a_n} = o(\frac{n}{\log n})$ then whp the length

of the greedy tour produced is within $1 + o(1)$ of the minimum i.e. it is *asymptotically optimal*. This is interesting because it analyzes a simple algorithm that has been considered in other contexts e.g. worst-case analysis. It would be of some interest to consider other such algorithms.

This analysis shows that in the case when the $c_{i,j}$ are independent uniform $[0,1]$ i.e. $a_n = 0$ and $b_n = 1$, the tour generated by this greedy algorithm is close to $\log n$. This is not good as it is known that the minimum tour length is at most 2 whp. We must look for a more sophisticated heuristic to make progress in this important case. In this case, Karp [498] proved the rather surprising fact that

$$ATSP(C) = AP(C) + o(1) = 1 - o(1) \quad \text{whp.} \quad (23)$$

He constructed an $O(n^3)$ patching algorithm to do this. We see then that Karp's algorithm is asymptotically optimal. (Note that the lower bound in (23) has been increased to more than 1.5 and it is conjectured that $AP(C) \approx \frac{\pi^2}{6}$ whp.) Later Karp and Steele [500] improved this to

$$ATSP(C) = AP(C) + O(n^{-1/2}) \quad \text{whp}$$

and Dyer and Frieze [262] made further improvements showing

$$ATSP(C) = AP(C) + O((\log n)^4 / (n \log \log n)).$$

More recently, Frieze and Sorkin [336] have made a small improvement, showing

$$ATSP(C) - AP(C) \leq c_1 \frac{(\log n)^2}{n} \quad \text{whp} \quad (24)$$

$$\mathbb{E}(ATSP(C) - AP(C)) \geq \frac{c_2}{n} \quad (25)$$

for some constants $c_1, c_2 > 0$.

We now outline the proof of (24). It was shown in [336] that whp the optimum assignment solution (cycle cover) only has edges of length $\leq c_2 \frac{\log n}{n}$ whp for some constant $c_2 > 0$. We let an edge be colored

Red: $c(i, j) \in [0, c_2 \frac{\log n}{n}]$; Blue : $c(i, j) \in [c_2 \frac{\log n}{n}, 2c_2 \frac{\log n}{n}]$;

Green: $c(i, j) \in [2c_2 \frac{\log n}{n}, 3c_2 \frac{\log n}{n}]$; Black otherwise.

Then whp the optimum assignment solution (cycle cover) \mathcal{C} will consist entirely of Red edges. By symmetry it will have the cycle structure of a random cycle cover, i.e. have $\leq 2 \log n$ cycles whp.

Let a cycle of \mathcal{C} be small if its length is $\leq n_0 = n \log \log n / \log n$. We use the rotation-closure algorithm of the previous section to remove small cycles. To grow the trees we are only allowed to use edges (i, j)

which either one of the K shortest out of i or one of the K shortest into j for some sufficiently large $K > 0$. One can show that whp the trees grow at a rate of at least $K - 3$ per level and so they only need to be grown to a depth of $O(\log n)$. The expected length of these edges is $O(1/n)$. When we have grown enough paths, we can whp close one using a Blue edge. Thus whp it costs $O(\frac{\log n}{n})$ to remove a small cycle, the cost of the added Red/Blue edges. There are $O(\log n)$ cycles altogether and so it costs $O(\frac{(\log n)^2}{n})$ to make all cycles of length at least n_0 .

We can then whp patch all these cycles into one at the extra cost of $o(\log n \times (\log n/n))$ proving (24).

3.4. Asymmetric Case: Enumeration Algorithm

The AP can be expressed as a *linear program*:

$$\begin{aligned} \text{LP : Minimize } & \sum_{i,j} c(i, j)x_{i,j} \text{ subject to} \\ & \sum_i x_{i,k} = \sum_j x_{k,j} = 1, \forall k, 0 \leq x_{i,j} \leq 1, \forall i, j. \end{aligned}$$

This has the dual

$$\text{DLP : Maximize } \sum_i u_i + \sum_j v_j \text{ subject to } u_i + v_j \leq c(i, j), \forall i, j.$$

Suppose now that we condition on an optimal basis for LP and insist that $u_1 = 0$. It follows that the remaining dual variables are uniquely determined, with probability 1 – they satisfy $2n - 1$ linear equations. Furthermore the reduced costs $\bar{c}(i, j) = c(i, j) - u_i - v_j$ of the *non-basic* variables N are independently and uniformly distributed in the interval $[\max\{0, u_i + v_j\}, 1 - u_i - v_j]$. (Within this interval $c(i, j) \in [0, 1]$ and $\bar{c}(i, j) \geq 0$.) It is shown in [336] that whp

$$\max_{i,j} \{|u_i|, |v_j|\} = O\left(\frac{\log n}{n}\right). \tag{26}$$

Let I_k denote the interval $[2^{-k}c_1\frac{(\log n)^2}{n}, 2^{-(k-1)}c_1\frac{(\log n)^2}{n}]$ for $k \geq 1$. It follows from (26) and the distribution of the reduced costs $\bar{c}(i, j)$ of the non-basic variables, that whp (i) there are $\leq c_1 2^{-(k-1)}n \log n$ non-basic variables $x_{i,j}$ whose reduced cost is in $I_k, 1 \leq k \leq k_0 = \frac{1}{2} \log_2 n$ and (ii) there are $\leq 2c_1 \sqrt{n} \log n$ non-basic variables $x_{i,j}$ whose reduced cost is $\leq c_1 \frac{(\log n)^2}{n^{3/2}}$.

We can search for the optimal solution to ATSP by choosing a set of non-basic variables, setting them to 1 and then resolving the assignment

problem. If we try all sets and choose the best tour we find, then we will clearly solve the problem exactly. However, it follows from (24) that we need only consider sets which contain $\leq 2^k$ variables with reduced costs in I_k and none with reduced cost $\geq c_1 \frac{(\log n)^2}{n}$. Thus whp we need only check at most

$$2^{2c_1\sqrt{n} \log n} \prod_{k=1}^{k_0} \sum_{t=1}^{2^k} \binom{c_1 2^{-(k-1)} n \log n}{t} = e^{\tilde{O}(\sqrt{n})}$$

sets.

We have thus shown

Theorem 14 *ATSP can be solved exactly whp in $e^{\tilde{O}(\sqrt{n})}$ time.*

3.5. Open Problems

Problem 15 *Is G_{3-out} Hamiltonian whp?*

It is unthinkable that the answer is no.

Problem 16 *Is there an extension-rotation algorithmic proof of the fact that random cubic graphs are Hamiltonian?*

Some empirical work suggests that there is such a proof.

Problem 17 *Show that $G_{n,r}$ is Hamiltonian whp in the case $r = r(n) \rightarrow \infty$.*

The paper [129] deals with the case where $m \approx \frac{1}{2}n \log n$.

Problem 18 *Let Q_n denote the n -cube – vertex set $\{0, 1\}^n$ and an edge joining two vectors of Hamming distance 1. Let $Q_{n,p}$ be the random subgraph of Q_n obtained by independently deleting edges with probability $1 - p$. Find the threshold for the existence of a Hamiltonian cycle in $Q_{n,p}$.*

The approximate threshold for connectivity is $p = \frac{1}{2}$ (Erdős and Spencer [281]); $p = \frac{1}{2}$ is also the approximate threshold for the existence of a perfect matching (Bollobás [125]).

Problem 19 *Is there a polynomial expected time algorithm for checking the Hamiltonian property of $G_{n,m}$ for all m ?*

Problem 20 *Suppose $c \geq 3/2$ is constant, $p = c/n$, and $G = G_{n,p}$. Is it true that*

$$\lim_{n \rightarrow \infty} \mathbb{P}[G_{n,p} \text{ is Hamiltonian} \mid \delta(G) \geq 3] = 1?$$

Bollobás, Cooper, Fenner and Frieze [127] prove this result for c sufficiently large.

Problem 21 *Can the result of Section 3.1 be improved to $B = o(n)$?*

Problem 22 *Is there an analogous algorithm to that of Section 3.3 for the symmetric case where $c_{i,j} = c_{j,i}$ are i.i.d. uniform $[0,1]$?*

A natural approach is to find a minimum weight 2-factor first and then try to patch the cycles together. The problem with this is proving that such a 2-factor does not have many cycles.

Problem 23 *Suppose that $np = c$ in Theorem 13. Determine the limiting probability that $AP(C) = ATSP(C)$.*

It is shown in [333] that this probability is not 1.

Problem 24 *Determine the precise order of magnitude of $|AP(C) - ATSP(C)|$ under the assumptions of Section 3.3. If this is small, is there a polynomial time algorithm that solves $ATSP(C)$ exactly whp?*

Problem 25 *Suppose the edges of $G_{n,m}$ are randomly colored using n colors. What is the threshold for the existence of a Hamiltonian cycle in which each edge has a different color?*

The same question for spanning trees was solved in Frieze and McKay [335].

4. Euclidean Traveling Salesman Problem

Most of what follows has appeared elsewhere in the literature and our aim is to bring together the various methods and results. The focus is on describing those probabilistic methods used to analyze the TSP which have the potential to describe the behavior of heuristics as well. The methods described here also treat the probabilistic behavior of prototypical problems of Euclidean combinatorial optimization, including the minimal spanning tree problem, the minimal matching problem, and the Steiner minimal spanning tree problem. See the monographs of Steele [772] and Yukich [835] for details. Here we describe two key tools used in the probabilistic analysis of the total edge length of the Euclidean TSP on random samples of large size. The first tool is the boundary functional method and the second tool involves the isoperimetric inequalities of Rhee and Talagrand.

4.1. Basic Properties of the Euclidean TSP

If $F \subset \mathbb{R}^d$ is a point set and $R \subset \mathbb{R}^d$ a d -dimensional rectangle, then we let $T(F, R)$ denote the total edge length of the shortest tour through $F \cap R$. We view T as a function on pairs of the form (F, R) , where F is a finite set and R is a rectangle. When $R = [0, 1]^d$ we write $T(F)$ for $T(F, [0, 1]^d)$. This notation emphasizes that T is a function of two arguments and helps draw out the sub-additivity and super-additivity intrinsic to T . Elementary but essential properties of T include:

(a) *Monotonicity.* If $F \subseteq G$, then $T(F, R) \leq T(G, R)$ for all rectangles R .

(b) *Scaling* (homogeneity). For all $\alpha > 0$, for all rectangles R , and for all $F \subset R$

$$T(\alpha F, \alpha R) = \alpha T(F, R).$$

(c) *Translation invariance.* For all $y \in \mathbb{R}^d$, for all rectangles R , and for all $F \subset R$ we have

$$T(F, R) = T(F + y, R + y).$$

(d) *Geometric sub-additivity.* Subdivide $[0, 1]^d$ into m^d sub-cubes Q_1, Q_2, \dots, Q_{m^d} of edge length m^{-1} . Given $F \subset [0, 1]^d$, let $T(F \cap Q_i)$ denote the length of the shortest tour through $F \cap Q_i$. By adding and deleting edges it is easy to see that the length of the shortest tour through F is bounded above by the sum of the tour lengths $T(F \cap Q_i)$, $1 \leq i \leq m^d$, plus at most $2m^d$ edges each of length at most twice the diagonal of any sub-cube Q_i . This last set of edges is used to connect the minimal tours on each set $F \cap Q_i$, $1 \leq i \leq m^d$, into a feasible grand tour through F . Thus we have shown that T satisfies geometric sub-additivity with an error term:

$$T(F, [0, 1]^d) \leq \sum_{i=1}^{m^d} T(F \cap Q_i, Q_i) + C_1 m^{d-1}, \tag{27}$$

where C_1 is a finite constant.

(e) *Growth bounds.* For all $d = 2, 3, \dots$ there is a constant $C_2 := C_2(d)$ such that for all $F \subset [0, 1]^d$ we have

$$T(F, [0, 1]^d) \leq C_2 |F|^{(d-1)/d}. \tag{28}$$

These growth bounds follow from an easy application of the pigeonhole principle and, as noted by Rhee [717], also follow from geometric sub-additivity [835].

(f) *Smoothness* (Hölder continuity). For all $d = 2, 3, \dots$ there is a constant $C_3 := C_3(d)$ such that for all sets $F, G \subset [0, 1]^d$, T satisfies the smoothness condition:

$$|T(F \cup G) - T(F)| \leq C_3 |G|^{(d-1)/d}.$$

Since T is monotonic, the proof of smoothness follows once we show

$$T(F \cup G) \leq T(F) + C_3 |G|^{(d-1)/d}.$$

This last estimate is immediate since the length of the shortest tour through $F \cup G$ is bounded above by the sum of the length $T(F)$ of the shortest tour through F , the length $T(G)$ of the shortest tour through G , and the length of two edges connecting the sets F and G . Since $T(G) \leq C_2 |G|^{(d-1)/d}$, smoothness follows.

If a functional such as the TSP satisfies conditions (b),(c),(d), and (f), then we say that it is a *smooth sub-additive Euclidean functional* [768], [835].

The probabilistic analysis of the Euclidean TSP is simplified conceptually and technically by two key ideas, which are developed in the next sections. The first idea involves the concentration of the shortest tour length around its average value. By showing that the Euclidean TSP is tightly concentrated around its mean value, the probabilistic analysis of the TSP often reduces to an analysis of the behavior of its average value. This idea lies at the heart of asymptotic analysis. The second key idea involves a modified TSP functional, called the *boundary functional*, which closely approximates the standard TSP, and which also has a natural super-additive structure. By combining super-additivity together with sub-additivity, the TSP functional becomes “nearly additive” in the sense that

$$T(F \cap R) \approx T(F \cap R_1) + T(F \cap R_2),$$

where R_1 and R_2 form a partition of the rectangle R . Relations of this sort are crucial in showing that the global TSP tour length can be approximately expressed as a sum of the lengths of local components.

4.2. The Concentration of the TSP Around its Mean

The following basic result gives the a.s. asymptotics for $T(U_1, \dots, U_n)$, where U_1, \dots, U_n are i.i.d. uniform random variables in $[0, 1]^d$, $d \geq 2$.

These asymptotics were first proved by Beardwood, Halton, and Hammersley [94].

Theorem 26 *For all $d = 2, 3, \dots$ there is a finite positive constant $\beta(d)$ such that*

$$\lim_{n \rightarrow \infty} \frac{T(U_1, \dots, U_n)}{n^{(d-1)/d}} = \beta(d) \quad \text{a.s.}$$

Thus, the length of the average edge is roughly $\beta(d)n^{-1/d}$. In dimension two Rhee and Talagrand [722] show that the length of the longest edge in the TSP tour on U_1, \dots, U_n is bounded above by $C(\log n/n)^{1/2}$ with high probability. Theorem 26 shows that there are not many “long” edges.

There are many ways to prove Theorem 26 and we refer to the classic papers of Steele [768], [771] for simple proofs as well as non-trivial generalizations. Perhaps the easiest way to prove Theorem 26 involves showing that

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(U_1, \dots, U_n)}{n^{(d-1)/d}} = \beta(d), \tag{29}$$

and then showing that the total tour length $T(U_1, \dots, U_n)$ is closely approximated by the mean tour length $\mathbb{E}T(U_1, \dots, U_n)$.

The isoperimetric method lies at the heart of this approach. Loosely speaking, isoperimetric methods show that suitably regular functions are “close” to their average values [781]. For a full appreciation of the power of these methods, the reader should consult Section 4.8 below. For the moment we will consider only the following isoperimetric inequality. It is stated in a generality which lends itself to the study of both heuristics and a panoply of related problems in Euclidean combinatorial optimization.

Theorem 27 *(Rhee’s isoperimetric inequality [717]) Let $X_i, i \geq 1$, be independent random variables with values in $[0, 1]^d, d \geq 2$. Let L be a smooth, sub-additive Euclidean functional. Then there is a constant $C_4 := C_4(d)$ such that for all $t > 0$*

$$\begin{aligned} & \mathbb{P}[|L(X_1, \dots, X_n) - \mathbb{E}L(X_1, \dots, X_n)| > t] \\ & \leq C_4 \exp\left(-\frac{(t/C_3)^{2d/(d-1)}}{C_4 n}\right). \end{aligned} \tag{30}$$

The estimate (30) is an example of a deviation inequality: it tells us how L deviates from its average value. We call (30) an isoperimetric inequality since its proof [717],[835] rests upon an isoperimetric inequality for the Hamming distance on $([0, 1]^d)^n$. The upshot of (30) is that, excepting a set with polynomially small probability, the functional

$L(X_1, \dots, X_n)$ and its mean $\mathbb{E}L(X_1, \dots, X_n)$ do not differ by more than $C(n \log n)^{(d-1)/2d}$. The most useful consequence of Rhee’s concentration estimate (30) is that it reduces the problem of showing complete convergence of L to one of showing the convergence of the mean of L . The mean of L is a scalar and showing convergence of scalars is usually easier than showing convergence of random variables.

Corollary 28 (*Convergence of means implies complete convergence*)
 Let $X_i, i \geq 1$, be i.i.d. random variables with values in $[0, 1]^d, d \geq 2$. Let L be a functional which is homogeneous, translation invariant, and smooth. If the mean of L converges in the sense that

$$\lim_{n \rightarrow \infty} \mathbb{E}L(X_1, \dots, X_n)/n^{(d-1)/d} = \alpha(L, d)$$

then

$$\lim_{n \rightarrow \infty} L(X_1, \dots, X_n)/n^{(d-1)/d} = \alpha(L, d) \text{ c.c.}$$

Proof: The deviation estimate (30) implies for all $\epsilon > 0$

$$\sum_{n=1}^{\infty} \mathbb{P} \left\{ \left| \frac{L(X_1, \dots, X_n) - \mathbb{E}L(X_1, \dots, X_n)}{n^{(d-1)/d}} \right| > \epsilon \right\} \leq C \sum_{n=1}^{\infty} \exp \left(- \left(\frac{\epsilon}{C_3} \right)^{2d/(d-1)} \frac{n}{C_4} \right).$$

Thus, by definition, $\left| \frac{L(X_1, \dots, X_n) - \mathbb{E}L(X_1, \dots, X_n)}{n^{(d-1)/d}} \right|$ converges completely to zero and the proof is complete. ■

Thus to prove Theorem 26 it will be enough to show the limit (29). The next section describes a method of proof involving the boundary TSP. This approach helps prove probabilistic statements that go considerably beyond Theorem 26. For estimates of $\beta(d)$ we refer to [716] and pages 50-51 of [835].

4.3. The Boundary TSP

Given $F \subset [0, 1]^d$, the *boundary TSP functional* $T_B(F)$ is, loosely speaking, the cost of the least expensive tour through F , where the cost of travel within $[0, 1]^d$ is the usual Euclidean distance, but where travel along any path on the boundary $\partial[0, 1]^d$ of the unit cube is free.

We provide a more precise and slightly more general definition as follows. For all rectangles R , finite sets $F \subset R$, and pairs $\{a, b\} \subset \partial R$, let

$T(F, R, \{a, b\})$ denote the length of the shortest path through $F \cup \{a, b\}$ with endpoints a and b . The boundary TSP functional is given by

$$T_B(F, R) := \min \left(T(F, R), \inf \sum_i T(F_i, R, \{a_i, b_i\}) \right),$$

where the infimum ranges over all partitions $(F_i)_{i \geq 1}$ of F and all sequences of pairs of points $(a_i, b_i)_{i \geq 1}$ belonging to ∂R .

Boundary functionals were first used by Redmond [701] and Redmond and Yukich [702], [703] in the analysis of general Euclidean functionals. They are reminiscent of the “wired boundary condition” and the “wired spanning forest” used in the study of percolation and random trees, respectively. T_B may be interpreted as the length of the shortest tour through F which may repeatedly exit to the boundary of R at one point and re-enter at another, incurring no cost for travel along the boundary.

The boundary TSP functional T_B satisfies geometric super-additivity: if the rectangle R is partitioned into rectangles R_1 and R_2 then

$$T_B(F, R) \geq T_B(F, R_1) + T_B(F, R_2). \tag{31}$$

To see this, note simply that the restriction of the global tour realizing $T_B(F, R)$ to rectangle R_i , $1 \leq i \leq 2$, produces a feasible boundary tour of $F \cap R_i$, $1 \leq i \leq 2$, which by minimality has a length which is at least as large as $T_B(F, R_i)$. The absence of an error term in super-additivity contrasts sharply with the geometric sub-additivity of T . This distinction, which has telling consequences, often makes the analysis of T_B far easier than the analysis of T .

By definition $T_B \leq T$. It is easily checked that T_B satisfies all of the basic properties of the usual Euclidean TSP, excepting geometric sub-additivity.

To prove convergence of the mean of T (which by Corollary 28 implies c.c. convergence) over independent random variables U_1, U_2, \dots, U_n with the uniform distribution on $[0, 1]^d$, namely to prove the limit

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(U_1, \dots, U_n)}{n^{(d-1)/d}} = \beta(d),$$

it is enough to prove the following two basic lemmas:

Lemma 29 (*Asymptotics for the boundary functional*) *For every integer $d \geq 2$ there is a constant $\beta(d)$ such that*

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T_B(U_1, \dots, U_n)}{n^{(d-1)/d}} = \beta(d).$$

Lemma 30 (T_B approximates T) *The following approximation holds:*

$$|\mathbb{E}T_B(U_1, \dots, U_n) - \mathbb{E}T(U_1, \dots, U_n)| = o(n^{(d-1)/d}).$$

The proof of Lemma 30 is deferred to the next section. There are at least two ways to prove Lemma 29. The first way, which bears a likeness to the original bare hands approach of Beardwood et al. [94], relies on straightforward analysis of super-additive sequences of real numbers. The second way draws heavily on a multi-parameter super-additive ergodic theorem.

First proof of Lemma 29. Set $\phi(n) = \mathbb{E}T_B(U_1, \dots, U_n)$. Partition $[0, 1]^d$ into m^d sub-cubes Q_1, Q_2, \dots, Q_{m^d} of edge length m^{-1} . The number of points from the sample (U_1, \dots, U_n) which fall in a given sub-cube of $[0, 1]^d$ of volume m^{-d} is a binomial random variable $B(n, m^{-d})$ with parameters n and m^{-d} . It follows from scaling and the super-additivity (31) of T_B that

$$\phi(n) \geq m^{-1} \sum_{i=1}^{m^d} \phi(B(n, m^{-d})).$$

By smoothness and Jensen’s inequality (1) in this order we have

$$\begin{aligned} \phi(n) &\geq m^{-1} \sum_{i=1}^{m^d} \left(\phi(nm^{-d}) - C_3 \mathbb{E}(|B(n, m^{-d}) - nm^{-d}|^{(d-1)/d}) \right) \\ &\geq m^{-1} \sum_{i=1}^{m^d} \left(\phi(nm^{-d}) - C_3 (nm^{-d})^{(d-1)/2d} \right). \end{aligned}$$

Simplifying, we get

$$\phi(n) \geq m^{d-1} \phi(nm^{-d}) - C_3 m^{(d-1)/2} n^{(d-1)/2d}.$$

Dividing by $n^{(d-1)/d}$ and replacing n by nm^d yields the homogenized relation

$$\frac{\phi(nm^d)}{(nm^d)^{(d-1)/d}} \geq \frac{\phi(n)}{n^{(d-1)/d}} - \frac{C_3}{n^{(d-1)/2d}}.$$

Set $\beta := \beta(d) := \limsup_{n \rightarrow \infty} \frac{\phi(n)}{n^{(d-1)/d}}$ and note that $\beta \leq C_2$ by growth bounds (28). (Here and elsewhere the symbol “:=” means “equal to by definition”.) For all $\epsilon > 0$, choose n_o such that for all $n \geq n_o$, we have $C_3/n^{(d-1)/2d} \leq \epsilon$ and $\phi(n_o)/n_o^{(d-1)/d} \geq \beta - \epsilon$. Thus, for all $m = 1, 2, \dots$ it follows that

$$\frac{\phi(n_o m^d)}{(n_o m^d)^{(d-1)/d}} \geq \beta - 2\epsilon.$$

To now obtain Lemma 29 we use the smoothness of T_B and a simple interpolation argument. For an arbitrary integer $k \geq 1$ find the unique integer m such that

$$n_o m^d < k \leq n_o(m + 1)^d.$$

Then $|n_o m^d - k| \leq C n_o m^{d-1}$, where here and elsewhere C denotes a finite positive constant whose value may change from line to line. By smoothness we therefore obtain

$$\begin{aligned} \frac{\phi(k)}{k^{(d-1)/d}} &\geq \frac{\phi(n_o m^d)}{(n_o(m + 1)^d)^{(d-1)/d}} - \frac{C_3(C n_o m^{d-1})^{(d-1)/d}}{(m + 1)^{d-1} n_o^{(d-1)/d}} \\ &\geq (\beta - 2\epsilon) \left(\frac{m}{m + 1}\right)^{d-1} - \frac{C_3(C n_o m^{d-1})^{(d-1)/d}}{(m + 1)^{d-1} n_o^{(d-1)/d}}. \end{aligned}$$

Since the last term in the above goes to zero as m goes to infinity, it follows that

$$\liminf_{k \rightarrow \infty} \phi(k)/k^{(d-1)/d} \geq \beta - 2\epsilon.$$

Now let ϵ tend to zero to see that the \liminf and the \limsup of the sequence $\frac{\phi(k)}{k^{(d-1)/d}}$, $k \geq 1$, coincide, that is

$$\lim_{k \rightarrow \infty} \frac{\phi(k)}{k^{(d-1)/d}} = \beta.$$

We have thus shown

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T_B(U_1, \dots, U_n)}{n^{(d-1)/d}} = \beta$$

as desired. In the next few paragraphs we will see that β is positive. This concludes the first proof of Lemma 29. ■

Before providing the second proof of Lemma 29 we recall a general super-additive ergodic theorem. Let $\mathcal{R}(d)$ denote the d -dimensional rectangles of \mathbb{R}^d . Let $L = \{L(R), R \in \mathcal{R}(d)\}$ be a multi-parameter functional defined on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$. L is *stationary* if for all $m \geq 1$, $R_1, \dots, R_m \in \mathcal{R}(d)$, and $u \in (\mathbb{R}^+)^d$, the joint distributions of $L(R_1), \dots, L(R_m)$ and $L(R_1 + u), \dots, L(R_m + u)$ are the same. L is *bounded* if $\sup_n \mathbb{E}L([0, n]^d)/n^d < \infty$ and L is *strongly super-additive* if $L(R) \geq \sum_{i=1}^m L(R_i)$, where the rectangles R_1, R_2, \dots, R_m form a partition of R . Notice that strong super-additivity is stronger than the usual super-additivity.

The following strong law of large numbers of Akcoglu and Krengel [12] generalizes Kingman’s sub-additive ergodic theorem.

Theorem 31 (*Super-additive ergodic theorem*) Let $L := \{L(R) : R \in \mathcal{R}(d)\}$ be a stationary, bounded, super-additive functional defined on $(\Omega, \mathcal{A}, \mathbb{P})$. Then

$$\lim_{n \rightarrow \infty} \frac{L([0, n]^d)}{n^d} = f(L, d)$$

a.s. and in L^1 , where $f(L, d) \in L^1(\Omega, \mathcal{A}, \mathbb{P})$. Moreover,

$$\mathbb{E}f(L, d) = \alpha(L, d) = \sup_R \frac{\mathbb{E}L(R)}{\text{volume } R}.$$

$\alpha(L, d)$ is the *spatial constant* for the process L . It is a generalization of the *time constant* in the theory of one-dimensional sub-additive processes. It is now an easy matter to provide an alternate proof of Lemma 29.

Second proof of Lemma 29 [834]. Set

$$T_B(R) = T_B(\Pi \cap R, R), \quad R \in \mathcal{R}(d),$$

where Π is a Poisson point process on $(\mathbb{R}^+)^d$ with intensity 1. T_B is stationary and strongly super-additive. Since $\Pi \cap [0, n]^d \stackrel{d}{=} n(U_1, \dots, U_N)$, where N is an independent Poisson random variable with parameter n^d , we see that T_B is bounded:

$$\begin{aligned} \mathbb{E}T_B([0, n]^d) &= \mathbb{E}T_B(n(U_1, \dots, U_N), [0, n]^d) \\ &= n\mathbb{E}T_B(\{U_1, \dots, U_N\}, [0, 1]^d) \\ &\leq C_2 n \mathbb{E}N^{(d-1)/d} \\ &\leq C_2 n^d, \end{aligned}$$

by Jensen's inequality (1). The L^1 convergence given by the super-additive ergodic theorem implies

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T_B([0, n]^d)}{n^d} = \alpha(d) = \sup_{R \in \mathcal{R}(d)} \frac{\mathbb{E}T_B(R)}{\text{volume } R},$$

where we note that the spatial constant $\alpha(d)$ is clearly positive. Since $\mathbb{E}T_B([0, n]^d) = n\mathbb{E}T_B(U_1, \dots, U_N)$ we obtain

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T_B(U_1, \dots, U_N)}{n^{d-1}} = \alpha(d).$$

By straightforward smoothness arguments, the above convergence is unaffected if N is replaced by its mean n^d . Moreover, interpolation arguments show that n^d may be replaced by n , which thus yields Lemma 29 with $\beta(d) = \alpha(d)$. Notice that this proof shows that $\beta(d)$ is strictly positive. This concludes the second proof of Lemma 29. ■

4.4. The Boundary TSP Approximates the Standard TSP

To obtain laws of large numbers, rates of convergence of means, and large deviation principles, it is extremely useful to know that the boundary TSP functional closely approximates the standard TSP functional. The following estimate, which establishes Lemma 30, is a start in this direction:

$$|\mathbb{E}T(U_1, \dots, U_n) - \mathbb{E}T_B(U_1, \dots, U_n)| \leq Cn^{(d-2)/d}. \tag{32}$$

We will show (32) by following the approach of Redmond and Yukich [702], [703]. Since $T_B \leq T$, in order to prove (32) it suffices to show

$$\mathbb{E}T(U_1, \dots, U_n) \leq \mathbb{E}T_B(U_1, \dots, U_n) + Cn^{(d-2)/d}.$$

To show this, we first estimate the cardinality of points which are joined directly to the boundary. As in [835], let F denote one of the faces of $[0, 1]^d$. Letting $\mathcal{U}_F \subset \{U_1, \dots, U_n\}$ be the set of points that are joined directly to F by the graph realizing T_B , we first show that $\mathbb{E}|\mathcal{U}_F| \leq Cn^{(d-1)/d}$. For all $\epsilon > 0$ and $x \in F$, let $C(\epsilon, x)$ denote the cylinder in $[0, 1]^d$ determined by the ϵ disk in F centered at x . We make the critical observation that in the part of $C(\epsilon, x)$ which is at a distance greater than ϵ from F , there are at most two points which are joined to F . Were there three or more points, then two of these points could be joined with an edge, which would result in a cost savings, contradicting optimality. Since F can be covered with $O(\epsilon^{-(d-1)})$ disks of radius ϵ , we have the bound

$$\mathbb{E}|\mathcal{U}_F| \leq \mathbb{E}|\{x \in (U_i)_{i \leq n} : d(x, F) \leq \epsilon\}| + C\epsilon^{-(d-1)},$$

where $d(x, F)$ denotes the distance between the point x and the set F . The above is bounded by $n\epsilon + C\epsilon^{-(d-1)}$ and so putting $\epsilon = n^{-1/d}$ gives the desired estimate $\mathbb{E}|\mathcal{U}_F| \leq Cn^{(d-1)/d}$. If $\mathcal{U} \subset \{U_1, \dots, U_n\}$ denotes the set of points which are joined directly to any face of the boundary, then

$$\mathbb{E}|\mathcal{U}| \leq Cn^{(d-1)/d}.$$

We are now positioned to show (32). Let $\mathcal{U}' \subset \partial[0, 1]^d$ be the set of points on the boundary of $[0, 1]^d$ which are joined to points in \mathcal{U} . Let $T(\mathcal{U}')$ denote the total edge length of the graph of the minimal tour through \mathcal{U}' whose edges lie on $\partial[0, 1]^d$. Note that the union of the minimal tour graph through \mathcal{U}' and the boundary TSP graph through U_1, \dots, U_n defines an Eulerian path through $\{U_1, \dots, U_n\} \cup \mathcal{U}'$, where all vertices have even degree. Deleting some of the edges in this path yields a feasible tour

through $\{U_1, \dots, U_n\} \cup \mathcal{U}'$. It follows that

$$\mathbb{E}T(U_1, \dots, U_n) \leq \mathbb{E}T((U_1, \dots, U_n) \cup \mathcal{U}') \leq \mathbb{E}T_B(U_1, \dots, U_n) + \mathbb{E}T(\mathcal{U}').$$

Now $\mathbb{E}T(\mathcal{U}') \leq C\mathbb{E}|\mathcal{U}'|^{(d-2)/(d-1)}$ since \mathcal{U}' lies in the union of sets of dimension $d-1$. By Jensen's inequality (1), we find $\mathbb{E}T(\mathcal{U}') \leq Cn^{(d-2)/d}$, showing (32) as desired. ■

The approximation (32) is probabilistic. With a little extra effort one can show the following deterministic estimate. Let $F \subset [0, 1]^d$ and $|F| = n$. If $m^d = n/\sigma(n)$, where $\sigma(n)$ is an unbounded increasing function of n , then as explained in detail in [835], T satisfies the following near additivity condition

$$|T(F) - \sum_{i=1}^{m^d} T(F \cap Q_i, Q_i)| \leq C\sigma^{-1/d(d-1)}n^{(d-1)/d}. \tag{33}$$

The first term on the right hand side of (33) arises from approximating $T(F \cap Q_i)$, $1 \leq i \leq m^d$, by the length of the restriction of the global tour to sub-cube Q_i and the last term in (33) is just the sub-additive error m^{d-1} from (27).

In other words,

$$|T(F) - \sum_{i=1}^{m^d} T(F \cap Q_i, Q_i)| = o(n^{(d-1)/d}). \tag{34}$$

The approximation (32) shows that the asymptotics for $\mathbb{E}T(U_1, \dots, U_n)$ follow from the asymptotics for $\mathbb{E}T_B(U_1, \dots, U_n)$. However, these approximations, together with approximations (33) and (34) carry many additional benefits and provide:

- a straightforward probabilistic analysis of partitioning heuristics,
- estimates for the *rate of convergence* of $\mathbb{E}T(U_1, \dots, U_n)$,
- asymptotics for $T(X_1, \dots, X_n)$, where X_i , $i \geq 1$, are i.i.d. with an arbitrary distribution, and
- large deviation principles for $T(U_1, \dots, U_n)$.

The following sections explore these benefits.

4.5. Analysis of Heuristics

The a.s. asymptotics of Theorem 26 led Karp [496], [497] to find efficient methods for approximating the length $T(U_1, \dots, U_n)$ of the shortest

path through i.i.d. uniformly distributed random variables U_1, \dots, U_n on the unit square. In his seminal work [496], [497], Karp developed the “fixed dissection algorithm” which provides a simple heuristic having a tour of length $T_H(U_1, \dots, U_n)$ with the property that

$$T_H(U_1, \dots, U_n)/T(U_1, \dots, U_n)$$

converges completely to 1 and has polynomial mean execution time.

Karp’s fixed dissection algorithm consists of dividing the unit cube $[0, 1]^d$ into m^d congruent sub-cubes Q_1, \dots, Q_{m^d} , finding the shortest tour T_i of length $T(\{U_1, \dots, U_n\} \cap Q_i)$ on each of the sub-cubes, constructing a tour T which links representatives from each T_i , and then deleting excess edges to generate a grand (heuristic) tour through U_1, \dots, U_n having length $T_H(U_1, \dots, U_n)$.

Karp and Steele [500] show via elementary methods that the partitioning heuristic T_H is ϵ -optimal with probability one:

Theorem 32 (Karp and Steele, [500]) *If $m^d := n/\sigma$, where σ is an unbounded increasing function of n , then for all $\epsilon > 0$*

$$\sum_{n=1}^{\infty} \mathbb{P} \left\{ \frac{T_H(U_1, \dots, U_n)}{T(U_1, \dots, U_n)} \geq 1 + \epsilon \right\} < \infty.$$

Theorem 32 shows that the ratio of the lengths of the heuristic tour and the optimal tour converges completely to 1. Given the computational complexity of the TSP, it is remarkable that the optimal tour length is so well approximated by a sum of individual tour lengths, where the sum has polynomial mean execution time.

Since Karp’s work [496], [497], considerable attention has been given to developing the probability theory of heuristics. Goemans and Bertsimas [384] develop the a.s. asymptotics for the Held-Karp heuristic [444]. They do this by essentially showing that the Held-Karp heuristic is a smooth sub-additive Euclidean functional and therefore is amenable to the methods discussed here. To develop the probability theory of other heuristics, such as Christofides’ heuristic, one could hope to apply modifications of the methods discussed here. This could involve approximating the heuristic by a super-additive “boundary heuristic”. Here we limit ourselves to a discussion of the proof of Theorem 32 using the approximations (33) and (34).

Given $F \subset [0, 1]^d$ consider the feasible tour through F obtained by solving the optimization problem on the sub-cubes Q_i , $1 \leq i \leq m^d$, and then adding and deleting edges in the resulting graph to obtain a global solution on the set F . This feasible solution, which we call the canonical

heuristic H , has a total edge length denoted by $T_H(F, m^d)$. $T_H(F, m^d)$ is the sum of the lengths $T(F \cap Q_i)$, $1 \leq i \leq m^d$, plus a correction term which is bounded by $C_1 m^{d-1}$. Thus T_H satisfies

$$T(F) \leq T_H(F, m^d) \leq \sum_{i=1}^{m^d} T(F \cap Q_i) + C_1 m^{d-1}.$$

As in the previous section, we let the number of sub-cubes depend on the cardinality of F , denoted by $|F|$ for brevity. To make this precise, let $\sigma := \sigma(n)$ denote a function of n such that $\sigma(n)$ and $n/\sigma(n)$ increase up to infinity. Such functions σ define heuristics $H := H(\sigma)$ having length

$$T_{H(\sigma)} \left(F, \frac{|F|}{\sigma(|F|)} \right).$$

Thus the heuristic $H(\sigma)$ subdivides the unit cube into $\frac{|F|}{\sigma(|F|)}$ sub-cubes. If we let $m^d := \frac{|F|}{\sigma(|F|)}$ we obtain from the near additivity condition (33)

$$|T(F) - T_{H(\sigma)}(F)| = o(n^{(d-1)/d}). \tag{35}$$

Thus the length $T_{H(\sigma)}(F)$ is larger than $T(F)$ by a quantity which is deterministically small compared to $n^{(d-1)/d}$. Therefore the asymptotic behavior of the scaled heuristic

$$T_{H(\sigma)}(X_1, \dots, X_n)/n^{(d-1)/d} \tag{36}$$

coincides with the asymptotic behavior of

$$T(X_1, \dots, X_n)/n^{(d-1)/d}, \tag{37}$$

where X_i , $i \geq 1$, are i.i.d. random variables with values in the unit cube $[0, 1]^d$. We will see shortly (see Theorem 35 below) that the ratio (37) converges completely to a positive constant whenever the law of X_1 has a continuous part. By (35) it thus follows that (36) also converges completely to a constant. It therefore follows by standard arguments that the ratio of (36) to (37) converges completely to 1. We have thus extended Karp and Steele's result to general sequences of random variables:

Theorem 33 *(The heuristic T_H is ϵ -optimal over general sequences)*
 For all $\epsilon > 0$ and all i.i.d. sequences X_i , $i \geq 1$, of random variables with a continuous part, the heuristic $T_{H(\sigma)}$ is ϵ -optimal:

$$\sum_{n=1}^{\infty} \mathbb{P} \left\{ \frac{T_H(X_1, \dots, X_n)}{T(X_1, \dots, X_n)} \geq 1 + \epsilon \right\} < \infty. \tag{38}$$

We conclude the discussion of heuristics by verifying that the expected execution time for $T_H(U_1, \dots, U_n)$ is polynomially bounded. The required computing time is bounded by

$$T_n := \sum_{i=1}^{n/\sigma(n)} f(N_i),$$

where $N_i := |\{Q_i \cap \{U_1, \dots, U_n\}\}|$, $1 \leq i \leq n/\sigma(n)$, and where $f(N)$ denotes a bound on the time needed to compute $T(F)$, $|F| = N$. It is well-known that we may take f to have the form $f(x) = Ax^B 2^x$, for some constants A and B . Since the N_i , $1 \leq i \leq n/\sigma(n)$, are binomial random variables, straightforward calculations [500] show that

$$\mathbb{E}T_n \leq 4An(\sigma(n))^{B-1} \exp(\sigma(n)).$$

Now choose $\sigma(n) = \log n$ to conclude that the expected execution time for the heuristic T_H is $O(n^2 \log^{B-1} n)$, as desired.

4.6. Rates of Convergence of Mean Values

The sub-additivity of the TSP functional is not enough to give rates of convergence. Sub-additivity only yields one sided estimates whereas rate results require two sided estimates. However, since the TSP functional can be made super-additive through use of the boundary TSP functional this will be enough to yield rates of convergence. Similar methods apply for other problems in Euclidean combinatorial optimization [835]. Alexander [15] obtains rate results without the use of boundary functionals. Rhee [719] shows that the following rates are optimal on the unit square provided that $\mathbb{E}T(U_1, \dots, U_n)$ is replaced by the Poissonized version $\mathbb{E}T(U_1, \dots, U_{N(n)})$.

Theorem 34 (*Rates of convergence of means*) *For all $d = 2, 3, \dots$, there is a constant C such that*

$$|\mathbb{E}T(U_1, \dots, U_n) - \beta(d)n^{(d-1)/d}| \leq Cn^{(d-2)/d}.$$

Proof: We will first prove

$$|\mathbb{E}T(U_1, \dots, U_{N(n)}) - \beta(d)n^{(d-1)/d}| \leq Cn^{(d-2)/d}, \tag{39}$$

where we adhere to the convention that $N(n)$ denotes an independent Poisson random variable with parameter n . The proof of (39) involves simple but useful sub-additive techniques. We set

$$\phi(n) = \mathbb{E}T(U_1, \dots, U_{N(n)}).$$

It follows from translation invariance, homogeneity, and sub-additivity, that

$$\phi(nm^d) \leq m^{-1} \sum_{i=1}^{m^d} \phi(n) + Cm^{d-1} = m^{d-1}\phi(n) + cm^{d-1}.$$

Dividing by $(nm^d)^{(d-1)/d}$ yields the homogenized relation

$$\frac{\phi(nm^d)}{(nm^d)^{(d-1)/d}} \leq \frac{\phi(n)}{n^{(d-1)/d}} + \frac{C}{n^{(d-1)/d}}.$$

The limit of the left side as m tends to infinity exists and equals $\beta(d)$. Thus

$$\frac{\phi(n)}{n^{(d-1)/d}} - \beta(d) \geq \frac{-C}{n^{(d-1)/d}}$$

or simply

$$\phi(n) - \beta(d)n^{(d-1)/d} \geq -C. \tag{40}$$

Setting $\phi_B(n) := \mathbb{E}T_B(U_1, \dots, U_{N(n)})$ and using the super-additivity of T_B in the same way that we exploited the sub-additivity of T , we obtain the companion estimate to (40) where we may now let $C = 0$:

$$\phi_B(n) - \beta(d)n^{(d-1)/d} \leq 0. \tag{41}$$

By closeness (32) of T and T_B , Fubini's theorem, and independence, we have

$$\begin{aligned} |\phi_B(n) - \phi(n)| &\leq \mathbb{E}_N |\mathbb{E}_U T_B(U_1, \dots, U_N) - \mathbb{E}_U T(U_1, \dots, U_N)| \\ &\leq \mathbb{E}_N (CN^{(d-2)/d}) \\ &\leq C(n^{(d-2)/d}) \end{aligned}$$

where \mathbb{E}_N and \mathbb{E}_U denote the expectation with respect to the random variables N and U , respectively. Now (39) follows from (40) and (41). It remains to de-Poissonize (39) to obtain Theorem 34. Notice by smoothness that

$$|\mathbb{E}T(U_1, \dots, U_N) - \mathbb{E}T(U_1, \dots, U_n)| \leq C\mathbb{E}|N - n|^{(d-1)/d} \leq Cn^{(d-1)/2d},$$

and thus when $d \geq 3$ we easily obtain Theorem 34. For $d = 2$ this simple method does not work and to obtain Theorem 34 we need slightly more work (see [835]).

This completes the proof of Theorem 34. ■

4.7. Asymptotics over Non-Uniform Samples

Let $X_i, i \geq 1$, be i.i.d. random variables with values in $[0, 1]^d, d \geq 2$, and let f denote the density of the absolutely continuous part of the law μ of X_1 . In their seminal paper, Beardwood, Halton, and Hammersley [94] established the following asymptotics for the length of the shortest tour $T(X_1, \dots, X_n)$. It generalizes Theorem 26.

Theorem 35 (*Asymptotics for the TSP over non-uniform samples*) For all $d = 2, 3, \dots$ we have

$$\lim_{n \rightarrow \infty} \frac{T(X_1, \dots, X_n)}{n^{(d-1)/d}} = \beta(d) \int_{[0,1]^d} f(x)^{(d-1)/d} dx \quad \text{c.c.} \quad (42)$$

The limit (42) is proved in two stages. One first proves (42) when the density of μ is a step function of the form $\sum_{i=1}^{m^d} \alpha_i 1_{Q_i}, \alpha_i \in \mathbb{R}^+,$ where $Q_i, 1 \leq i \leq m^d,$ is a partition of $[0, 1]^d$ into sub-cubes of edge length m^{-1} . Then (42) is deduced by approximating general densities by step densities. There are several ways to carry out this program. The easiest arguments (see Steele [768], [772]) make clever use of the monotonicity of T , but such arguments are not easily generalized to treat other Euclidean functionals. We now outline a simple approach to (42) which is general enough to deliver asymptotics for a wide variety of problems in combinatorial optimization and computational geometry and which has the potential to describe asymptotics for heuristics as well. The approach yields c.c. asymptotics and it can be extended to treat the case when the random variables $X_i, i \geq 1,$ have unbounded support (Rhee [718]). The methods here are part of the ‘‘umbrella approach’’ described in detail in Yukich [835].

The proof of (42) depends upon two observations which greatly simplify the analysis. The first is that by Corollary 28, it is enough to show that (42) holds in expectation, i.e. to show

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(X_1, \dots, X_n)}{n^{(d-1)/d}} = \beta(d) \int_{[0,1]^d} f(x)^{(d-1)/d} dx. \quad (43)$$

The limit (43) is a statement about a sequence of scalars and, in principle, is easier to prove than the limit (42).

The second observation is that in the presence of smoothness of T , it is enough to establish (43) for a special class of distributions which we call *blocked* distributions. These are distributions μ on $[0, 1]^d$ with the form $\phi(x)dx + \mu_s,$ where $\phi(x)$ is a simple non-negative function of the form $\sum_{i=1}^{m^d} \alpha_i 1_{Q_i},$ where the measure μ_s is purely singular and $Q_i, i \geq 1,$

are the usual sub-cubes. More precisely, we have the following lemma which is due to Steele [770].

Lemma 36 *Suppose that for every sequence of i.i.d. random variables $X_i, i \geq 1$, distributed with a blocked distribution $\mu := \phi(x)dx + \mu_s$ we have that*

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(X_1, \dots, X_n)}{n^{(d-1)/d}} = \beta(d) \int_{[0,1]^d} \phi(x)^{(d-1)/d} dx. \tag{44}$$

We then have that

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(Y_1, \dots, Y_n)}{n^{(d-1)/d}} = \beta(d) \int_{[0,1]^d} f(x)^{(d-1)/d} dx \tag{45}$$

where $Y_i, i \geq 1$, are i.i.d. random variables whose law has an absolutely continuous part given by $f(x)dx$.

Proof: Assume that the distribution of Y has the form $\mu_Y := f(x)dx + \mu_s$, where μ_s is singular. For all $\epsilon > 0$ we may find a blocked approximation to μ_Y of the form $\mu_X := \phi(x)dx + \mu_s$, where $\phi := \phi_\epsilon$ approximates f in the L^1 sense:

$$\int_{[0,1]^d} |\phi(x) - f(x)| dx < \epsilon.$$

By standard coupling arguments there is a joint distribution for the pair of random variables (X, Y) such that $\mathbb{P}[X \neq Y] \leq 2\epsilon$. Thus it follows that

$$\begin{aligned} |\mathbb{E}T(X_1, \dots, X_n) - \mathbb{E}T(Y_1, \dots, Y_n)| &\leq C \mathbb{E}|\{i \leq n : X_i \neq Y_i\}|^{(d-1)/d} \\ &\leq C(\epsilon n)^{(d-1)/d}. \end{aligned}$$

Thus, by (44) we obtain

$$\lim_{n \rightarrow \infty} \left| \frac{\mathbb{E}T(Y_1, \dots, Y_n)}{n^{(d-1)/d}} - \beta(d) \int_{[0,1]^d} \phi(x)^{(d-1)/d} dx \right| \leq C(\epsilon^{(d-1)/d}). \tag{46}$$

For all $a, b \geq 0$ we have

$$|a^{(d-1)/d} - b^{(d-1)/d}| \leq |a - b|^{(d-1)/d}$$

and therefore by the L^1 approximation

$$\left| \int f(x)^{\frac{d-1}{d}} dx - \int \phi(x)^{\frac{d-1}{d}} dx \right| \leq \int |f(x) - \phi(x)|^{\frac{d-1}{d}} dx \tag{47}$$

$$\leq \epsilon^{\frac{d-1}{d}}. \tag{48}$$

Combining (46) and (47) and letting ϵ tend to zero gives (45) as desired. ■

We are now ready to prove Theorem 35. By the above lemma, the proof of Theorem 35 is reduced to showing (44). By simple smoothness arguments it is enough to show

$$\lim_{n \rightarrow \infty} \frac{\mathbb{E}T(X_1, \dots, X_{N(n)})}{n^{(d-1)/d}} = \beta(d) \int_{[0,1]^d} \phi(x)^{(d-1)/d} dx, \tag{49}$$

where $N(n)$ is a Poisson random variable with parameter n . For simplicity we will assume $\mu_s = 0$.

Note that, for each $1 \leq j \leq m^d$, $|\{i \leq N(n) : Y_i \in Q_j\}|$ is a Poisson random variable $N(n\alpha_j m^{-d})$. Let $U_i, i \geq 1$, be i.i.d. uniform random variables with values in $[0, 1]^d$. By geometric sub-additivity and scaling we have

$$\mathbb{E}T(X_1, \dots, X_{N(n)}) \leq m^{-1} \sum_{j=1}^{m^d} \mathbb{E}T(\{U_i\}_{i=1}^{N(n\alpha_j m^{-d})}) + Cm^{d-1}.$$

By Theorem 26 it follows that

$$\begin{aligned} \limsup_{n \rightarrow \infty} \frac{\mathbb{E}T(X_1, \dots, X_{N(n)})}{n^{(d-1)/d}} &\leq \\ \sum_{j=1}^{m^d} \limsup_{n \rightarrow \infty} \left(\frac{\mathbb{E}T(U_1, \dots, U_{N(n\alpha_j m^{-d})})}{n^{(d-1)/d} \alpha_j^{(d-1)/d} m^{-(d-1)}} \cdot \alpha_j^{(d-1)/d} m^{-d} \right) &= \\ \beta(d) \int_{[0,1]^d} \phi(x)^{(d-1)/d} dx. \end{aligned}$$

Similarly, by geometric super-additivity and by Lemma 29 we see that

$$\liminf_{n \rightarrow \infty} \frac{\mathbb{E}T_B(X_1, \dots, X_{N(n)})}{n^{(d-1)/d}} \geq \beta(d) \int_{[0,1]^d} \phi(x)^{(d-1)/d} dx$$

and the desired limit (49) follows since $T \geq T_B$.

4.8. Talagrand’s Isoperimetric Theory

Section 4.2 described how the TSP deviates from its average value. The concentration estimates of Theorem 27 are general and apply to functionals which are homogeneous, translation invariant, and smooth. In the case of the TSP, however, the deviation estimates of Theorem 27

are far from optimal. In this section we will see that Talagrand's [781], [782] deep isoperimetric methods for product spaces yields improved tail estimates for the TSP. These estimates resemble the tail estimates of a standard normal random variable and suggest that perhaps the TSP is asymptotically normally distributed. Isoperimetric methods have the potential to describe the deviations of heuristics as well.

Talagrand's isoperimetric theory is substantial and has had a profound impact in the probability theory of random graphs. This section illustrates how one piece of Talagrand's theory may be used to study the tail behavior of the TSP. Our presentation is motivated by the fine exposition of Steele [772]. Naturally, we will not present all the technical aspects of the theory and refer the reader to Steele [772] and Talagrand [781] for complete details.

To study the tail behavior of $T(U_1, \dots, U_n)$, where U_i , $i \geq 1$, are i.i.d. with the uniform distribution on the unit cube $[0, 1]^d$, it is useful to view T as a functional defined on the n -fold product space $\Omega^n := ([0, 1]^d)^n$. We adopt this point of view and will thus consider isoperimetric methods on the product space Ω^n , where the triple $(\Omega, \mathcal{A}, \mathbb{P})$ is a probability space. The following discussion makes no use of the fact that Ω is the unit cube and indeed, the value of isoperimetric theory lies precisely in the fact that $(\Omega, \mathcal{A}, \mathbb{P})$ can be any abstract probability space.

Given $A \subset \Omega^n$, Talagrand's isoperimetric theory provides estimates of the measure of the set of points in Ω^n which are within a specified distance of A . There are many ways to define a distance on Ω^n and perhaps the simplest is the Hamming distance $d_H(x, y) := |\{1 \leq i \leq n : x_i \neq y_i\}|$. The Hamming distance from a point $x \in \Omega^n$ to a set $A \subset \Omega^n$ is given by

$$d_H(x, A) := \inf_{y \in A} \sum_{i=1}^n I_{\{x_i \neq y_i\}}.$$

Rhee's isoperimetric inequality (30) is actually based on a simple isoperimetric inequality involving the Hamming distance d_H .

Talagrand [781] shows for every $\lambda > 0$ and every probability measure \mathbb{P} on Ω that the Hamming distance satisfies the following exponential integrability condition:

$$\int_{\Omega^n} \exp(\lambda d_H(x, A)) d\mathbb{P}^n \leq \frac{1}{\mathbb{P}^n[A]} \exp(-n\lambda^2/4). \quad (50)$$

We will see shortly how to use this sort of condition to derive concentration estimates for the TSP. Talagrand's method of proof actually

applies to all of the Hamming metrics

$$d_a(x, y) := \sum_{i=1}^n a_i I_{\{x_i \neq y_i\}}, \quad a = (a_1, \dots, a_n) \in (\mathbb{R}^+)^n,$$

provided that the factor of n in the exponent of (50) is replaced by $\|a\|^2 = \sum_{i=1}^n a_i^2$. We let $d_a(x, A) := \inf_{y \in A} d_a(x, y)$.

By considering a still different and slightly more complicated method for measuring distance we will be able to improve upon the isoperimetric inequality (30) when the functional L is the TSP. Talagrand's *convex hull control* distance or simply *convex* distance is given by

$$d_c(x, A) := \sup_{\|a\|=1} d_a(x, A).$$

The convex distance uniformly controls the Hamming metrics d_a , $\|a\| = 1$. The present definition of d_c , while the simplest for our purposes, somewhat obscures its convexity properties.

There is a second way to formulate the definition of d_c :

$$d_c(x, A) := \min \left\{ t : \forall \{a_i\} \exists y \in A \text{ s.t. } \sum_{i=1}^n a_i I_{\{x_i \neq y_i\}} \leq t \left(\sum_{i=1}^n a_i^2 \right)^{1/2} \right\}.$$

It is straightforward to see that these two definitions of d_c are equivalent. Notice that by letting $a_i = n^{-1/2}$ for all $1 \leq i \leq n$ we see that $d_c(x, A)$ is always at least as large as $n^{-1/2} d_H(x, A)$.

The convex distance satisfies an exponential square integrability condition, as described by the next theorem, also due to Talagrand [781]. The proof [781] of this theorem uses induction on n and while it is surprisingly short, we will not reproduce it here. Notice that this integrability condition is in general stronger than (50).

Theorem 37 *For every set $A \subset \Omega^n$ we have*

$$\int_{\Omega^n} \exp \left(\frac{1}{4} d_c^2(x, A) \right) d\mathbb{P}^n(x) \leq \frac{1}{\mathbb{P}^n[A]}.$$

It follows by the generalized Chebyshev inequality (2) with $f(x) = \exp(\frac{1}{4}x^2)$ that

$$\mathbb{P}^n[d_c(x, A) > t] \leq \frac{e^{-t^2/4}}{\mathbb{P}^n[A]}. \tag{51}$$

This is an abstract and general deviation bound for the convex distance. It is far from obvious that this general bound has relevance to

the tail behavior of the TSP. We will now see shortly how to put this to good use. We first need one technical lemma which helps us compare the behavior of T on $x := \{x_1, \dots, x_n\}$ and $y := \{y_1, \dots, y_n\}$. For details concerning the proof see Steele [772].

Lemma 38 *There is a non-negative weight function $a(x) := (a_1(x), \dots, a_n(x))$ such that for all $x := \{x_1, \dots, x_n\}$ and $y := \{y_1, \dots, y_n\}$ the TSP functional satisfies*

$$T(x_1, \dots, x_n) \leq T(y_1, \dots, y_n) + \sum_{i=1}^n a_i(x) I_{\{x_i \neq y_i\}}, \tag{52}$$

where $\|a(x)\|^2 \leq C^2$ uniformly in x .

Combining the above lemma and Theorem 37 we can now show that T does not deviate much from its median. We recall that $U_i, i \geq 1$, are i.i.d. random variables with the uniform distribution on $[0, 1]^d$. We let M_n denote a median of $T(U_1, \dots, U_n)$ and we follow Steele [772] closely. The next statement, which improves upon the martingale estimates of Rhee and Talagrand ([720] and [721]), shows that $T(U_1, \dots, U_n) - M_n$ exhibits sub-Gaussian tail behavior. For $d = 2$ and t in the range $0 \leq t \leq Cn^{1/2}$, this tail behavior is sharp [715].

Corollary 39 *(Concentration for the TSP) We have*

$$\mathbb{P}[|T(U_1, \dots, U_n) - M_n| \geq t] \leq 4 \exp(-t^2/4C^2).$$

Proof: Following Steele [772], we use Theorem 37 for the parameterized family of sets

$$A(b) := \{\{y_1, y_2, \dots, y_n\} : T(y_1, \dots, y_n) \leq b\}.$$

Let $a := a(x)$ be as in Lemma 38. We know by inequality (52) for all $\{x_1, x_2, \dots, x_n\}$ and $\{y_1, y_2, \dots, y_n\}$ that

$$T(\{x_1, \dots, x_n\}) \leq T(\{y_1, \dots, y_n\}) + \sum_{i=1}^n a_i(x) I_{\{x_i \neq y_i\}}.$$

Minimizing over $y \in A(b)$ gives

$$\begin{aligned} T(x_1, x_2, \dots, x_n) &\leq b + \min_{y \in A(b)} \sum_{i=1}^n a_i(x) I_{\{x_i \neq y_i\}} \\ &= b + \min_{y \in A(b)} \|a\| \sum_{i=1}^n (a_i(x)/\|a\|) I_{\{x_i \neq y_i\}} \\ &\leq b + Cd_c(x, A(b)) \end{aligned}$$

by the definition of $d_c(x, A(b))$ and by the bound $\|a\| \leq C$. We obtain for $x = \{U_1, U_2, \dots, U_n\}$ that $d_c(x, A(b)) \geq C^{-1}(T(U_1, U_2, \dots, U_n) - b)$.

By Theorem 37 we have

$$\mathbb{P}^n[d_c(x, A(b)) > t] \leq \frac{1}{\mathbb{P}^n[A(b)]} \exp(-t^2/4),$$

and therefore if we write $T_n := T(U_1, \dots, U_n)$ then

$$\mathbb{P}^n[T_n \geq b + Ct] \leq \frac{1}{\mathbb{P}^n[A(b)]} \exp(-t^2/4).$$

Thus, letting $u = Ct$, we have

$$\mathbb{P}^n[T_n \leq b] \mathbb{P}^n[T_n \geq b + u] \leq \exp(-u^2/4C^2)$$

and by letting $b = M_n$ and then $b = M_n - u$ we complete the proof of Corollary 39. ■

4.9. Remarks and Further Applications of Boundary Functionals

As seen in earlier sections, boundary functionals are well suited for describing rates of convergence, asymptotics, and the analysis of heuristics. Here we provide some additional benefits of boundary functionals.

(a) *Large Deviation Principles.* Letting $F = \{U_1, \dots, U_n\}$, the near additivity condition (33) expresses the fact that the global tour length through F is roughly the sum of i.i.d. local tour lengths. Together with smoothness, this condition shows that if $X(n) = n^{1/d}T(U_1, \dots, U_{N(n)})$, where $N(n)$ is the usual independent Poisson random variable with parameter n , then $X(n)$ satisfies the following Donsker-Varadhan large deviation principle. This principle quantifies the precise deviations of $X(n)/n$ and refines the strong law of large numbers expressed in Theorem 26. For $t \in \mathbb{R}$ we let

$$\Lambda(t) := \lim n^{-1} \log \mathbb{E}[\exp tX(n)]$$

be the logarithmic moment generating function for $X(n)$. The convex dual is

$$\Lambda^*(x) := \sup_{t \in \mathbb{R}} \{tx - \Lambda(t)\}.$$

Theorem 40 (*Large deviation principle for the TSP [750]*)

For all closed sets $F \subset \mathbb{R}$ we have

$$\limsup_{n \rightarrow \infty} \log \mathbb{P}[n^{-1}X(n) \in F] \leq - \inf_{x \in F} \Lambda^*(x)$$

and for all open sets $O \subset \mathbb{R}$ we have

$$\liminf_{n \rightarrow \infty} \log \mathbb{P}[n^{-1} X(n) \in O] \geq - \inf_{x \in O} \Lambda^*(x).$$

Moreover, Λ^* has its unique zero at $\beta(d)$.

The proof of Theorem 40 is long and involved. Complete details may be found in Seppäläinen and Yukich [750], which also gives an entropic characterization of Λ^* .

(b) *Power Weighted Edges.* For all $p > 0$, consider the length $T^p(F)$ of the shortest tour through F with p th power weighted edges. Thus

$$T^p(F) := \min_T \sum_{e \in T} |e|^p,$$

where the minimum is over all tours T and where $|e|$ denotes the Euclidean edge length of the edge e . The method of boundary functionals [551], [835] shows that for all $0 < p < d$ we have the following generalization of (42):

$$\lim_{n \rightarrow \infty} \frac{T^p(X_1, \dots, X_n)}{n^{(d-p)/d}} = \beta(d, p) \int_{[0,1]^d} f(x)^{(d-p)/d} dx \quad c.c., \quad (53)$$

where $\beta(d, p)$ is a constant depending only on d and p . For the case $p = d$, a more delicate use of boundary functionals [833] shows that if U_i , $i \geq 1$, are i.i.d. with the uniform distribution on $[0, 1]^d$ then

$$\lim_{n \rightarrow \infty} T^d(U_1, \dots, U_n) = C(d) \quad a.s.,$$

where $C(d)$ is some finite constant.

(c) *Worst Case Tour Length.* Let the largest possible length of a minimal tour with p th power weighted edges through n points in $[0, 1]^d$ be denoted by

$$\tau^p(n) := \max_{F \subset [0,1]^d, |F|=n} T^p(F).$$

By considering boundary TSP functionals (with power weighted edges) it is particularly easy to show the asymptotics

$$\lim_{n \rightarrow \infty} \frac{\tau^p(n)}{n^{(d-p)/d}} = \gamma(d, p)$$

where $\gamma(d, p)$ is some positive constant. Steele and Snyder [773] were the first to prove these asymptotics, although they restricted attention

to the case $p = 1$. Using boundary functionals, Yukich [835] treats the case $1 \leq p < d$ and Lee [552] treats the cases $0 < p < 1$ and $p \geq d$.

(d) *Directed TSP*. Consider the random directed graph G_n whose vertices are independent and uniformly distributed random variables U_1, \dots, U_n on the unit square. For $1 \leq i \leq j \leq n$, the orientation of the edge $U_i U_j$ is selected at random, independently for each edge and independently of the U_i , $i \geq 1$. The edges in G_n are given a direction by flipping fair coins. The directed TSP involves finding the shortest directed path through the random vertex set. If $D(n) = D(U_1, \dots, U_n)$ denotes the length of the shortest directed path through the sample U_1, \dots, U_n , then Steele [769] shows that

$$\lim_{n \rightarrow \infty} \mathbb{E}D(n)/n^{1/2} = \alpha$$

for some constant α . Talagrand [780] shows that this convergence can be improved to complete convergence. It is not clear whether boundary functionals can be used to obtain asymptotics over more general point sets as in (42).

(e) *Maximum Length Tours*. Although it is more natural to consider minimum length tours, there has been some interesting work on maximum length tours, see Chapter 12. We mention here a result of Dyer, Frieze and McDiarmid [263]. n points are placed uniformly at random in $[0, 1]^2$. A tour T is constructed as far as possible from edges which join points which are almost reflections in the center. This gives a tour which is close to twice the length of a maximum weight matching and hence is almost of maximum weight. It is shown that if L_n denotes the length of T and L_{\max} denotes the length of the longest tour then almost surely

$$\frac{L_{\max}}{n} \rightarrow 2 \int_0^1 \int_0^1 r(x, y) dx dy \approx 0.7652 \quad \text{and} \quad \frac{L_n}{L_{\max}} \rightarrow 1,$$

where $r(x, y)$ is the Euclidean distance between the points (x, y) and $(\frac{1}{2}, \frac{1}{2})$.

4.10. Open Problems

Problem 41 *Theorem 40 and Talagrand’s concentration inequality for the TSP provide circumstantial evidence that the TSP functional satisfies asymptotic normality in the following sense:*

$$\frac{T(U_1, \dots, U_n) - \mathbb{E}T(U_1, \dots, U_n)}{(\text{Var}T(U_1, \dots, U_n))^{1/2}} \xrightarrow{d} N(0, 1).$$

Here $N(0,1)$ denotes a standard normal random variable. Proving or disproving the above central limit theorem remains a difficult open problem and would add to the central limit theorem for the length of the Euclidean minimal spanning tree over a random sample [16], [503], [550].

Problem 42 *Develop the a.s. limit theory for Christofides' heuristic. In particular, does Christofides' heuristic satisfy the limit given by Theorem 35?*

The methods described here can possibly be modified to treat Christofides' heuristic. This may involve defining a superadditive boundary heuristic.

Problem 43 *Develop the a.s. limit theory for the directed TSP: investigate whether the directed TSP satisfies the limit in Theorem 35.*

Problem 44 *Establish that the scaled variance of the TSP functional converges. In dimension 2, this means showing that $\text{Var}(U_1, \dots, U_n) \rightarrow C$, where C is some positive constant.*

Problem 45 *Find theoretical values for the limiting constants $\beta(d)$ and $\gamma(d, p)$. Perhaps the problem is simplified by using a metric on \mathbb{R}^d other than the Euclidean metric. Rhee [716] shows that in high dimensions $\beta(d)$ is close to $(d/2\pi e)^{1/2}$.*

Problem 46 *Use the Aldous - Steele objective method [14] to show the a.s. and the L^1 convergence of $T^p(U_1, \dots, U_n)$ when p equals the dimension d .*

This would provide a second way to identify the limiting constant $\beta(d, d)$ and it would complement the Aldous-Steele results [14] on the random Euclidean minimal spanning tree. It would also add to the limit result (53).

Problem 47 *The large deviation principle (Theorem 40) holds for a Poisson number of uniform random variables. Does this LDP hold for a fixed deterministic number of uniform random variables?*

Problem 48 *Generalize Theorem 40 by establishing a general large deviation principle for the random variables $T(X_1, \dots, X_n)$, where X_i , $i \geq 1$, are i.i.d. with an arbitrary distribution on $[0, 1]^d$. Show that the rate function exhibits quadratic behavior.*

Problem 49 *Bi-partite matching. Given X_i , $1 \leq i < \infty$, and Y_i , $1 \leq i < \infty$, two independent sequences of random variables with the uniform distribution on $[0, 1]^d$, the bi-partite matching problem studies the*

behavior of

$$T_n = \min_{\pi} \sum_{i=1}^n \|X_i - Y_{\pi(i)}\|,$$

where π runs over all permutations of the integers $1, 2, \dots, n$. When $d \geq 3$ subadditive methods [257] imply that $\mathbb{E}T_n/n^{(d-1)/d} \rightarrow C$ for some constant C . When $d = 2$, subadditive methods fail and it is known (see [11], [783]) only that there are positive constants C_1 and C_2 such that

$$C_1 \leq \mathbb{E}T_n/(n \log n)^{1/2} \leq C_2.$$

The logarithmic term in the denominator prevents us from applying the usual subadditive arguments to conclude that

$$\mathbb{E}T_n/(n \log n)^{1/2} \rightarrow C, \tag{54}$$

where C is some positive constant. Proving or disproving (54) remains an intriguing unsolved problem.

Problem 50 *Bi-partite TSP.* Given $X_i, 1 \leq i < \infty$, and $Y_i, 1 \leq i < \infty$, two independent sequences of random variables with the uniform distribution on $[0, 1]^2$, the bi-partite TSP involves finding the length of the shortest tour through $X_i, 1 \leq i \leq n$, and $Y_i, 1 \leq i \leq n$, such that each X point is joined to two Y points and vice versa. If T_n denotes the length of the shortest such tour then clearly T_n is bounded below by the length of the bi-partite matching on the union of $X_i, 1 \leq i \leq n$ and $Y_i, 1 \leq i \leq n$. However, finding the asymptotics of T_n remains open.

Problem 51 *Non-standard scaling.* Theorem 35 tells us that whenever random variables X_1, X_2, \dots have an absolutely continuous part, then $T(X_1, \dots, X_n)$ scales like $n^{(d-1)/d}$. This raises the following question: given an arbitrary increasing function $f(n) = o(n^{(d-1)/d})$, when do there exist i.i.d. random variables X_1, \dots, X_n such that

$$\lim_{n \rightarrow \infty} \mathbb{E}T(X_1, \dots, X_n)/f(n) = C,$$

where C is a positive finite constant?

Chapter 8

LOCAL SEARCH AND METAHEURISTICS

César Rego

Hearin Center for Enterprise Science

School of Business Administration, University of Mississippi, MS 38677

crego@bus.olemiss.edu

Fred Glover

Hearin Center for Enterprise Science

School of Business Administration, University of Mississippi, MS 38677

fglover@bus.olemiss.edu

1. Background on Heuristic Methods

The Traveling Salesman Problem (TSP) is one of the most illustrious and extensively studied problems in the field of Combinatorial Optimization. Covering just the period from 1993 to mid-2001 alone, the web databases of INFORMS and Decision Sciences report more than 150 papers devoted to the TSP. The problem can be stated in graph theory terms as follows. Let $G = (V, A)$ be a weighted complete graph, where $V = \{v_1, \dots, v_n\}$ is a vertex (node) set and $A = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ is an edge set. $C = [c(v_i, v_j)]$ is a $n * n$ matrix associated with A , where $c(v_i, v_j)$ is a non-negative weight (distance or cost) on edge (v_i, v_j) if there is an edge between v_i and v_j . Otherwise $c(v_i, v_j)$ is infinity.

The problem is said to be symmetric (STSP) if $c(v_i, v_j) = c(v_j, v_i)$ for all $(v_i, v_j) \in A$, and asymmetric (ATSP) otherwise. Elements of A are often called arcs (rather than edges) in the asymmetric case. The STSP (ATSP) consists of finding the shortest Hamiltonian cycle (circuit) in G , which is often simply called a *tour*. In the symmetric case, G is an *undirected* graph, and it is common to refer to the edge set $E = \{(v_i, v_j) | v_i, v_j \in V, i < j\}$ in place of A . The version of STSP in which distances satisfy the triangle inequality ($c(v_i, v_j) + c(v_j, v_k) \geq c(v_i, v_k)$ for all distinct $v_i, v_j, v_k \in V$) is perhaps the most-studied special case of

the problem, notably including the particular instance where V is a set of points in a 2-dimensional plane and $c(v_i, v_j)$ is the Euclidean distance between v_i and v_j .

Important variants and extensions of the TSP arise in the setting of vehicle routing (see Laporte and Osman [539]). A variety of other interesting problems not directly related to routing can also be modeled as TSPs, as is shown in the survey of Laporte [533]. Distances or costs that are symmetric and satisfy the triangle inequality are predominantly encountered in such applications. This chapter mainly deals with the STSP and for the sake of simplicity we will generally refer to the problem as the TSP.

The TSP can be formulated as an integer linear programming (ILP) model, and a number of exact algorithms are based on such a formulation. However, there are also some advantages to representing the TSP directly as a permutation problem without transforming it into an ILP, and we will focus on such a direct representation. Let π denote a cyclic permutation mapping so that the sequence $i, \pi(i), \pi^2(i), \dots, \pi^{n-1}(i)$ for $i \in N = \{1, \dots, n\}$ identifies a permutation of the elements of N , where $\pi^n(i) = i$. Let Π be the set of all such mappings. Thus, solving a particular instance of the TSP consists of finding a cycle permutation (tour) $\pi^* \in \Pi$ such that

$$\sum_{i=1}^{n-1} c_{i\pi^*(i)} = \min_{\pi \in \Pi} \sum_{i=1}^{n-1} c_{i\pi(i)}$$

The TSP is one of the classical instances of an NP-complete problem, and therefore there is no polynomial-time algorithm able to determine π^* for all possible instances of the problem (unless $P=NP$). Perhaps because of the simplicity of its statement the TSP has been a key source of important developments in NP-completeness theory (see e.g., Johnson and Papadimitriou, 1985). It has also been the subject of several polyhedral studies (see Chapters 2 and 3). As a result, although $P=NP$ continues to be an improbable hypothesis, and hence a polynomial-time algorithm for the TSP is not likely to be discovered, current specialized TSP optimization codes have been solving general TSP instances involving about three-thousand vertices. Specifically, the Concorde package of Applegate, Bixby, Chvatal and Cook [29] solved all instances up to 3200 cities in the 8th DIMACS TSP Challenge testbed (Johnson, McGeoch, Glover, and Rego [462]) using its default settings, except one 3162-city random uniform Euclidian instance for which non-default twiddling was necessary to find the optimal tour.

State-of-the-art exact solution methods (which guarantee an optimal solution if run long enough) can typically solve problems involving about 1000 vertices in reasonable computation time, but encounter significant difficulties in solving larger problems, where they generally require computational effort that exceeds the realm of practicality. Even for modest-size problems, exact methods require substantially greater computation time than the leading heuristic methods, which in addition are capable of finding optimal or very-close-to-optimal solutions for instances far larger than those reasonably attempted by exact methods. An extensive empirical analysis of computational results and algorithmic performance for several classes of TSP heuristics is described in Chapter 9.

The aim of this chapter is to present an overview of classical and modern local search procedures for the TSP and discuss issues involved in creating more efficient and effective algorithms. Heuristic algorithms for the TSP can be broadly divided into two classes: *tour construction procedures*, which build a tour by successively adding a new node at each step; and *tour improvement procedures*, which start from an initial tour and seek a better one by iteratively moving from one solution to another, according to adjacency relationships defined by a given neighborhood structure. (Specialized tour construction heuristics are treated in Chapter 9.) Combined, such approaches yield *composite procedures* that attempt to obtain better solutions by applying an improvement procedure to a solution given by a construction procedure. Often, the success of these algorithms depends heavily on the quality of the initial solution. Iterated variants of construction and improvement procedures provide a natural means of elaborating their basic ideas, as subsequently discussed.

Recent Developments in Overview. Recent progress in local search methods has come from designing more powerful neighborhood structures for generating moves from one solution to another. These advances have focused on *compound neighborhood structures*, which encompass successions of interdependent moves, rather than on simple moves or sequences of independent moves. On the other hand, the more sophisticated neighborhood structures entail greater numbers of operations, and therefore an increased effort to perform each step of the algorithm. Thus, several studies have investigated strategies to combine neighborhoods efficiently, and thereby reduce the computational effort of generating trajectories within them. These methods are generally *variable depth methods*, where the number of moves carried out at each iteration is dynamically determined, and usually varies from one iteration to the next. A common characteristic of these methods is a *look ahead* process where a relatively large sequence of moves is generated, each step leading

to a different trial solution, and the compound move that yields the best trial solution (from the subsequences beginning with the initial move) is the one chosen. Two types of variable depth neighborhood structures have become prominent:

- (1) *connected neighborhood structures* as represented by:
 - a. Variable Neighborhood Search (VNS) (Mladenović and Hansen [602], Hansen and Mladenović [433, 432]),
 - b. Sequential Fan (SF) methods (Glover and Laguna [379]),
 - c. Filter and Fan (FF) methods (Glover [376]);
- (2) *disconnected neighborhood structures* as represented by:
 - a. Lin-Kernighan (LK) methods (Lin and Kernighan [563]),
 - b. Chained and Iterated LK methods (Martin, Otto and Felten [588], Johnson and McGeogh [463], Applegate, Cook and Rohe [32]),
 - c. Ejection Chain (EC) methods (Glover [372], [374]).

In the TSP setting, connected neighborhood procedures are exemplified at a simple level by classical k -opt and Or-opt methods which keep the Hamiltonian (feasible tour) property at each step. Variable depth methods of these types consist of component moves that directly link one tour to the next, thus generating streams of moves and associated trial solutions. Conversely, the LK and EC methods consider sequences of moves that do not necessarily preserve the connectivity of the tour, although they enable a feasible tour to be obtained as a trial solution by performing an additional move. Apart from this commonality, Lin-Kernighan and Ejection Chains differ significantly in the form of the intermediate (disconnected) structures that link one move to the next in the sequence. LK methods rely on a Hamiltonian path as an intermediate structure, while EC methods embrace a variety of intermediate structures, each accompanied by appropriate complementary moves to create feasible trial solutions. The Lin-Kernighan procedure is described in Section 2.2. Ejection chains structures and the moves that join and complement them are elaborated in Section 2.3, followed by advanced variable depth methods in Section 2.4.

Local Search and Meta-Heuristic Approaches. Local search techniques (which terminate at a local optimum) and associated meta-heuristic strategies (which modify and guide local techniques to explore the solution space more thoroughly) have been the focus of widespread scientific investigation during the last decade. For more than twenty years

two main "meta models" for heuristic techniques have been ascendant: those based on "single stream" trajectories, and those based on "multiple stream" trajectories, where the latter seek to generate new solutions from a collection (or population) of solutions. The distinction is essentially the same as that between serial and parallel algorithms, with the allowance that population-based methods can also be applied in a serial manner, as in a serial simulation of a parallel approach. Consequently, as may be expected, there are some overlaps among the best procedures of these two types. Traditionally, however, population-based methods have often been conceived from a narrower perspective that excludes strategies commonly employed with single stream methods. Thus, more modern approaches that embody features of both methods are often called hybrid procedures.

Some of the methods discussed in this chapter have fairly recently come into existence as general purpose methods for a broad range of combinatorial optimization problems, and have undergone adaptation to provide interesting specializations for the TSP. This manifests one of the reasons for the enduring popularity of the TSP: it often serves as a "problem of choice" for testing new methods and algorithmic strategies.

The remainder of this chapter is organized as follows. Section 2 presents classical and more recent improvement methods that have proven effective for the TSP. It also discusses several special cases of neighborhood structures that can be useful for the design of more efficient heuristics. Section 3 gives an overview of the tabu search metaheuristic, disclosing the fundamental concepts and strategies that are relevant in the TSP context. Section 4 extends the exploration of metaheuristics to the description and application of recent unconventional evolutionary methods for the TSP. Section 5 presents some concluding observations and discusses possible research opportunities.

2. Improvement Methods

Broadly speaking, improvement methods are procedures that start from a given solution, and attempt to improve this solution by iterative change, usually by manipulating relatively basic solution components. In graph theory settings, depending on the problem and the type of algorithm used, these components can be nodes, edges, (sub)paths or other graph-related constructions. We consider three classes of improvement methods according to the type of neighborhood structures used:

- (1) *constructive neighborhood methods*, which successively add new components to create a new solution, while keeping some components of the current solution fixed. (These include methods that assem-

ble components from different solutions, and methods that simply choose different parameters of a construction procedure using information gathered from previous iterations.)

- (2) *transition neighborhood methods*, usually called local search procedures, which iteratively move from one solution to another based on the definition of a neighborhood structure.
- (3) *population-based neighborhood methods*, which generalize (1) and (2) by considering neighborhoods of more than one solution as a foundation for generating one or more new solutions.

In this section we focus our discussion on the second class of improvement methods and specifically on those that are either considered classical or the core of the most efficient TSP algorithms known to date.

For the following development we assume a starting TSP tour is given and is recorded by identifying the immediate predecessor and successor of each node v_i , which we denote respectively v_{i-} and v_{i+} .

2.1. Basic Improvement Procedures

Fundamental neighborhood structures for the TSP (and for several other classes of graph-based permutation problems) are based on edge-exchanges and node-insertion procedures. Classical procedures of these types are the k -exchange (Lin [562]) and the Or-insertion (Or [635]) which also form the core of several more advanced procedures covered in the next sections. Before describing the various neighborhood structures underlying these two classes of procedures, it is appropriate to note that the concept of local optimality has a role in the nomenclature of k -Opt and Or-Opt – terms sometimes used inappropriately in the TSP literature. In a local search method a neighborhood structure is introduced to generate moves from one solution to another and, by definition, a local optimum is a solution that can not be improved by using the neighborhood structure under consideration. Accordingly, a local optimum produced by an improvement method using k -exchanges or Or-insertion yields what is called a k -optimal (k -Opt) or a Or-optimal (Or-opt) solution, respectively.

2.1.1 k -exchange Neighborhoods. The terminology of k -exchange neighborhoods derives from methods initially proposed by Lin [562] to find so-called “ k -opt” TSP tours. The 2-exchange (2-opt) procedure is the simplest method in this category and is frequently used in combinatorial problems that involve the determination of optimal cir-

cuits (or cycles) in graphs. This includes the TSP and its extensions to the wider classes of assignment, routing and scheduling problems.

The 2-opt procedure is a local search improvement method, and a starting feasible solution is required to initiate the approach. The method proceeds by replacing two non-adjacent edges (v_i, v_{i+}) and (v_j, v_{j+}) by two others (v_i, v_j) and (v_{i+}, v_{j+}) , which are the only other two edges that can create a tour when the first two are dropped. In order to maintain a consistent orientation of the tour by the predecessor-successor relationship, one of the two subpaths remaining after dropping the first two edges must be reversed. For example, upon reversing the subpath (v_{i+}, \dots, v_j) the subpath $(v_i, v_{i+}, \dots, v_j, v_{j+})$ is replaced by $(v_i, v_j, \dots, v_{i+}, v_{j+})$. Finally, the solution cost change produced by a 2-exchange move can be expressed as $\Delta_{ij} = c(v_i, v_j) + c(v_{i+}, v_{j+}) - c(v_i, v_{i+}) - c(v_j, v_{j+})$. A 2-optimal (or 2-opt) solution is obtained by iteratively applying 2-exchange moves until no possible move yields a negative Δ value.

The 2-opt neighborhood process can be generalized to perform k -opt moves that drop some k edges and add k new edges. There are $\binom{n}{k}$ possible ways to drop k edges in a tour and $(k-1)!2^{k-1}$ ways to relink the disconnected subpaths (including the initial tour) to recover the tour structure. For small values of k , relative to n , this implies a time complexity of $O(n^k)$ for the verification of k -optimality, and therefore the use of k -opt moves for $k > 3$ is considered impractical unless special techniques for restricting the neighborhood size are used. (To date, $k = 5$ is the largest value of k that has been used in algorithms for large scale TSPs.) We now summarize some of the main advances in the design of more efficient k -opt procedures.

2.1.2 Special Cases of k -opt Neighborhoods. A useful observation for implementing restricted k -opt moves is that any k -opt move for $k > 2$ is equivalent to a finite sequence of 2-opt moves, assuming the graph is fully dense. (This is a result of the easily demonstrated fact that in such a graph any tour can be transformed into any other by a succession of 2-opt moves.) Consequently, if no sequence of k consecutive 2-opt moves can improve the current tour, then it is also a k -optimal tour. However, the reverse is not necessarily true – i.e. a tour can be k -optimal, but obviously there may exist a sequence of k successive 2-opt moves that reduces the length of the tour (since every tour can be reached in this way in a fully dense graph). Thus, a comparative analysis of neighborhoods with successive k values provides a foundation for designing more efficient k -opt procedures by restricting the attention to moves that are not included within $(k-1)$ -opt neighborhoods.

By direct analogy to the 2-opt move, a 3-opt move consists of deleting three edges of the current tour (instead of two) and relinking the endpoints of the resulting subpaths in the best possible way. For example, letting (v_i, v_{i+}) , (v_j, v_{j+}) and (v_k, v_{k+}) denote the triplet of edges deleted from the current tour, two of the seven possible ways to relink the three subpaths consist of (1) creating edges (v_i, v_{j+}) , (v_k, v_{i+}) , and (v_j, v_{k+}) ; and (2) creating edges (v_i, v_j) , (v_{j+}, v_{k+}) , and (v_{i+}, v_k) .

An important difference between these two possibilities to create 3-opt moves is that the orientation of the tour is preserved in (1), while in (2) the subpaths (v_{i+}, \dots, v_j) and (v_{j+}, \dots, v_k) have to be reversed to maintain a feasible tour orientation. The cost of a 3-opt move can be computed as Δ_{ijk} , the sum of the costs of the added edges minus the sum of the costs of the deleted edges, where a negative Δ represents an improving move. Generically, similar computations and conditions for reversing subpaths result for any k -opt move.

Another way to reduce the time complexity of 3-opt moves comes from the observation that a 2-opt move is a special case of a 3-opt move in which a deleted edge is added back to relink a subpath. Consequently, three of the seven possible 3-opt moves correspond to 2-opt moves. Thus, if the tour is already 2-optimal, then these three types of 2-exchange moves need not be checked in the 3-opt process. An additional class of 3-opt moves can be obtained as a sequence of two (non-independent) 2-opt moves. This special case may occur when an edge inserted by the first 2-opt move is deleted by the application of the second 2-opt move. Three other 3-opt moves fall into this special case. Figure 8.1 illustrates one of these possibilities, applied to the TSP tour given in Figure 8.1A. The 3-opt move is represented in Figure 8.1B where the symbol e_0 is used to label the edges deleted by the move. Similarly, Figure 8.1C illustrates the application of two successive 2-opt moves where e_0 and e_1 are the edges deleted by the first and the second application of the move. Note that edge e_1 is one of the edges added by the first application of the 2-opt move. Figure 8.1D represents the TSP tour that results from the application of either the 3-opt move or the indicated sequence of two 2-opt moves.

The foregoing observations indicate that out of the seven possible 3-opt moves only one requires a sequence of three 2-opt moves, so that only a very small fraction of the $\binom{3}{k}$ possible combinations need to be considered. Also, as described in Christofides and Eilon [190] a 4-opt neighborhood (which involves 47 possible 4-opt moves) includes six 2-opt

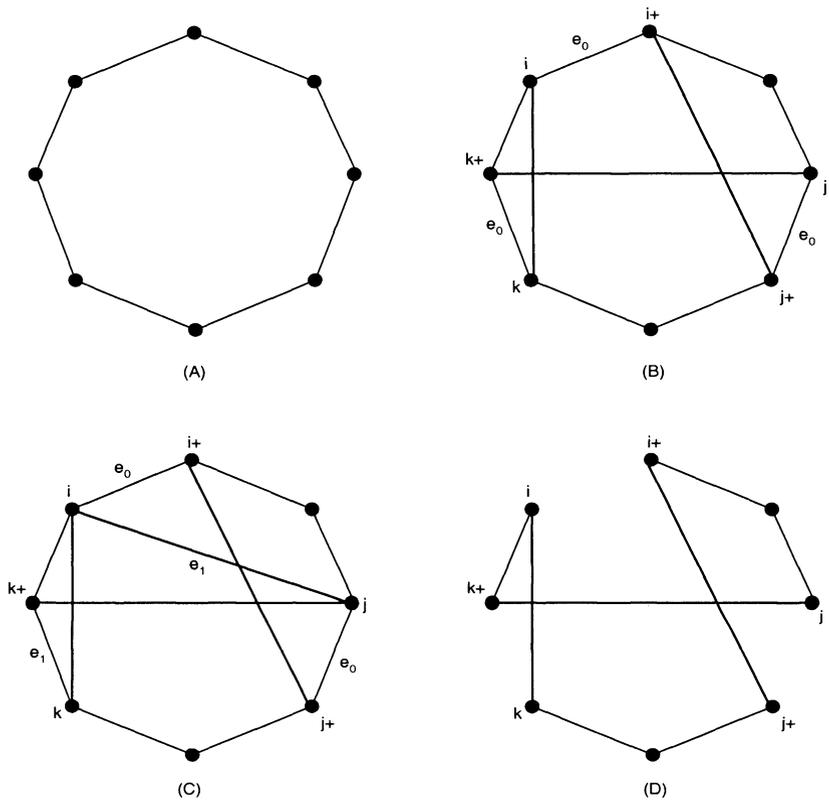


Figure 8.1. 3-opt obtained by two successive 2-opt moves.

moves, sixteen sequences of two 2-opt moves, and twenty-five sequences of three 2-opt moves. Consequently, the consideration of sequences of three 2-opt moves derived from interconnecting restricted 2-opt moves in successive levels is sufficient to ensure the tour is 4-optimal. In contrast, the determination of a 5-optimal tour requires examination of a sequence of at least five 2-opt moves, yielding significantly more combinatorial possibilities than the sequences of three 2-opt moves required by 3-opt and 4-opt neighborhoods. This provides an indication of the relatively greater advantage of 5-opt neighborhoods over 4-opt neighborhoods when compared to the advantages of 4-opt over 3-opt, as demonstrated by Christofides and Eilon [190] and more recently by Helsgaun [446]. A multi-stage 2-opt approach appears particularly useful to implement variable depth methods, as discussed in Section 2.4.

2.1.3 Special Cases of Insertion and k -Opt Neighborhoods.

Another useful relationship emerges from the comparative analysis of k -opt moves relative to several classes of insertion TSP moves. We define two basic types of node-based moves within the TSP setting:

- (1) *node insertion moves*: a selected node v_i is inserted between two adjacent nodes v_p and v_q in the tour by adding edges (v_p, v_i) , (v_i, v_q) , (v_{i-}, v_{i+}) and dropping edges (v_p, v_q) , (v_{i-}, v_i) , (v_i, v_{i+}) .
- (2) *node exchange moves*: two nodes v_i and v_j exchange positions by adding edges (v_{i-}, v_j) , (v_j, v_{i+}) , (v_{j-}, v_i) , (v_i, v_{j+}) and dropping edges (v_{i-}, v_i) , (v_i, v_{i+}) , (v_{j-}, v_j) , (v_j, v_{j+}) . An exception occurs if (v_i, v_j) is an edge of the tour, in which case the move is equivalent to inserting v_i between v_j and v_{j+} (or inserting v_j between v_{i-} and v_i).

Or-Opt Neighborhoods and Extensions. A generalization of the foregoing node-based neighborhoods consists of extending these processes to insert and exchange sequences (or subpaths) of consecutive edges in the tour. By treating subpaths as if they were nodes this generalized process can be implemented using operations similar to the ones defined for the node insertion/exchange moves.

Two classical methods that seek to reduce the complexity of the 3-opt procedure are Bentley's 2.5-opt and Or-opt (Or [635]). 2.5-opt is an extension of the 2-opt procedure that considers a single-node insertion move when 2-opt fails to improve (Bentley [103]). The Or-opt heuristic proceeds as a multi-stage generalized insertion process, which starts by considering the insertion of three-node subpaths (between two adjacent nodes) and then successively reduces the process to insert two-node subpaths (hence edges) and finally to insert single nodes, changing the type

of move employed whenever a local optimum is found for the current neighborhood. Note that node-insertion and edge-insertion moves are special cases of 3-opt moves when a subpath between two dropped edges of the 3-opt move consists of just one node or edge, respectively. Also, as mentioned before, most 3-opt moves do not preserve the orientation of the tour. Now, it is easy to see that the Or-opt procedure restricts the 3-opt neighborhood to a subclass of moves that preserves the current tour orientation. The time complexity of this procedure is $O(n^2)$. However, while the Or-opt neighborhood has proved relatively efficient when applied to some constrained traveling salesman and vehicle routing problems, the procedure does not appear to present a competitive advantage when compared to efficient implementations of the 3-opt procedure. (Chapter 9 provides details on efficient implementations of 3-opt.) A possible enhancement of the classical Or-opt procedure arises from a generalization based on an ejection chain framework, as we discuss in Section 2.3.2. Such a generalization gives the procedure the ability to create a variable depth neighborhood search similar to the one the Lin-Kernighan procedure performs with classical k -opt moves.

Constructive/Destructive Neighborhoods for Restricting k -opt Moves. Gendreau, Hertz, and Laporte [351] propose a generalized insertion procedure (GENI) which may be viewed as a combination of single-node insertion moves with 4-opt and 5-opt moves. GENI is used in the constructive phase of their GENIUS algorithm to create a starting tour, beginning from an arbitrary cycle of 3 vertices. The alternating use of GENI with its reverse procedure (destructively removing nodes from the tour) forms the basis of the String/Unstring (US) neighborhood structure used in the local search phase of the GENIUS algorithm. Thus, the successive application of Unstring and String creates a destructive/constructive type of neighborhood structure that typically generates restricted forms of 8-opt, 9-opt, and 10-opt moves. The process can be viewed as a one-step (unit depth) strategic oscillation (see Section 3.3).

The destructive *Unstring* neighborhood structure removes a vertex from the current tour by replacing k edges by $k - 1$ other edges for $k = 4$ or 5. Figure 8.2 depicts an example of the *Unstring* process where node v_i is removed from the tour. In the figure, diagrams A and C represent the initial tours to apply an Unstring process with $k = 4$ and $k = 5$, respectively. Diagrams B and D represent the resulting subgraphs after applying the *Unstring* move, which removes the edges labeled “e” and relinks the respective subpaths as illustrated. In this procedure, edges defined by nodes identified with different letters generically represent subpaths having these nodes as their endpoints.

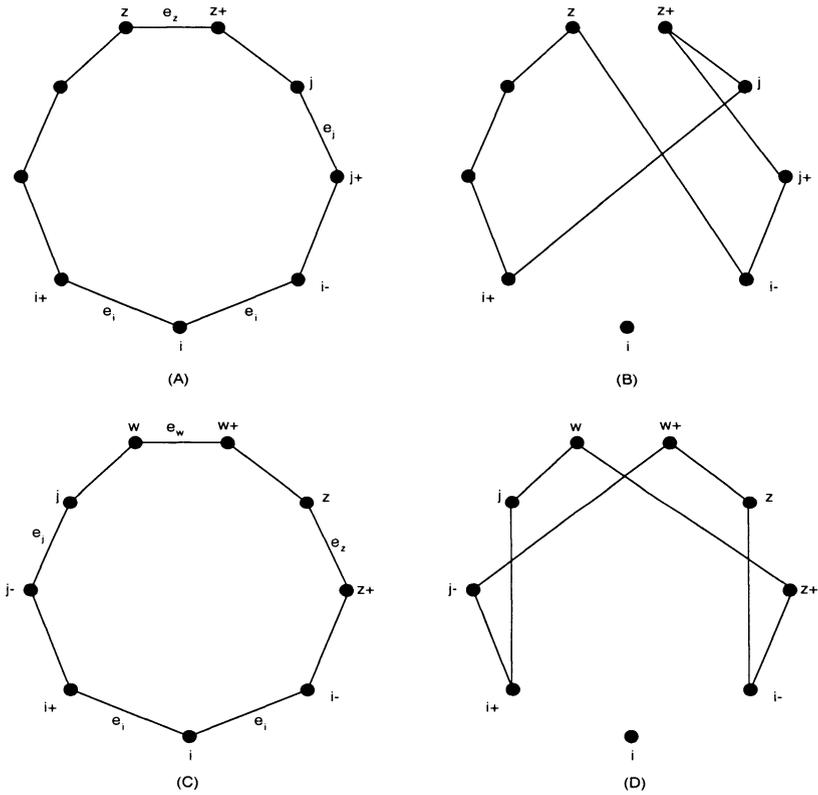


Figure 8.2. The String/Unstring Neighborhood Structure

String is a constructive neighborhood structure that reverses the operations of the *Unstring* procedure to insert a disconnected vertex v_i between two other vertices (not necessarily consecutive in the tour) by replacing k edges by $k + 1$ other edges for $k = 3$ or 4 . Figure 8.2 illustrates *String* moves for $k = 3$ and $k = 4$ by following the diagrams in the order from B to A and from D to C, respectively.

Recent Results on Restricting k -Opt Neighborhoods. Useful results for reducing computation required by k -opt procedures are provided by Glover [375] and Helsgaun [446]. Glover’s paper shows that the best move from a sub-collection of 4-opt moves (which embraces all 2-opt moves, an additional class of 3-opt moves, and two principal classes of 4-opt moves) can be found in the same order of time required to find a best 2-opt move. The method is based on an acyclic shortest path model underlying the creation of *dynamic alternating paths and cycles* generated by an ejection chain framework as discussed in Section 2.3. The use of ejection chains to generate special forms of alternating paths and cycles also proves useful in the implementation of the stem-and-cycle ejection chain method described in Rego [704] and discussed in Section 2.3.3. Helsgaun considers the use of 5-opt moves to replace 2-opt moves in the basic step of the classic implementation of the Lin-Kernighan (LK) procedure as discussed in Section 2.2. In Helsgaun’s particular variant of the LK procedure 5-opt moves are made computationally practicable by restricting the possible alternatives using special candidate lists, in this case augmenting a “close-by neighbor” list to include additional nodes identified by solving Lagrangean relaxations over minimum spanning tree (1-tree) relaxations as introduced by Held and Karp ([444, 445]).

2.2. The Classical Lin-Kernighan Procedure

The Lin-Kernighan (LK) procedure (Lin and Kernighan [563]) is a strategy for generating k -opt moves where the value of k is dynamically determined by performing a sequence of 2-opt moves. Although, as noted, any k -opt move can be represented by a sequence of 2-opt moves, the LK procedure limits attention to a particular subset of these sequences. The goal is to restrict the neighborhood search and at the same time to generate high quality k -opt moves. The 2-opt moves are generated in successive levels where a 2-opt move of level i ($i = 2, \dots, L, k = i + 1$) drops one of the edges that has been added by the 2-opt move of the previous level ($i - 1$). An exception is made for the first 2-opt move of the sequence, which can start either from the current tour or after performing a special class of 3-opt or 4-opt moves. However, this excep-

tion is only allowed when a sequence of 2-opt moves in the regular LK search fails to improve the initial tour. These special cases will become clear later in the detailed explanation of each step of the method.

The method starts by generating a 2-opt, 3-opt or 4-opt move and then deletes an edge adjacent to the last one added to create a Hamiltonian path. Thereafter, each new move consists of adding an edge to the degree 1 node that was not met by the last edge added, and dropping the unique resulting edge that again will recover a Hamiltonian path (thus completing the last add-drop portion of a 2-opt move).

It is interesting to note that these moves (and in fact the first add-drop half of any 2-opt move) “pass through” a *stem-and-cycle structure*, which is one of the reference structures introduced by Glover [374] as a foundation for creating more flexible and general types TSP moves. However, this structure is not identified as relevant in the LK method, which relies solely on the *Hamiltonian path structure* as a basis for composing its moves, and consequently is left unexploited within the LK framework. (The expanded set of possibilities provided by the stem-and-cycle structure is described in Section 2.3.3.)

Figure 8.3 illustrates the three types of starting moves for initiating the LK approach (2, 3 and 4-opt moves).

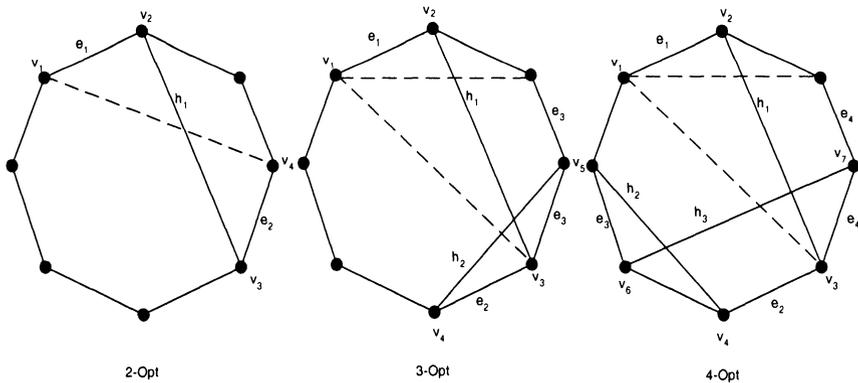


Figure 8.3. Possible moves at the first level of the Lin-Kernighan process

The initial k -opt moves ($k = 2, 3, 4$) that are used to launch the LK procedure can be created as follows, as illustrated by the diagrams in Figure 8.3, preceding:

$k = 2$ – The first step of the LK procedure selects the first edges $e_1 = (v_1, v_2)$ and $h_1 = (v_2, v_3)$ to drop and add (respectively), so that

they produce a cost improvement, and hence yield a negative value of $E_1 = c(v_2, v_3) - c(v_1, v_2)$. (Such a choice must be possible if a better tour exists.) A first level 2-opt move is obtained by dropping edge $e_2 = (v_3, v_4)$ where v_4 is in the tour neighbor of v_3 that is in the cycle $(v_2, v_3, v_4, \dots, v_2)$ created when edge h_1 was added. This last edge-drop operation implicit when h_1 is chosen creates a Hamiltonian path $H_1 = (v_1, \dots, v_3, v_2, \dots, v_4)$ that is used to close up the tour by adding edge (v_4, v_1) . Compute $T_1 = c(v_4, v_1) - c(v_3, v_4)$ and examine the solution cost change by computing $\Delta_1 = E_1 + T_1$.

$k = 3$ - Accompanying the initial drop-add choice that removes e_1 and adds h_1 , drop edge $e_2 = (v_3, v_4)$ where v_4 is adjacent to v_3 . There are two possibilities to create a 3-opt move: (1) if v_4 is in the cycle, the move is direct extension of the LK process from the level 1 to level 2; (2) if v_4 is the endpoint of the path from v_1 to v_4 , create a Hamiltonian path H_2 by linking v_4 to one vertex v_5 (corresponding to edge $h_2 = (v_4, v_5)$) and drop one of its adjacent edges $e_3 = (v_5, v_6)$ where v_5 is a vertex in the cycle (v_2, v_3, \dots, v_2) . Link v_6 to v_1 to create a tour. There are some subtleties in the way edge e_3 is chosen. When v_4 is in the path, the method selects v_5 by computing $v_5 = \operatorname{argmin}\{c(v_4, v_5) - \max\{c(v_5, v_{5-}), c(v_5, v_{5+})\}\}$. Once v_5 is chosen two trial values are examined to select v_6 by computing $v_6 = \operatorname{argmin}\{c(v_6, v_1) - c(v_5, v_6) : v_6 = v_{5-}, v_{5+}\}$. (The computation of v_6 in the way just defined is equivalent to finding the better of the two trial tours produced by adding the links (v_{5-}, v_1) and (v_{5+}, v_1) , as suggested in the original Lin and Kernighan paper.) The corresponding cost changes are given by $E_2 = E_1 + c(v_4, v_5) - c(v_3, v_4)$, $T_2 = c(v_6, v_1) - c(v_5, v_6)$, resulting in a tour cost change of $\Delta_2 = E_2 + T_2$.

$k = 4$ - As in the $k = 2$ and $k = 3$ cases, begin by dropping e_1 and adding h_1 , and then drop edge $e_2 = (v_3, v_4)$ where v_4 is adjacent to v_3 . There are two possibilities to create a 4-opt move: (1) if v_4 is in the cycle, the move is direct extension of the LK process from the level 2 to level 3; (2) if v_4 is the endpoint of the path from v_1 to v_4 , create a Hamiltonian path H_3 in two steps: (i) link v_4 to a vertex v_5 in the path (creating edge $h_2 = (v_4, v_5)$) and drop its adjacent edges $e_3 = (v_5, v_6)$ where v_6 is in the path (v_5, v_6, \dots, v_4) ; (ii) link v_6 to one vertex v_7 in the cycle (creating $h_3 = (v_6, v_7)$) and drop one of its adjacent edges $e_4 = (v_7, v_8)$. Link the endpoints of the resulting Hamiltonian path H_3 to create a tour. Edge e_3 is chosen by setting $e_3 = \operatorname{argmin}\{c(v_4, v_5) - \max\{c(v_5, v_{5-}), c(v_5, v_{5+})\} : v_6 = v_{5-}, v_{5+}\}$. Once e_3 is chosen, edge e_4 is selected by computing $e_4 = \operatorname{argmin}\{c(v_6, v_7) - \max\{c(v_7, v_{8-}), c(v_7, v_{8+})\} : v_8 = v_{7-}, v_{7+}\}$. (In this case vertices v_7 and v_8 are selected at the same time simply by choosing the largest cost edge incident at v_7 , which implies that only one trial tour will be examined as suggested in the original paper.) The corresponding

cost changes are given by $E_3 = E_1 + c(v_4, v_5) + c(v_6, v_7) - c(v_3, v_4)$, $T_3 = c(v_8, v_1) - c(v_7, v_8)$, and consequently the tour cost change is given by $\Delta_3 = E_3 + T_3$.

Extending LK moves to further levels. As mentioned, the regular LK search (which does not include the special 3-opt and 4-opt moves) consists of generating k -opt moves ($k \geq 2$) (as a sequence of 2-opt moves) in successive levels where at each level a new vertex is selected to extend the process to the next level by recovering a Hamiltonian path H_i (dropping the last link (v_{2i}, v_1)) and using this path as a reference structure to generate a new path H_{i+1} of the same type (by linking its endpoint to another vertex in the path and dropping one of its adjacent vertices). Starting with $E_0 = 0$, for a fixed vertex v_{2i} at a level i of the LK search, a new vertex v_{2i+1} is chosen in such a way that $E_i = E_{i-1} + c(v_{2i}, v_{2i+1}) - c(v_{2i-1}, v_{2i})$ yields a negative value. (An exception is made for some special alternative 3-opt and 4-opt moves as explained above.) Consequently, a trial move (that yields a trial tour) for the level i ($i > 0$) can be obtained by computing $T_i = c(v_{2i+2}, v_1) - c(v_{2i+1}, v_{2i+2})$. Similarly, the total tour cost change is given by $\Delta_i = E_i + T_i$. At each level i of the LK process, the method keeps track of the minimum Δ value (corresponding to the best trial tour) and records the sequence of the vertices associated with the e_i and h_i edges, which will be used to perform a move at the end of the LK search.

The LK procedure also considers a backtracking process which allows the method to repeat, each time choosing a different starting vertex associated with an untried alternative for inserting or deleting an edge $h_i = (v_i, v_{i+1})$ or $e_i = (v_{2i-1}, v_{2i})$, respectively. Such alternatives (which include the special 3-opt and 4-opt moves) are successively examined starting from level i and proceeding back to level 1, where the examination of edges h_i and e_i is alternated so that a candidate for edge e_i is examined after exhausting all possible candidates for h_i without finding any improvement of the starting tour. Candidates for level $i - 1$ are examined after exploring all alternatives for level i . As soon as an improvement is found at some level, the backtracking process stops and the LK process continues from that level (and its accompanying e_i or h_i choice), progressing forward again to higher levels as previously described. If no improvement occurs in the backtracking process, including those cases where alternative choices for the vertex v_1 are examined, the procedure stops and returns the best tour found so far. (In the original paper backtracking is used only for $i = 1$ and 2.)

A refinement of the basic LK method considers a look-ahead process where the choice of a current h_i edge takes into consideration the cost of the associated e_{i+1} edge. The evaluation criterion (via this “look-ahead” rule) is the one that minimizes $E_i = E_{i-1} + c(v_{2i}, v_{2i+1}) - c(v_{2i+1}, v_{2i+2})$ for a fixed vertex v_{2i} ($i > 1$), which gives the shortest Hamiltonian path for the next reference structure. Lin and Kernighan mention evaluating the tour length only for this choice, but of course there is little extra effort in evaluating the tour length for all choices and record the best for future reference (even if it is not the one that generates the next reference structure). Proceeding from this level a trial tour can be obtained at each additional level of the LK process by adding an edge (v_{2i+2}, v_1) which closes up the current Hamiltonian path adding the cost $T_i = c(v_{2i+2}, v_1)$. Again, the total tour cost change for the current level is given by $\Delta_i = E_i + T_i$.

Before providing the outline of the LK procedure we recall that a standard local search process involves solving a subproblem at each iteration. The solution space for this subproblem is implicitly defined by the neighborhood structure and the particular subset of available moves that is singled out to identify “reasonable choices” (and hence to restrict the solution space for this subproblem). Thus, it is relevant to keep in mind that the various minimization functions used in neighborhood search for TSPs assume the consideration of a neighbor list that restricts the number of choices for the vertices/edges involved in the moves. The original paper of Lin and Kernighan suggests a neighbor list made up of the 5 nearest neighbors of the node. However, other authors such as Johnson and McGeoch [463] claim better results for lists of the 20 nearest neighbors, and Applegate, Bixby, Chvatal, and Cook [32] use lists of different lengths at different levels of the search. In addition, Lin and Kernighan required that no added edges be subsequently deleted in the same k -opt move and no deleted edges be subsequently added. Johnson and McGeoch apply only the first of these two restrictions, which by itself is enough to insure that the search will have no more than n levels. A general outline of the Lin-Kernighan procedure is presented in Figure 8.4.

The Lin-Kernighan procedure introduces an important framework to generate compound moves. The wider class of variable depth methods known as Ejection Chains methods discussed in the next section shares several characteristics with this procedure. First, the creation of a reference structure (which in the LK procedure is implicitly given by the path H_1) makes it possible to create moves whose depth goes significantly beyond the one that can be handled in a practical manner

Step 1. Initialization

- (a) Generate a starting tour T .
- (b) Set $i = 1$. Choose for v_1 some vertex that has not taken this role since the last best tour was found.

Step 2. Choose e_1 and h_1 to initiate the LK search.

- (a) Set $e_1 = (v_1, v_2)$.
- (b) Select $h_1 = (v_2, v_3)$ such that $E_1 < 0$. If this is not possible, stop.

Step 3. Perform LK Search

- (a) Set $i = i + 1$. Choose $e_i = (v_{2i-1}, v_{2i})$ such that v_{2i} is in the cycle created when h_{i-1} was added.
- (b) Compute Δ_i and keep track of the e and h edges associated with the current Hamiltonian path H_i . Record the level i^* associated with the minimum Δ value found so far.
- (c) Choose $h_i = (v_{2i}, v_{2i+1})$ such that $E_i < 0$ and e_{i+1} exists. If such a h_i exists, go to Step 3.

Step 4. Backtraching for Levels 1 and 2

Perform 3-opt moves:

- (a) If there is at least one h_2 not examined, set $i = 2$ and go to Step 3(c).
- (b) If there is at least one e_2 not examined, choose $e_2 = (v_3, v_4)$ such that v_4 is in the path (v_1, \dots, v_4, v_3) .
- (c) Choose $h_2 = (v_4, v_5)$ and the associated adjacent edge $e_3 = (v_5, v_6)$ such that v_5 is in the cycle (v_2, v_3, \dots, v_2) .
- (d) Compute E_2 and Δ_2 . If Δ_2 is smaller than the best (least) Δ value found so far, update i^* and the new e and h edges considered in this Step. If $E_2 < 0$, set $i = 3$ and go to Step 3(c).

Perform 4-opt moves:

- (e) Choose $h_2 = (v_4, v_5)$ and the associated adjacent edge $e_3 = (v_5, v_6)$ such that v_5 is in the path (v_1, \dots, v_4) and e_3 is in the cycle (v_5, \dots, v_4, v_5) created when h_2 was added.
- (f) Choose $h_3 = (v_6, v_7)$ and the associated $e_4 = (v_7, v_8)$ where v_7 is in the cycle (v_2, v_3, \dots, v_2) and v_8 corresponds to the largest cost edge incident at v_7 .
- (g) Compute E_3 and Δ_3 . If Δ_3 is smaller than the best Δ value found so far, update i^* and the new e and h edges considered in this Step. If $E_3 < 0$, set $i = 4$ and go to Step 3(c).

Perform 2-opt moves:

- (h) If there is at least one h_1 not examined, set $i = 1$ and go to Step 2(b).
- (i) If there is at least one e_1 not examined, set $i = 1$ and go to Step 2(a).
- (j) If there is at least one v_1 not examined, go to Step 1(b). Otherwise Stop.

Figure 8.4. The General Lin-Kernighan Procedure

with classic k -opt neighborhoods. Second, the iterative characteristic of building the neighborhood to successive levels provides a form of “advance look-ahead” which leads to better choices. Finally, the evaluation of trial solutions throughout the neighborhood construction provides a way for the method to adaptively select the most appropriate level (or depth) of the move.

A fundamental drawback of the k -opt neighborhoods traditionally used, including the ones constructed in the basic Lin-Kernighan approach, is that the edges added and dropped are successively adjacent. These moves can be identified by numbering the vertices in the sequence determined by the added and dropped edges, noting that the last vertex is always connected by a tour edge to the first. We call these *sequential neighborhoods*, as opposed to *non-sequential neighborhoods* where such a successive adjacency requirement is not imposed. Sequential neighborhoods can be shown to generate a restricted instance of a classical alternating path, as introduced in graph theory by Berge [104].

Sequential neighborhoods can fail to find certain types of improved tours even if they are close to the current tour. This is illustrated in Figure 8.5, which depicts the so-called *double-bridge* as an example of a move in a non-sequential neighborhood. The tour reached by this move cannot be reached by means of any move within a bounded sequential neighborhood. Lin and Kernighan first identified this class of moves and suggested looking for improving double-bridge moves as a supplement to their variable-depth moves. Subsequently, Martin, Otto, and Felten [588] proposed using random double-bridge moves as a method for restarting the algorithm once a local optimum had been found, and variants on the resulting “Chained” (or “Iterated”) Lin-Kernighan algorithm have proved quite successful (Applegate et al. [27], Applegate, Cook, and Rohe [32], Helsgaun [446], Johnson and McGeoch [463]). A discussion of the performance of these approaches in practice is provided in Chapter 9.

2.3. Ejection Chain Methods

We have noted that the LK procedure relies on a Hamiltonian path as the basic reference structure to generate moves at each level of neighborhood construction. The fact that this structure has a configuration very close to a valid tour is convenient for visualization, but also constitutes a limitation of the procedure. More general Ejection Chain methods avoid this limitation by providing a wide variety of reference structures, which have the ability to generate moves not available to neighborhood search

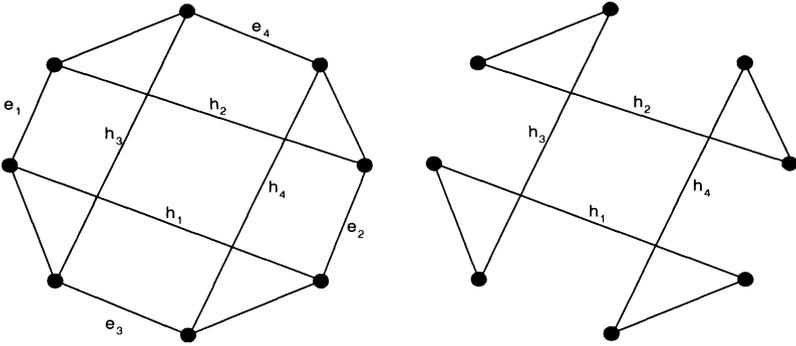


Figure 8.5. The double-bridge neighborhood.

approaches traditionally applied to TSPs. This section explores some of these ejection chain approaches and provides a framework for efficient implementations.

Ejection Chains are variable depth methods that generate a sequence of interrelated simple moves to create a more complex compound move. There are several types of ejection chains, some structured to induce successive changes in problem variables and others structured to induce changes in particular types of model components (such as nodes and edges of a graph). For a comprehensive description of ejection chain methods on graphs we refer the reader to Glover [371] and Glover ([372, 374]). Implementations and some extensions of these types of ejection chains for the TSP can be found in Pesch and Glover [667], Rego [704] and Glover and Punnen [382]. Applications of ejection chains to a cardinality-constrained TSP are discussed in Cao and Glover [158].

Ejection chains have also been successfully applied to combinatorial problems other than the TSP. For example, Dorndorf and Pesch [259] and Hubscher and Glover [372] use node-based ejection chains for clustering problems, while Laguna et al. [528] and Yagiura, Ibaraki and Glover [830] use ejection chains for the generalized assignment problem. Rego [706, 705], examines neighborhood structures based on node and subpath ejections to produce highly effective results for the vehicle routing problem. Finally, Cavique, Rego, and Themido [172] apply an ejection chain model to combine different types of moves for a real-world crew scheduling problem.

In this section we provide basic definitions and concepts of ejection chains, and then discuss some specialized ejection chain methods for the traveling salesman problem.

2.3.1 Ejection Chains Basics. Broadly speaking, an ejection chain consists of a succession of operations performed on a given set of elements, where the k th operation changes the state of one or more elements which are said to be *ejected* in the $k + 1$ th operation. This ejection thereby changes the state of other elements, which then lead to further ejections, until no more operations can be made according to pre-defined conditions. State-change steps and ejection steps typically alternate, and the options for each depend on the cumulative effect of previous steps (usually, but not necessarily, being influenced by the step immediately preceding).

In the ejection chain terminology, the order in which an element appears in the chain determines its *level*. The conditions coordinating the ejection chain process are called *legitimacy conditions*, which are guaranteed by associated *legitimacy restrictions*.

We focus on ejection chain methods for carrying out operations on graphs. The objective is to create mechanisms, namely neighborhood structures, allowing one solution subgraph to be successfully transformed into another.

In this context, relative to a specific graph G , an ejection chain of L levels consists of a succession of operations $m_1, \dots, m_k, \dots, m_L$ called *ejection moves*, where m_k transforms a subgraph G_k of G into another subgraph G_{k+1} by disconnecting (or ejecting) specified components (nodes, edges, subpaths) and relinking them to other components. The number of levels L is the *depth* of the ejection chain. The particular level chosen (from among the L levels generated to provide a move executed by a local search method) usually varies from one iteration to the next. The total number of levels L can likewise vary, and hence ejection chains fall within the class of so-called *variable depth methods*. In an ejection chain framework, the subgraph obtained at each level of the chain may not represent a feasible solution but may be transformed into a feasible solution by using a complementary operation called a *trial move*.

More formally, let S_i be the current solution at iteration i of the local search method, and let m_k, t_k be the ejection move and the trial move, respectively, at a level k of the chain. A neighborhood search ejection chain process consists of generating a sequence of moves $m_1, t_1, \dots, m_k, t_k, \dots, m_L, t_L$ on S_i such that the transition from solution S_i to S_{i+1} is given by performing a *compound move* $m_1, m_2, \dots, m_{k^*}, t_{k^*}$, where k^* represents the level associated with the highest quality trial solution visited during the ejection chain construction. In the ejection chain context we use the terms *compound move* and *transition move* interchangeably, to specify the move leading from one solution to another in an iteration of the local search procedure.

The effectiveness of such a procedure depends on the criterion for selecting component moves. More specifically, neighboring solutions obtained by an ejection chain process are created by a succession of embedded neighborhoods that lead to intermediate trial solutions at each level of the chain. However, the evaluation of ejection moves can be made independently from the evaluation of the trial moves, in which case trial moves are only evaluated after performing the ejection move at the same level of the chain. In this variant of the approach, the evaluation of an ejection move m_k only depends on the cumulative effect of the previous ejection moves, m_1, \dots, m_{k-1} , and is kept separate from the evaluations of trial solutions encountered along the way. The trial moves are therefore restricted to the function of finding the best trial solution that can be obtained after performing the associated ejection move.

We stress that our preceding description of ejection chain processes simply constitutes a taxonomic device for grouping methods that share certain useful features. The value of the taxonomy, however, is evidenced by the role it has played in uncovering new methods of considerable power for discrete optimization problems across a broad range of applications. Within the TSP setting, as will be seen, the ejection chain framework provides a foundation for methods that embrace a variety of compound neighborhood structures with special properties for combining moves, while entailing a relatively modest computational effort.

2.3.2 Node-based Ejection Chain Methods. Node-based ejection chain methods derive from extensions of customary single node insertion and node exchange neighborhoods found useful in several classes of graph problems including: machine scheduling, clustering, graph-coloring, vertex covering, maximum clique or independent problems, vehicle routing problems, generalized and quadratic assignment problem, and the traveling salesman problem, to cite only a few.

Since the worst case complexity of evaluating a single node-insertion and node-exchange neighborhood is $O(n^2)$, creating compound neighborhoods by combinations of these moves requires an effort that grows exponentially with the number of moves considered in the combination. More precisely, the best compound neighborhood of k moves can be generated and evaluated with $O(n^k)$ effort. This effort can be notably reduced by using appropriate candidate lists that we discuss in Section 3.1. Such lists also apply to several other types of neighborhood structures, including the ones discussed in this section.

We present here ejection chain methods to implement a *multi-node insertion* move and a *multi-node exchange* move that yield an important form of *combinatorial leverage*. Specifically, the number of moves

represented by a level k neighborhood is multiplicatively greater than the number of moves in a level $k - 1$ neighborhood, but the best move from the neighborhoods at each successive level can be determined by repeating only the effort required to determine a best first level move. In our application, for example, the moves of the first, second and third levels are respectively $O(n^2)$, $O(n^3)$, and $O(n^4)$ in number, but the best member of each can be found by repeating the $O(n^2)$ effort required to determine the best move of the first level, so the total effort is still $O(n^2)$. For a worst case analysis and proofs of the complexity of these ejection chain processes see Glover [371]. Here we focus on special properties for comparative analysis of different neighborhood structures and examine some implementation issues for improving algorithm performance.

Figure 8.6 illustrates a multi-node insertion produced by an ejection chain method. In the figure, a starting TSP tour is represented by the convex hull of the nodes, e_k denotes edges which are deleted at level k of the chain (and which identify the associated ejected nodes). Edges shown “inside” the starting tour are the ones that are added by the ejection chain process. To simplify the diagrams node labels are not used, but a node v_k is implicitly identified by the two adjacent e_k edges.

The ejection chain starts by identifying a node pair v_0, v_1 that yields the best (highest evaluation) ejection move that disconnects node v_0 from its current position and inserts it into the position currently occupied by node v_1 . Thus, a first level ejection move consists of adding edges (v_0, v_{1-}) , (v_0, v_{1+}) and deleting edges e_0 and e_1 . This creates an intermediate structure where node v_1 is temporarily disconnected from the tour. However, a trial move can be performed by creating edge (v_{0-}, v_{0+}) , and inserting node v_1 between nodes p_1 and q_1 , creating edges (v_1, v_{p_1}) , and (v_1, v_{q_1}) , and deleting edge t_1 . For the subsequent levels, ejection moves consist of selecting a new candidate node to be ejected by the previously ejected node, and so forth, until no other legitimate node exists for ejection.

This move is illustrated in the second level of the ejection chain shown in the middle diagram of Figure 8.6, where node v_1 ejects node v_2 by adding edges (v_1, v_{2-}) , (v_1, v_{2+}) and deleting edge e_2 . The trial move used in the first level is not considered for the construction of further levels of the chain. Instead, the ejection move generates a new move (of the same type) for the next level. A trial move is then executed as previously indicated, now by linking node v_2 to nodes p_2 and q_2 , and deleting edge t_2 . The corresponding level 2 trial solution is given in diagram on the right in Figure 8.6.

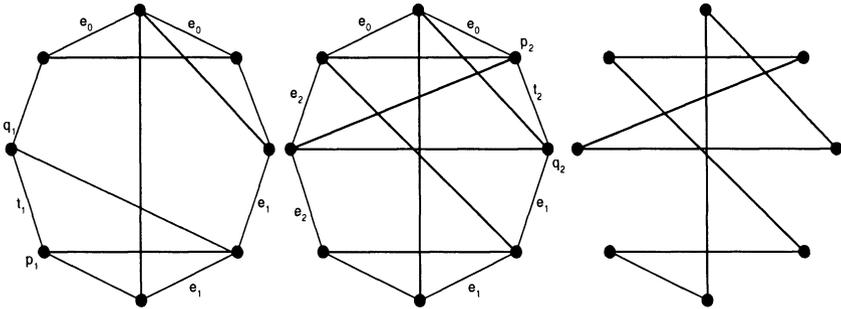


Figure 8.6. Two levels of a multi-node insertion ejection chain

A multi-node exchange move can be obtained at each level of the chain by considering a trial move that simply relocates the current ejected node to occupy the vacant position left by the node v_0 that initiates the chain. This is carried out by creating two edges (v_{0-}, v_k) , (v_k, v_{0+}) , where v_k denotes the node ejected at a level k of the chain.

In the multi-node insertion ejection chain method a trial move can be evaluated in time $O(n)$, but in the multi-node exchange method the move is evaluated in constant time, $O(1)$. Experiments with this ejection chain method for the vehicle routing problem (VRP) have shown that multi-node insertion is usually more efficient than multi-node exchange (Rego [706]). However, both types of moves can be efficiently combined in the same ejection chain process to select the best compound move at each level of the chain. Such an ejection chain produces a more complex neighborhood which dynamically combines insertion with exchange moves.

Figure 8.7 depicts this second type of ejection chain using the same ejection moves illustrated in Figure 8.6. Note that the first level of the chain is a standard single-node exchange move where nodes v_0 and v_1 exchange their positions. However, this exchange move produced by the ejection chain does not necessarily represent the highest evaluation two-node exchange move, unless we decide (for this first level) to evaluate the ejection move and the trial move conjointly. This decision is just a matter of preference since in this particular type of ejection chain either criterion can be evaluated in $O(n^2)$.

In the figure, level 1 and level 2 trial moves consist of adding edges (v_{0-}, v_1) , (v_1, v_{0+}) and edges (v_{0-}, v_2) , (v_2, v_{0+}) , respectively. Note that although edge (v_{0-}, v_2) has been deleted by the second ejection move

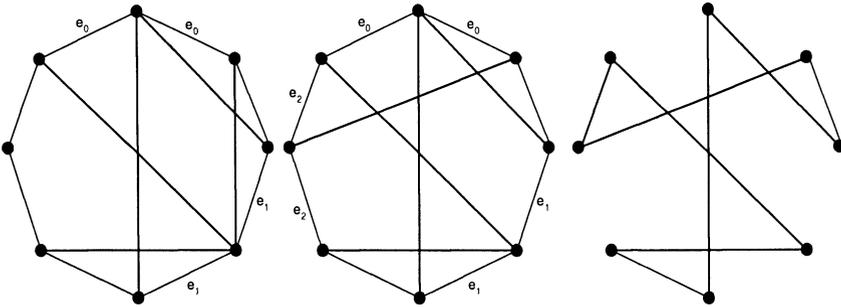


Figure 8.7. Two levels of a multi-node exchange ejection chain

it is added back by the associated trial move to create a tour from the intermediate structure.

In each of these methods, a *legitimate structure* for the ejection chain is defined by the requirement that each ejected node occurs only once in the chain. The preceding characterization of legitimacy implies that no edge will be added or dropped more than once by the transition move associated with the ejection chain, and consequently the tour cost change created by the transition move equals the sum of the added edges minus the sum of the dropped edges. These relationships are useful for the goal of efficiently generating and evaluating chains of progressively greater depth with a suitable combinatorial leverage.

Both types of node-based ejection chains can be completely determined by the succession of ejected nodes, $v_0, \dots, v_k, \dots, v_L$, which we designate by the set Z_L . Accordingly, we let Z_{L-} and Z_{L+} respectively denote the set of predecessors and the set of successors of vertices in Z_L , and let W_L denote the set of all the vertices involved in the ejection chain process, $W_L = Z_{L-} \cup Z_L \cup Z_{L+}$. Thus, the legitimacy restrictions consist of stipulating that each vertex in Z_L occurs only once in W_L . However, any vertex in Z_{L-} may reappear in Z_{L+} and vice versa, without violating this restriction.

An ejection chain of L levels can be recursively evaluated by computing the ejection values for these levels and summing them to give the trial value for each level. We denote a legitimate neighborhood for a node v_k in Z_k by $LN(v_k)$, thereby identifying a subset of nodes of G that do not violate the legitimacy restrictions. For convenience we denote the cost of two adjacent edges (v_i, v_j) and (v_j, v_k) as $c(v_i, v_j, v_k) = c(v_i, v_j) + c(v_j, v_k)$. Figure 8.8 provides a general neighborhood search procedure.

As shown in Glover [371] it is possible to create other variants of these ejection chain methods by growing the chain in the opposite direction.

Step 0. Initialization

- (a) Initialize a legitimate neighborhood for all vertices.
- (b) Denote the starting solution by S .
- (c) Set $k = 0$.

Step 1. Create the first level of the ejection chain

- (a) Determine a set of two initial vertices v_k, v_{k+1} by computing:
- (b) $E_k = \min\{c(v_{k-}, v_{k-1}, v_{k+}) - c(v_{k-1-}, v_{k-1}, v_{k-1+}) - c(v_{k-}, v_k, v_{k+}) + \lambda c(v_{k-1-}, v_{k-1+}) : v_i, v_j \in V\}$, where $\lambda = 1$ if multi-node insertion is used and $\lambda = 0$ otherwise.
- (c) Set $Z_k = \{v_k\}$.

Step 2. Grow the chain to further levels

- (a) Set $k = k + 1$ and set $Z_k = Z_{k-1} \cup \{v_k\}$.
- (b) Evaluate the trial tour cost for the current level by computing the value: $\Delta_k = E_k + \min\{c(v_p, v_k, v_{p+}) - c(v_p, v_{p+}) : v_p \in V \setminus Z_k\}$ if multi-node insertion is used. Otherwise compute $\Delta_k = E_k + c(v_{0-}, v_k, v_{0+})$.
- (c) Keep track of the best level k^* that produces the best trial tour.
- (d) Determine the new vertex $v \in LN(v_k)$ by computing: $E_k = E_{k-1} + \min\{c(v_{k-}, v_{k-1}, v_{k+}) - c(v_{k-}, v_k, v_{k+}) : v_i, v_j \in W_{k-1}\}$.
- (e) Set $v_{k+1} = v$.
- (f) Update the legitimate neighborhoods for each vertex $v_i \in W_{k^*}$.
- (g) If $k < L$ and $LN(v_k)$ is not empty, return Step 2. Otherwise go to Step 3.

Step 3. Perform the compound move

- (a) Apply to S the sequence of ejection moves up to the level k^* .
- (b) Complete the update of S by executing the trial move for the level k^* associated with multi-node insertion or multi-node exchange.

Figure 8.8. Neighborhood search iteration for node-based ejection chains.

Thus, for a multi-node ejection chain, the method starts by an insertion move which disconnects one node from its current position, followed by inserting it between two others. Then, the chain grows by selecting a node to fill the vacant position, which in turn opens a new “hole” for the next level of the chain. This technique is particularly relevant for using ejection chains to provide a *construction method* with attractive properties. A constructive multi-node insertion ejection chain method starts by choosing an initial single-node insertion move and making the

corresponding edge additions to generate a partial subgraph of the tour. Then, the subgraph is extended by adding one node (external to the current subgraph) to become the new v_0 node in the chain. The process is repeated until the partial subgraph becomes a spanning subgraph of G , thus corresponding to a TSP tour in G . The use of the ejection chain as a construction method always assures a legitimate TSP structure is produced. Since each new node v_0 is external to the current subgraph, it can not correspond to any of the spanning nodes of the ejection chain.

2.3.3 Generalizing Insertion and Exchange Ejection Chain Methods.

The foregoing ejection chain process can be easily extended to eject subpaths in addition to nodes. In its simplest form the procedure can be viewed as a generalization of the Or-opt neighborhood implemented in an ejection chain framework. A straightforward way to implement this generalized insertion ejection chain method is to collapse subpaths so they are essentially treated as nodes. (These collapsed subpaths are sometimes called *supernodes*.) Conversely, the method can implicitly create “holes” in subpaths and these can be possibilities for ejecting nodes inside of subpaths.

2.3.4 Subpath Ejection Chain Methods.

In a subpath ejection chain (SEC) the ejection moves involve the re-arranging of paths rather than individual nodes. One example is the variable depth search of the Lin-Kernighan procedure. In this section we discuss a potentially more powerful SEC method that forms the core of one the most efficient local search algorithms for the TSP, and whose performance is discussed in Chapter 9. The method is based on the *stem-and-cycle* (S&C) reference structure, which is a spanning subgraph of G that consists of a path $ST = \{v_t, \dots, v_r\}$ called a stem, attached to a cycle $CY = (v_r, v_{s_1}, \dots, v_{s_2}, v_r)$.

The first diagram of Figure 8.9 illustrates the creation of an S&C reference structure for the first level of the ejection chain process. The structure is created by dropping one edge in the tour (denoted by e_0) and linking one of the endpoints of the resulting Hamiltonian path to a different vertex in this path (denoted by v_r). Vertex v_r , which is the intersection of the stem and the cycle, is called the *root*. The two vertices in the cycle adjacent to v_r , denoted by v_{s_1} and v_{s_2} , are called *subroots*. The vertex v_t is called the *tip* of the stem.

The method starts by creating a stem-and-cycle reference structure from a TSP tour. Then, the method proceeds by performing an ejection move which links the tip node v_t to one of the remaining nodes v_p of

the graph, excluding the one that is adjacent to the tip. The root is considered as belonging to the cycle.

We differentiate two types of ejection moves depending on whether the operation is applied to the stem or to the cycle:

- (1) *Stem-ejection move*: add an edge (v_t, v_p) where v_p belongs to the stem. Identify the edge (v_p, v_q) so that v_q is a vertex on the subpath (v_t, \dots, v_p) . Vertex v_q becomes the new tip node.
- (2) *Cycle-ejection move*: add an edge (v_t, v_p) where v_p belongs to the cycle. Select an edge (v_p, v_q) of the cycle to be deleted where $v_q = v_{p-}$ or $v_q = v_{p+}$. Vertex v_q becomes the new tip node.

As with other types of ejection chains, an ejection move transforms an intermediate structure into another of the same type, which usually does not represent a feasible structure for the problem. The only exception is the degenerate case where the tip v_t is also the root v_r and hence the stem is of length 0 and the cycle is a tour. This can arise for instance as the result of a cycle-ejection move where v_p is a cycle-neighbor of v_r . Even though the root is fixed during one ejection chain search it is possible to change it whenever the degenerate case occurs.

In the general case of a non-degenerate S&C structure a feasible tour can always be obtained at each level of the chain by linking the tip node to one of the subroots and deleting the edge that links the subroot to the root node.

Figure 8.9 illustrates one level of the stem-and-cycle ejection chain process where edges that lie on the convex-hull of the vertex set are members of the initial tour and edges “inside” the tour are those added by the component ejection chain moves. We denote by e_k and d_k the edges deleted at level k by ejection and trial moves, respectively, and denote by t_k the tip node at level k . The S&C reference structure is created in the left-hand diagram by adding a link between two nodes in the tour and deleting one of the edges adjacent to either one of these nodes. Hence v_r becomes the root node with subroots s_1 and s_2 , and t_0 identifies the initial tip node. The middle diagram illustrates an example of a *stem-ejection move* which links t_0 to s_2 and deletes e_1 , thus making t_1 the new tip node. In the example, the associated trial move consists of adding the edge (t_1, s_1) and deleting edge (s_1, v_r) . Another possible trial move can be obtained by relinking t_1 to s_2 and deleting d_1 .

The right-hand side diagram illustrates a *cycle-ejection move* which links t_0 to v_p (in the cycle) and deletes e_1 . Again, two possible trial moves can be obtained by linking t_1 to one of the subroots and deleting the associated d_1 . A trial move can also be generated just after creating the

S&C structure. However, at this initial level only one trial move leads to a different tour, which in the example consists of adding edge (t_0, s_1) and deleting edge (s_1, v_r) . This restricted possibility yields the initial 2-opt trial move considered in the LK procedure. At each subsequent level, the two trial moves available by the S&C reference structure, and the enriched set of continuations for generating successive instances of this structure, provide a significantly enlarged set of tours accessible to the S&C approach by comparison to those accessible to the LK approach.

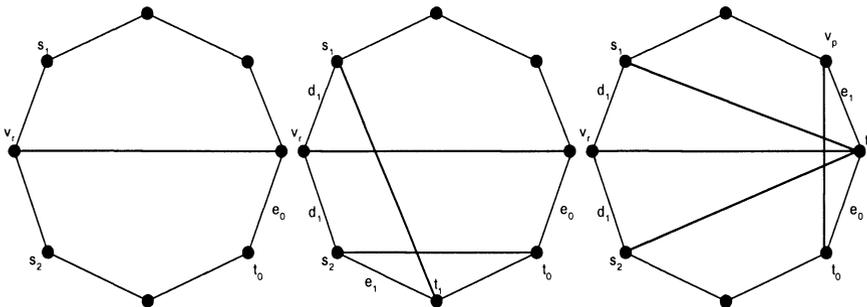


Figure 8.9. The stem-and-cycle ejection chain

In the design of the stem-and-cycle neighborhood search procedure legitimacy conditions can be added with two main purposes: (1) to prevent the method from visiting solutions already inspected during the ejection chain process; (2) to generate special forms of *alternating paths* which have proved useful in several classical graph theory problems. For the first purpose it is sufficient to stipulate that no deleted edge is added back during the construction of the chain. The second purpose deserves some additional consideration.

In classical alternating path methods in graph theory, and in neighborhood search processes related to them, the customary approach is to restrict the edges deleted to be edges of the starting solution. Methods that use this approach, which include the classical Lin-Kernighan procedure, may be viewed as *static alternating path methods*. However, certain neighboring solutions can not be obtained except by generating alternating paths in which previously added path edges are also candidates to be deleted. Thus, in contrast to classical approaches, this produces a *dynamic alternating path*. In fact, the paths provided by the S&C structure give the ability to reach any TSP tour from any other tour, in contrast to the situation illustrated earlier where the paths provided by the LK approach are unable to reach some tours that differ only

by 4 edges from the current tour. Moreover, as demonstrated in Glover [374], this ability can be assured by a simple “non-reversal” condition, which prevents an edge from being deleted if it is inserted immediately after deleting another edge that was previously inserted. These restrictions define the legitimacy conditions for the S&C algorithm described in Rego [704], and are also incorporated into an enhanced version of this algorithm reported in the 8th DIMACS TSP Implementation Challenge (Johnson, McGeoch, Glover, and Rego [462]).

A general design of the stem-and-cycle neighborhood search procedure can be described as in Figure 8.10, where we define a legitimate neighborhood for a node v_i , denoted by $LN(v_i)$, as the subset of nodes of G that do not violate the legitimacy restrictions identified above. Also, as shown in Rego [704] the maximum number of levels for a S&C ejection chain is bounded by $2n$, but since the best trial solution is usually found in a relatively lower level, L is considered a user-supplied parameter.

2.4. New Methods for Variable Depth Search

We have illustrated how ejection chain methods can be useful to generate compound neighborhood structures of several types, encompassing a variety of special properties for the traveling salesman problem. As previously mentioned, this framework for generating neighborhoods has proved highly effective for exploring the solution space in several other hard combinatorial problems. However, we recall that ejection chains characteristically generate moves that can not be obtained by neighborhoods that preserve feasibility at each step. We now discuss methods to efficiently combine other, more customary, neighborhoods based on the creation of appropriate candidate lists, which likewise can easily be exploited by parallel processing.

In the context of the TSP the terms *candidate lists* and *neighbor lists* are often used interchangeably because the restricted sets of elements considered for evaluation are usually defined by the relative distance between nodes in the problem data space. We will examine several types of neighbor lists in Section 3.1. However, for the exposition of the methods discussed in the present section, it is important to clearly differentiate neighbor lists from candidate lists. While both lists are associated with strategies to reduce the neighborhood size, neighbor lists constitute simply one possibility for creating candidate lists, and in many settings are used to initialize a more general candidate list approach. Neighbor lists are static, and keep neighbor elements from changing during the search process. (Methods that temporarily change the problem data

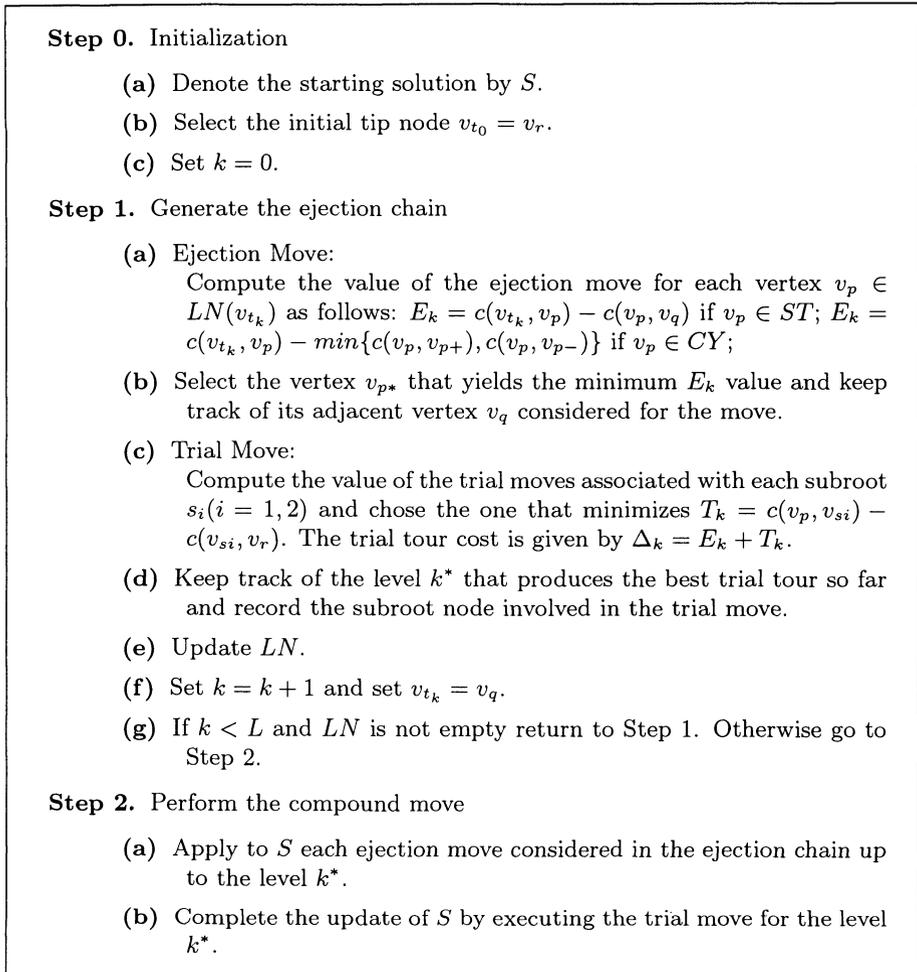


Figure 8.10. An Iteration of the Stem-and-Cycle Procedure

such as *space smoothing* (Gu and Huang [409], Steven et al. [774]) and *noising methods* (Charon and Hudry [179, 180]), can change the static relationship, however.) Conversely, candidate lists do not necessarily rely on the problem data but rather are made up of solution attributes. These attributes, which can include the types of elements used in the neighbor lists and many others, change dynamically, based on information gathered from the search process. Candidate lists have been chiefly proposed in association with tabu search, which exploits strategic information embodied in memory structures. In such settings the changes in the candidate lists are represented within an *adaptive memory program-*

ming implementation either by explicitly replacing some attributes with others or by changing values of these attributes.

We next discuss candidate lists that offer a useful basis for creating compound moves within exponential neighborhoods, while using an economical degree of effort. For illustration purposes we consider standard single-node insertion moves as a basic element to create more complex moves.

2.4.1 The Sequential Fan Method. This Sequential Fan method may be viewed as a natural generalization of *beam search*, an approach that is extensively used in scheduling. (See Morton and Pentico [609], for a survey of beam search and its applications.) Beam search is applied within sequential construction methods as a restricted breadth-first tree search, which progresses level by level, without backtracking, but only exploring the most promising nodes at each level. As a schedule is constructed, beam search progressively truncates the tree by choosing a parameter (the "beam width") that determines a constant number β of nodes (partial solutions) at each depth from the root that are permitted to generate nodes at the next depth. A variant called *filtered beam search* (Ow and Morton [638]) refines this approach by using a two-step evaluation to choose the β best moves at each level. The method first picks some number δ (the "filter width") of locally best moves, and then submits these to more rigorous examination by extending each one in a single path look-ahead to the end of the tree (choosing the locally best move at each step of the path). The total path evaluations are used to select the β best moves from the initial δ candidates.

By extension, the sequential fan method operates in the local search setting as well as in the sequential construction setting, and works with complete solutions in neighborhood spaces as well as with the types of partial solutions used by beam search. It also more generally varies the width of the search strategically as a function of both the depth and the quality of the solutions generated. In a simple application to the TSP, for example, moves that interchange their current positions in the tour can be checked to identify a few good options, and for each of these, follow-on options are checked by pruning the total number of alternatives that are tracked at each level according to quality and depth.

The basic construction of the Sequential Fan tree underlies the use of a candidate list strategy based on weeding out promising moves by applying evaluations in successive levels of the tree search, where the moves that pass the evaluation criteria at one level are subjected to additional evaluation criteria at the next. The basis for the creation of the *sequential fan candidate list strategy* can be described as follows. A

list of moves $M(k)$ is associated with each level k , where list $M(k)$ is derived by applying criterion k to evaluate the moves on list $M(k - 1)$. To start, list $M(1)$ is created from the set of all available moves or from a subset determined by another type of candidate list (e.g. a *neighbor list* as commonly used in the TSP setting) and contains the α_1 best of these moves by criterion 1. List $M(2)$ then contains the α_2 best of the moves from $M(1)$ according to criterion 2, and so on.

More generally, subsequent lists may not merely contain moves that are members of earlier lists, but may contain moves derived from these earlier moves. A useful way to make successive refined evaluations is to employ a deeper look-ahead in subsequent layers. For example, list $M(1)$ may apply criterion 1 to evaluate immediate moves, while $M(2)$ may apply criterion 2 to evaluate moves from the solutions produced by $M(1)$ to create *compound moves*. More advanced constructions of this look-ahead process may be conceived by the use of ejection chain processes (performed from nodes at the current level) as a foundation to determine promising component moves to dynamically update the candidate list. Also, high evaluation trial solutions found throughout the ejection chain can be recorded for further consideration, as we discuss in Section 4.3.

2.4.2 The Filter and Fan Method. The Filter and Fan method (F&F) is a combination of the filtration and sequential fan candidate list strategies used in tabu search.

By our earlier conventions, a compound move is one that can be decomposed into a sequence of more elementary component moves (or submoves), and the best compound move is the best (highest evaluation) combination of these submoves. As we have seen, a complete evaluation of simple node-insertion and node-exchange moves in dense TSPs requires $O(n^2)$ effort, and the effort of evaluating a combination of L of these moves is $O(n^L)$, and hence grows exponentially with L . However, this effort can be notably reduced based on the assumption that the best L -compound move is a combination of L submoves such that each is one of the $M(k)$ highest evaluation moves for the corresponding level k of the tree ($k = 1, \dots, L$). Thus, instead of evaluating all possible combinations of k moves the F&F method proceeds by progressively creating new solutions for a level k ($k > 0$), which derive from the solutions generated in the level $k - 1$ by applying a restricted subset $A(k)$ of the highest evaluation moves, selected from a larger set $M(0)$ of potentially good moves, $|M(0)| = \eta_0$.

The Filter and Fan Model. The F&F model can be viewed as a *neighborhood tree* where branches represent submoves and nodes identify solutions produced by these moves. An exception is made for the root node, which represents the starting solution to which the compound move is to be applied. The maximum number of levels considered in one sequence defines the depth of the tree. The neighborhood tree is explored level by level in a breadth search strategy. For each level k , the method generates $\eta_1 * \eta_2$ moves by the *fan candidate list strategy*, then a subset $M(k)$ of η_2 moves is selected by the *filter candidate list strategy* to generate the solutions for the next level.

An illustration of the Filter and Fan model is provided in Figure 8.11, where black nodes identify a local optimum with respect to the L -neighborhood. The method starts as a standard descent method by performing 1-moves as long as they improve the best current solution. Once a local optimum is found (in the descent phase) the best $M(0)$ moves (among the M moves evaluated to establish local optimality) are used to create the first level of the F&F neighborhood tree. The next levels are created as follows. Letting η_1 be the number of $M(k)$ moves for level k , the method proceeds by selecting a subset $A_i(k)$ of η_2 moves from $M(0)$ associated with each solution $X_i(k)$ ($i = 1, \dots, \eta_1$) to generate $\eta = \eta_1 * \eta_2$ trial solutions for the level $k + 1$ (as a result of applying η_2 moves to each solution at level k). For convenience we consider $\eta_1 = 4$ and $\eta_2 = 2$ for the example illustrated in Figure 8.11. (The process of selecting η_2 moves has to obey to a set a legitimacy conditions that will be specified later.)

We define $A(k) = \{A_1(k), A_2(k), \dots, A_{\eta_1}(k)\}$ ($|A_i(k)| = \eta_2$) as the set of η moves evaluated at the level k from which the set $M(k) = \{m_{1k}, m_{2k}, \dots, m_{\eta_1 k}\}$ is selected, $M(k) \subset A(k)$, $k > 0$. The process is repeated by creating a set $X(k + 1)$ of solutions obtained by applying $M(k)$ moves to the associated solutions in $X(k)$ and keeping these solutions as starting points for the next level of the F&F tree.

For the purpose of illustration we consider the *fan candidate list strategy* to be the one that identifies the best η_2 component moves for each solution at a level k , and the *filter candidate list strategy* to be the one that identifies a subset of η_1 of the η moves generated. Also, our example constitutes a variant in which the method stops branching as soon as an improved solution is found, then switches back to the descent phase starting with this new solution. However, other strategies to create both types of candidate lists are possible.

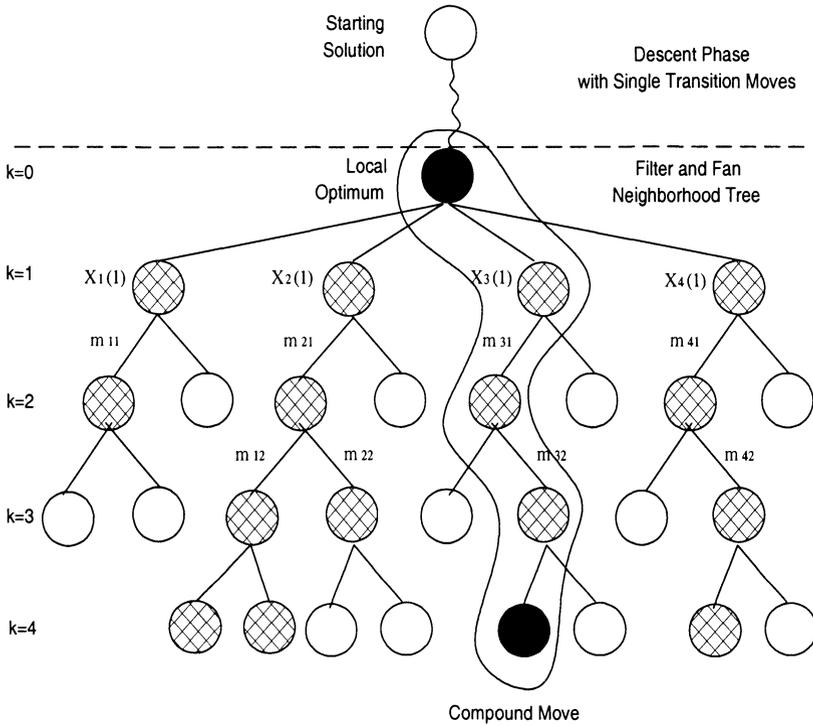


Figure 8.11. The Filter and Fan Model

More elaborate designs of the F&F method allow different types of moves for combination at each level of the tree, so that compound moves can be obtained by different neighborhoods applied under appropriate legitimacy conditions. By continuing the tree search after a local optimum is found, local optimality is overcome in “intermediate” levels of the F&F tree. Then the best trial solution encountered throughout the tree is chosen to re-initiate the descent phase.

More advanced versions result by replacing the descent phase with a tabu search phase, which, for example, can activate the F&F strategy based on the use of critical event memory. Thus, the F&F strategy can be used either to intensify the search in regions of high quality (elite) solutions or to diversify the search by propelling the method to a different region of the solution space.

Generating legitimate multi-node insertion moves by an F&F strategy. In order to create legitimate trial solutions when applying the F&F method legitimacy conditions have to be defined according to the type of component move used for the problem. We characterize legitimacy conditions for an F&F method using single-node insertion component moves for the TSP.

A component move will be called *legitimate* at a level k if this move can be performed by the usual neighborhood search rules (e.g. as a customary node insertion) after performing the associated $(k - 1)$ -move. Otherwise, the move is *illegitimate*. By this definition, a move that is illegitimate relative to a solution $X_i(k)$ ($1 \leq i \leq \eta_1$) will remain illegitimate throughout further levels of the subtree rooted by $X_i(k)$.

We further stipulate that the legitimacy conditions ensure the component move evaluations do not change during the F&F neighborhood search. Thus, the solution cost-changes associated with each move in M are carried forward through the tree to provide information for evaluating the $A(k)$ moves. By doing so, the neighborhood of a solution $X_i(k)$ can be restricted to consist of η_2 potentially good moves. The $M(k)$ moves ($k > 0$) are chosen according to the quality of the trial solutions produced by the $A(k)$ moves.

Consider an F&F process based on single node-insertion moves, which insert a node v_i between two consecutive nodes v_p and v_q in the tour. To maintain the legitimacy of an L -move it is sufficient to forbid the insertion of a node v_i between nodes for which the corresponding edge (v_p, v_q) has been deleted in one of the $L - 1$ levels of the corresponding L -move.

Additional considerations for implementation. An efficient implementation of the F&F procedure requires the identification of appro-

priate data structures for handling different parts of the method and speeding up the search.

The first issue in implementing the F&F method concerns the creation of $M(0)$ in the descent phase. Assume the simplest form of the F&F strategy is employed, where the initial phase is a pure memoryless descent procedure. Hence $M(0)$ is a subset of the best M moves evaluated on the last step of the descent (to verify that local optimality has been reached). It may be computationally more efficient to create $M(0)$ by performing an additional iteration of the local search procedure after reaching the local optimum S^* , rather than to keep track of the η_0 best moves at each iteration. A priority queue based on a binary heap data structure can be used to identify the best η_0 moves during the neighborhood search process in $O(\log(\eta_0))$ time. (See, e.g., Cormen, Leiserson, and Rivest, [218], pages 140-152.) Since the additional iteration consists of repeating the evaluation of moves in the previous iteration, several strategies can be used to restrict the size of the neighborhood, thus reducing the time complexity to create $M(0)$.

Another issue concerns the creation of $A_i(k)$ for each solution $X_i(k)$. Instead of searching $M(0)$ for the best η_2 legitimate moves it can be advantageous to consider the m_{jk} moves ($j = 1, \dots, \eta_{1k}, j \neq i$) as the candidates for $A_i(k)$. The creation of this candidate list assumes that good moves in one level of the tree are potentially good in deeper levels of the tree. However, such a strategy increases the chance for re-generating solutions previously visited. One way to counter this tendency is to use a tabu list of move attributes associated with each solution $X_i(k)$, thus introducing a further level of legitimacy. Additional moves to complete $A_i(k)$ can be examined in $M(0)$ whenever the number of legitimate moves for $X_i(k)$ is smaller than η_2 . An outline of the general F&F procedure is provided in Figure 8.12.

3. Tabu Search

The Tabu Search (TS) metaheuristic has proved highly successful in solving a wide range of challenging problems. A key feature of TS is its use of adaptive memory, accompanied by a collection of strategies for taking advantage of this memory in a variety of contexts. Characteristically, TS can be implemented at multiple levels to exploit tradeoffs between ease of implementation and sophistication of the search. Simpler forms of TS incorporate a restricted portion of its adaptive memory design and are sometimes applied in preliminary analyses. These approaches have proved useful for testing the performance of a limited subset of TS components, and for identifying cases where more fully in-

- Step 0.** Generate a candidate list of component moves
- (a) Consider a starting solution S and perform 1-moves using a local search method until a local optimum S^* is found.
 - (b) Create a candidate list $M(0)$ with the η_0 highest evaluation moves in the neighborhood where S^* was found.
 - (c) Apply the best η_1 moves in $M(0)$ to S^* to create the first level of the F&F tree with solutions $X_i(1)(i = 1, \dots, \eta_1)$. Set $k = 1$.
- Step 1.** Generate the Filter and Fan tree
- (a) Identify the best η_2 legitimate moves derived from $M(0)$ for each solution $X_i(k)(i = 1, \dots, \eta_1)$ to create sets $A_i(k)(j = 1, \dots, \eta_1)$.
 - (b) Evaluate each move in $A_i(k)$, applied to the associated solution $X_i(k)$, and compute the value of the corresponding trial solution.
 - (c) If the best trial solution found is better than S^* , perform the associated move from $X_i(k)$ on S and go to Step 0.
 - (d) Otherwise, select the $A(k)$ moves that led to the best η_1 trial solutions to become the members of $M(k)$.
 - (e) Apply the $M(k)$ moves to the corresponding solutions $X_i(k)$ to create $X(k + 1)$.
 - (f) If $k = L$ stop. Otherwise set $k = k + 1$ and repeat Step 1.

Figure 8.12. A General Filter and Fan Procedure

egrated strategies are not required. However, versions of tabu search that include a more comprehensive and advanced set of its elements generally prove superior to more limited versions of the approach.

A strategic component of TS that is sometimes omitted involves maintaining and analyzing a collection of high quality solutions to infer properties of other high quality solutions. Such processes provide a connection between Tabu Search and certain evolutionary approaches, as represented by the Scatter Search method discussed in the next section.

So far algorithmic studies of large TSP instances have chiefly focused on isolating efficient neighborhood structures (such as those based on Lin-Kernighan and Ejection Chain procedures) and on using appropriate candidate lists. As reported in the 8th DIMACS TSP Implementation Challenge, recent implementations of LK and EC procedures can now find near-optimal solutions for very-large scale TSP instances in a relatively short time.

Motivated by the experiences reported in other problem settings we speculate that still better TSP solutions may be found by including advanced features of tabu search. In this section we discuss some key

strategies in TS that deserve special consideration to achieve improved outcomes.

3.1. Candidate List Strategies

As we have already emphasized, efficient procedures for isolating good candidate moves are critically important for the efficiency of local search algorithms. In the TSP setting, for example, the use of candidate lists is mandatory when large instances have to be solved.

There are some subtleties in the ways candidate list strategies may be used. A number of studies have observed that optimal or near optimal solutions often can be constructed for the TSP by limiting consideration to a small number of shortest (least cost) arcs out of each node. A natural procedure is to create a candidate list defined by the nearest neighbor graph, giving the neighbor list previously discussed, where some limited number of nodes closest to each given node determines the edges permitted to be included in the tours generated. However, TSP instances exist where the best solutions significantly violate this restriction, as exemplified by the situation where vertices on a Euclidian plane occur in separated clusters. A drawback of the nearest neighbor list is the fact that its size is fixed and it does not exploit the geometric structure of the problem. Consequently, more efficient approaches create moves by requiring some but not all edges to belong to a shortest-edge collection. Reinelt [710] suggests a candidate list approach based on the computation of a Delaunay graph, which provides a set of edges to initialize the candidate list. Then the list is expanded by adding an edge (v_i, v_k) for each pair (v_i, v_j) and (v_j, v_k) in the initial set. It has been observed that even if this approach provides useful edges for clustered TSP instances it misses several other important edges and thus restricts the performance of the local search procedure.

A more efficient candidate list construction is the so-called k -quadrant neighbor graph, initially proposed by Miller and Pekny [598] for a 2-matching problem (which is a relaxation of the TSP) and first used by Johnson and McGeogh [463] in the TSP context. In this graph, each vertex v_j is the origin of a quadrant in a Euclidian plane and the $k/4$ vertices closest to the origin in each quadrant define the neighbors for vertex v_j . Let q_{ij} denote the number of vertices in the quadrant i for vertex v_j . If $\sum_{i=1}^4 q_{ij} < k$, then we fill out the candidate list for v_j with the $k - \sum_{i=1}^4 q_{ij}$ nearest cities to v_j not already included. This candidate list is used in several of the most efficient implementations of local search algorithms submitted to the DIMACS TSP Challenge, including imple-

mentations of the Lin-Kernighan and Ejection Chain algorithms. A more sophisticated approach is used in Helsgaun's variant of Lin-Kernighan [446], where the candidate list for v_i consists of its k nearest neighbors v_j under a new metric produced in a two-step derivation process from the original distances (see Chapter 9).

We conjecture that the design of efficient candidate lists for these hard TSP instances (where vertices are not uniformly diffused but may clump and cluster) depend in part on their *amplification factor* [371], which is the ratio of the total number of arcs added by the move to the number that belong to the k -shortest category. For a simple example, consider single node insertion and exchange moves. Requiring that a "first added edge" in each of these moves must be on a nearest neighbor list) instead of requiring that all added edges belong to such lists) will achieve amplification factors of 3 and 4 respectively. The logical conditions defining such a candidate list in the present context can be specified more precisely as follows. For an insertion move, where a selected node v_i is repositioned to lie between nodes v_p and v_q , we require one of these two added edges (v_p, v_i) or (v_i, v_q) to be among the k shortest arcs linked to node v_i . Since three edges are added by the move (including the arc joining v_{i-} to v_{i+}), this single-arc requirement gives an amplification factor of 3. (More than one of the three added edges may belong to the k -shortest category, but only one is compelled to do so.) Given node v_i , the form of the insertion move is completely determined once either edge (v_p, v_i) or (v_i, v_q) is specified. Similarly, for exchange moves, where nodes v_i and v_j interchange positions, we require only one of the four added edges (v_{i-}, v_j) , (v_j-, v_i) , (v_i, v_{j+}) , (v_j, v_{i+}) to belong to the k -shortest group, yielding an amplification factor of 4. Here, a given added edge can be used to define two different moves. By extension, the subpath insertions and exchanges of the type described in the ejection chain method provide a means for achieving significantly higher amplification factors.

The features attending these cases are characteristic of those exhibited by a special type of candidate list proposed in tabu search called a *preferred attribute candidate list*. In this instance, the k shortest edges of a node v_i identify the "preferred attributes" used to control the construction of moves where each attribute (or attribute set) on the list exhibits a property expected to be found in good solutions. For the present setting, these candidate lists can be constructed as follows.

Consider first the case of insert moves where each preferred arc (v_i, v_j) generates two candidate moves: the first inserting v_i between v_j and v_{j+} , and the second inserting v_j between v_i and v_{i+} , excluding the case where (v_i, v_j) is an edge of the tour. Since we are dealing with the symmetric

case, the preferred edge (v_i, v_j) generates two insert moves in addition to those indicated. The first inserts v_i between v_j and v_{j+} , and the second inserts v_j between v_{i-} and v_i . The preferred attribute candidate list for exchange moves is similarly constructed. Each preferred edge (v_i, v_j) generates four candidate exchange moves, the first exchanging v_j with v_{j+} , the second exchanging v_i with v_{i-} , and two others that result by treating a preferred edge in its two equivalent forms of (v_i, v_j) and (v_j, v_i) . Note that the generalization of these constructions for multi-node insertion and exchange moves of the type considered by Or-opt neighborhoods is straightforward.

We suggest that fuller advantage can be gained from the preferred candidate list by replacing the costs $c(v_i, v_j)$ by non-negative reduced costs derived by solving 1-tree relaxation of the TSP. This will not change the move evaluations, but typically will change the identities of the k -shortest edges of each node. (Ties can be broken by reference to the original costs.) Additional shortest edges may be included as determined by “modified” reduced costs, where constraints violating the node degree are plugged in a Lagrangian function to amend the 1-tree structure.

In addition to the design of candidate list strategies, a careful organization that saves the results of evaluations from previous iterations rather than re-computing them from scratch, can also be valuable for reducing time. Time saved in this way allows a chance to devote more time to the search. In the TSP setting this objective has been chiefly achieved by the use of the so-called *don't-look bits* strategy introduced by Bentley [103]. This strategy is based on the observation that if the base vertex, e.g. v_1 in the LK procedure, and if the “tour neighbors” of this vertex have not changed since that time, it is unlikely that the selection of this vertex will produce an improving move. Thus, by associating a binary variable (or flag) with each vertex, the neighborhood is restricted to moves for which the base vertex v_1 's associated bit is turned off. A bit for a vertex v_i is turned on the first time the selection of this vertex does not produce an improving move. Conversely, it is turned off when one of its adjacent vertices is used for a move.

3.2. Intensification and Diversification Strategies

Intensification and diversification in tabu search underlie the use of memory structures which operate by reference to four main principal dimensions: *recency*, *frequency*, *quality*, and *influence*. The strategic

integration of different types of memory along these dimensions is generally known as *adaptive memory programming*.

Elements of memory can refer to both *attributive memory* and *explicit memory*. Attributive (or "Attribute-based") memory refers to either basic or created attributes – instead of recording complete solutions – as a way to generate strategies to guide the search. Attributive memory records information about solution attributes that change in moving from one solution to another. For example, in the TSP setting, attributes can consist of nodes or arcs that are added, dropped or repositioned by the moves executed. (In more abstract problem formulations, attributes may correspond to values of variables or functions.) Sometimes attributes are also strategically combined to create other attributes, as by hash functions or by chunking or "vocabulary building" methods ([379]). Tabu search also uses explicit memory (complete solutions), usually by recording a limited set of elite solutions which are analyzed to determine relationships between attributes in these solutions.

Broadly speaking, recency/frequency and quality/influence can be viewed as complementary dimensions. Recency-based and frequency-based memory record timing information about the use of specific memory elements while quality and influence classify solutions in terms of their significance for representing promising solution characteristics (or regions in the solution space) and the impact of certain choices on the quality of the solutions produced. The time span considered in recency-based and frequency-based memory gives rise to an important distinction between *short-term memory* and *longer-term memory*.

The short term memory component of tabu search, which is the starting point for many tabu search implementations, characteristically embodies a recency-based memory that modifies the solution trajectory by *tabu restrictions* (or *conditions*) and *aspiration criteria*. A tabu restriction prevents a particular solution, or set of solutions, from being chosen as the outcome of the next move. Most commonly used short term memory keeps track of solution attributes that have changed during the recent past. Recency-based memory is exploited by assigning a tabu-active designation to selected attributes that contribute to creating a tabu restriction. This prevents certain solutions from the recent past from belonging to the admissible neighborhood of the current solution and hence from being revisited. The process implicitly subdivides solutions into changing "similarity classes" whereby all solutions that share tabu-active attributes with solutions recently visited may likewise be prevented from being visited. Aspiration levels provide a supplementary basis for controlling the search, by allowing a move to be selected if the resulting solution is sufficiently good or novel, in spite of being classified

tabu-active. A simple aspiration criterion is to allow a tabu move to be selected if it leads to a solution better than the best one seen so far, or the best one seen within a particular region or duration of search. Advanced forms of short-term memory may consider various types of tabu restrictions associated with several aspiration criteria, which may be used in conjunction to make a decision about the declination of the tabu status of a particular move.

In the TSP context, tabu restrictions may be created, for example, by (1) preventing a dropped edge from being subsequently added back; (2) preventing an added edge from being subsequently dropped; (3) preventing a move that simultaneously adds a previously dropped edge and drops a previously added edge. Since there are generally fewer edges that can be dropped than can be added, a tabu restriction of type (1) allows a greater degree of flexibility than a restriction of type (2) or (3). (Still greater flexibility is provided by a restriction that prevents a dropped edge from being added back only if the move simultaneously drops a previously added edge.)

Tabu restrictions remain in operation for a certain number of iterations (the tabu tenure) which can vary according to the solution attributes involved and the current search state. In some implementations where all attributes receive the same tenure, the tabu restrictions are activated by placing the attributes on a *tabu list*, and the size of this list identifies the tenure. (An attribute whose tenure expires is removed from the list at the same time that a new attribute is added.) A first level of intensification and diversification can be achieved by changing the tabu list size. Small sizes encourage the exploration of solutions near a local optimum, and larger ones push the search out of the vicinity of the local optimum. Varying the tabu list size during the search provides one way to explore such an effect, which has proved useful in a number of tabu search applications.

A common means of implementing this type of short term memory is to create an array which records the iteration that an attribute becomes tabu-active. Then, the attribute remains tabu-active as long as the current iteration does not exceed the initial iteration value by more than the tabu tenure. A special type of tabu list results by creating “coded attributes” using hash functions. Such a representation may be viewed as a *semi-explicit memory* that can be used as an alternative to attributive memory. One variant of such an approach is a special form of tabu search known as *reactive tabu search* (Battiti and Tecchioli [91]). The goal of this technique is to differentiate more precisely among individual solutions, by making use of a fine gauge attribute memory. (Only individual solutions can pass through the mesh, if the hashing

is highly effective.) Other TS approaches usually incorporate broader gauge attribute definitions, which implicitly differentiate among subsets of solutions rather than individual solutions. In reactive TS, when the search appears to revisit a particular solution (by encountering its coded attributes) too often, the method introduces a diversification step to drive the solution into a new region.

Frequency-based memory provides a type of information that complements the information provided by recency-based memory. Frequency-based memory has two forms: *transition frequency memory* and *residence frequency memory*. Transition frequency relates to the number of times an attribute enters or leaves the solutions generated (as, for example, the number of times an edge is added or dropped). Residence frequency relates to the number of iterations during which an attribute belongs the solutions generated (as, for example, the number of iterations an edge belongs to a TSP tour, considering all tours generated). Frequency based memory can also be different according to the interval (or intervals) of time chosen for the memory. Frequency based memory that is applied only to elite solutions gives different information and is used in different ways than frequency based memory that is applied to all solutions (or "average" solutions). These memories are sometimes accompanied by extended forms of recency-based memory.

Intensification is sometimes based on keeping track of the frequency that attributes (assignments of elements to positions, edges of tours, fairly narrow ranges of value assignments, etc.) occur in elite solutions, and then favoring the inclusion of the highest frequency elements so the search can concentrate on finding the best supporting uses (or values) of other elements.

As part of a longer term intensification strategy, elements of a solution may be selected judiciously to be provisionally locked into the solution on the basis of having occurred with a high frequency in the best solutions found. In that case, choosing different mutually reinforcing sets of elements to be treated in such a fashion can be quite beneficial. In the TSP setting where typically good solutions have many elements in common, edges that belong to the intersection of elite tours may be locked into the solution, in order to focus the search on manipulating other parts of the tour. This creates a *combinatorial implosion* effect (opposite to a *combinatorial explosion* effect) that shrinks the solution space to a point where best solutions over the reduced space tend to be found

more readily. Such an intensification approach, where restrictions are imposed on parts of the problem or structure is a form of *intensification by decomposition* proposed in tabu search.

The backtracking mechanism used in the Lin-Kernighan procedure may be viewed as a simple type of intensification process that attempts to find a new improving solution by jumping back to successive trial solutions examined in the first steps of the current Lin-Kernighan iteration. This is a limited form of intensification in the sense that elite solutions chosen to restart the method are restricted to those encountered at the immediately preceding level and therefore are very similar to one another. In fact, since the backtracking process is only applied when no improving solution is found during the LK move generation, backtracking may be viewed as a perturbation mechanism locally applied to the last local optimum found (and therefore limited to the exploration of one elite solution at a time). The reverse extreme of this technique is the process of restarting the method from a new initial solution generated either randomly or by varying parameters of a constructive procedure. This represents a primitive form of diversification, without reference to memory to preserve promising characteristics of the elite solutions visited so far or to compel the generation of solutions that differ in specific ways from those previously seen.

An important strategy used in the most efficient implementations of the Lin-Kernighan procedure is the so-called “don’t look bits” (DLB) approach described in Section 3.1. The strategy may be viewed as an instance of applying a critical event tabu list structure, where the tabu-active status of an attribute terminates as soon as a specified event occurs. In the case of the DLB approach, the attribute under consideration is a node, which is forbidden to be involved in a move, and hence is not examined to introduce a new edge, after it fails to be considered for an improving move.

More precisely, the usual DLB implementation can be succinctly formulated as a restricted application of tabu conditions, making use of TS terminology to describe its operation, as follows. An attribute (in this case a node) is assigned a tabu-active status as soon as a search for an improving move with that node as the base node v_1 fails. The tabu-active status of the node renders moves that involve this node tabu, and the status is removed in the DLB approach as soon as an improving move is found that drops an edge adjacent to the tabu-active node, thus identifying the “critical event” in this particular context. More general TS designs identify unattractive attributes by frequency memory over specified regions of the search, and then penalize such attributes during an intensification or diversification phase. The “region” for the Don’t

Look Bits approach is simply the domain of moves examined during an iteration when the addition of the edge fails to produce an improving move, and the penalty is pre-emptive, as in the more common forms of short-term TS memory.

We conjecture that the memory structure introduced by the “don’t look bits” strategy, in conjunction with efficient candidate list constructions, provides a key contribution to the performance of modern implementations of the Lin-Kernighan procedure. If so, there may be advantages to using more general types of critical event memory structures, governed by correspondingly more general uses of frequency memory, as a basis for alternative implementations. In this connection, it is interesting to note that the present implementations of the Stem-and-Cycle ejection chain method do not incorporate any type of memory structures (including the “don’t look bits” structure) to restrict the solution space and guide the search process. The attractive outcomes of this ejection chain approach compared to the LK implementations are therefore somewhat surprising, and invite further examination of the Stem-and-Cycle and other ejection chain structures, where consideration is given to including the types of supplemental implementation strategies that have supported the LK procedures.

3.3. Strategic Oscillation

Strategic oscillation represents a special diversification approach in tabu search that deserves its own discussion. An important challenge in the design of local search algorithms is to create strategies that effectively avoid the trap of getting stuck in local optima. It is not unusual in combinatorial optimization for high quality local optima to lie in deep (or “large”) valleys of the solution space, sometimes attended by numerous smaller variations in elevation along the general valley floor. In such cases, a significant number of iterations may be required to leave these basins of attraction in order to find new local optima of higher quality. One way to overcome this difficulty is to change the neighborhood when the telltale features of such a basin of attraction are observed. The identification of critical levels of change required to escape from “insidious valleys” provides the basis to implement a strategic oscillation that alternates between different (and somewhat complementary) neighborhood structures.

A key issue often encountered in strategic oscillation is to allow the method to cross boundaries of feasibility instead of strictly remaining within the feasible region. In general combinatorial problems, a common technique for doing this consists of relaxing some of the “hard”

constraints and introducing penalties associated with those that are violated as a result. Penalty values are appropriately adjusted at each iteration in order to bring the search back into the feasible region. Approaches of this type have been adopted to the heuristic context in the tabu search algorithm of Gendreau, Hertz, and Laporte [352] for the classic vehicle routing problem (VRP), which includes embedded TSPs. In this application, the vehicle capacity and the route length constraints are temporarily relaxed and handled by a penalty function as described.

In the TSP setting where constraints consist of enforcing a particular graph structure – in this case, a Hamiltonian circuit (or cycle) – oscillation strategies must rely upon the ability of the neighborhood structures to deal with infeasible graph structures. Typical examples of such neighborhoods are provided by the reference structures used in the Lin-Kernighan and Stem-and-Cycle Ejection Chain procedures. In these approaches, as previously noted, a feasible TSP tour chosen at one level of the move generation process is obtained by performing a sequence of (infeasible) moves that transform one reference structure into another, and recovering feasibility by performing a complementary trial move.

Another way to implement a strategic oscillation is to utilize constructive/destructive neighborhoods, which follow each construction phase by destroying the feasibility of the problem graph structure, and then build up a new solution by reconnecting the solution subgraph in a different way.

The destructive process can be done either one component at a time or based on selecting a subset of graph components as in the vocabulary building strategy of tabu search. In either case, the destructive process yields a partial subgraph made up of a subset of disconnected components. The aim of the constructive process is then to efficiently re-insert the missing components into the partial graph to create a new complete tour. The GENIUS algorithm of Gendreau, Hertz, and Laporte [351] uses a simple one-step (unit depth) oscillation, as noted earlier, but more advanced forms of oscillation are clearly possible.

4. Recent Unconventional Evolutionary Methods

It is useful to base the design of the constructive/destructive process on the observation of commonalities between good TSP tours, making use of associated tabu search memory components. Additional ways to create memory structures to explore intensification and diversification arise in connection with Scatter Search and Path Relinking methods which embody a population-based approach.

4.1. Scatter Search Overview

Scatter search [368] is an evolutionary method proposed as a primal counterpart to the dual approaches called surrogate constraint methods, which were introduced as mathematical relaxation techniques for discrete optimization problems. As opposed to eliminating constraints by plugging them into the objective function as in Lagrangean relaxations, surrogate relaxations have the goal of generating new constraints that may stand in for the original constraints. The approach is based on the principle of capturing relevant information contained in individual constraints and integrating it into new surrogate constraints as a way to generate composite decision rules and new trial solutions (Glover [367]).

Scatter search combines vectors of solutions in place of the surrogate constraint approach of combining vectors of constraints, and likewise is organized to capture information not contained separately in the original vectors. Also, in common with surrogate constraint methods, SS is organized to take advantage of auxiliary heuristic solution methods to evaluate the combinations produced and generate new vectors. As any evolutionary procedure, the method maintains a population of solutions that evolves in successive generations.

A number of algorithms based on the scatter search approach have recently been proposed for various combinatorial problems (Kelly, Ranganwamy and Xu [502], Fleurent et al. [313], Cung et al. [231], Laguna and Martí [530], Campos et al. [157], Glover, Løkketangen and Woodruff [381], Atan and Secomandi [48], Laguna, Lourenço and Martí [529], Xu, Chiu and Glover [829], Cavique, Rego and Themido [173]). For tutorial descriptions of Scatter Search with examples of different applications we refer the reader to Glover, Laguna, and Martí [380], and Rego and Leão [708].

Scatter search operates on a set of reference solutions to generate new solutions by weighted linear combinations of structured subsets of solutions. The reference set is required to be made up of high-quality and diverse solutions and the goal is to produce weighted centers of selected subregions that project these centers into regions of the solution space that are to be explored by auxiliary heuristic procedures. Depending on whether convex or nonconvex combinations are used, the projected regions can be respectively internal or external to the selected subregions.

For problems where vector components are required to be integer, a rounding process is used to yield integer values for such components. Rounding can be achieved either by a generalized rounding method or iteratively, using updating to account for conditional dependencies that can modify the rounding options.

Regardless of the type of combinations employed, the projected regions are not required to be feasible and hence the auxiliary heuristic procedures are usually designed to incorporate a double function of mapping an infeasible point to a feasible trial solution and then to transform this solution into one or more trial solutions. (The auxiliary heuristic commonly includes the function of restoring feasibility, but this is not a strict requirement since scatter search can be allowed to operate in the infeasible solution space.) From the implementation standpoint the scatter search method can be structured to consist of the following subroutine.

Diversification Generation Method - Produces diverse trial solutions from one or more arbitrary seed solutions used to initiate the method.

Improvement Method - Transforms a trial solution into one or more enhanced trial solutions. (If no improvement occurs for a given trial solution, the enhanced solution is considered to be the same as the one submitted for improvement.)

Reference Set Update Method - Creates and maintains a set of reference solutions that are the “best” according to the criteria under consideration. The goal is to ensure diversity while keeping high-quality solutions as measured by the objective function.

Subset Generation Method - Generates subsets of the reference set as a basis for creating combined solutions.

Solution Combination Method - Uses structured and weighted combinations to transform each subset of solutions produced by the subset generation method into one or more combined solutions.

A general template for a scatter search algorithm can be organized in two phases outlined as follows (Figure 8.13).

4.2. Scatter Search for the TSP

An important aspect in any evolutionary approach is the way solutions are encoded as members of the population. In genetic algorithms solutions were originally encoded as bit strings, though there have been some departures to this practice in recent years. The disposition to use bit strings in GA methods derives from the fact that the first GA crossover mechanism for combining solutions were based on simple exchanges of

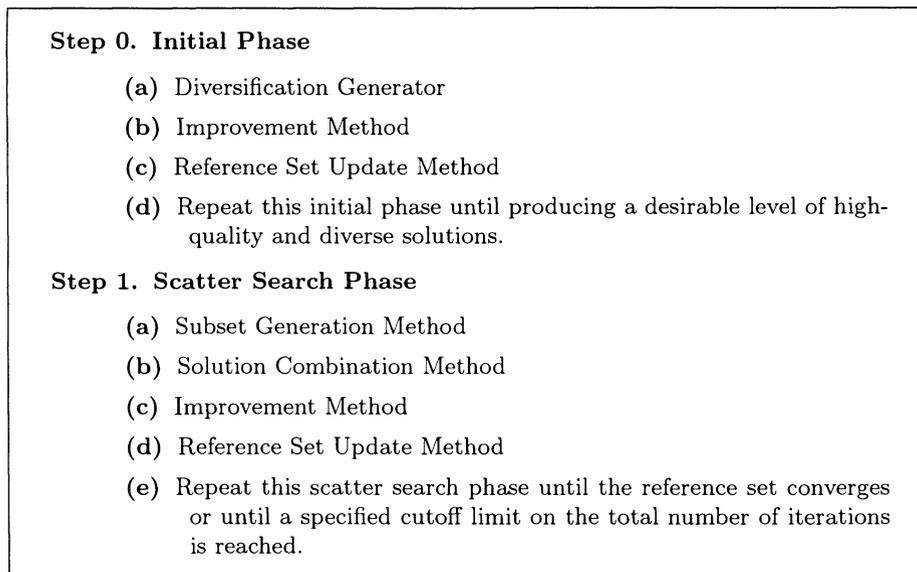


Figure 8.13. A Scatter Search Template

bits. In the classical GA bit string representations, continuous decision variables are usually encoded as substrings of the solution strings and their length depends on the precision required for these variables. Discrete decision variables are commonly encoded in these representations as a collection of zero-one variables, each corresponding to a single binary character in the solution string. For combinatorial problems defined on graphs, decision variables are typically associated with nodes or edges of the problem graph. In the TSP setting, where the number of edges is typically much larger than the number of nodes, solutions are usually encoded as a sequence of nodes representing a possible permutation for the problem. However, such a permutation-based representation used by GA approaches for the TSP entails several drawbacks subsequently noted.

By contrast, the original form of Scatter Search was not restricted to a specific type of encoding such as using bit strings, because the mechanism for combining solutions was not constrained to the limited crossover operations that governed the original GA formulations. In fact, SS readily incorporates different types of solution encodings in different parts of the method. In this section we discuss a Scatter Search approach for the TSP that utilizes such a “differential encoding” scheme. A node-based variable representation is used where information about

the relative value of the variables is not a primary issue, and an edge-based encoding is used otherwise.

To provide a general framework that discloses some critical features for applying scatter search to the TSP, a general design of the scatter search template for the TSP may be stated as follows.

Initial Phase

Diversification Generator. Scatter search starts by generating an initial set of diverse trial solutions, characteristically using a systematic procedure, which may include a stochastic component but which is highly “strategic” as opposed to relying chiefly on randomization.

Treating the TSP as a permutation problem, an illustrative approach for generating diverse combinatorial objects may be described as follows. A trial permutation P is used as a seed to generate subsequent permutations. Define the subsequence $P(h : s)$ to be the vector $P(h : s) = (s, s + h, s + 2h, \dots, s + rh)$, where r is the largest nonnegative integer such that $s + rh \leq n$. Relative to this, define the permutation $P(h)$ for $h < n$, to be $P(h) = (P(h : h), P(h : h - 1), \dots, P(h : 1))$. In the TSP context we consider permutations as n -vectors whose components are the vertices $v_i \in V$. Consider for illustration a TSP with $n = 14$ vertices, $h = 4$, and a seed permutation $P(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$ given by the sequence of vertices ordered by their indices. The recursive application of $P(4 : s)$ for $s = 4, \dots, 1$ results in the subsequences, $P = \{4, 8, 12\}$, $P = \{3, 7, 11\}$, $P = \{2, 6, 10, 14\}$, and $P = \{1, 5, 9, 13\}$, hence $P(4) = \{4, 8, 12, 3, 7, 11, 2, 6, 10, 14, 1, 5, 9, 13\}$. By varying h it is possible to generate up to n different permutations to initialize the reference set. The generated permutations can themselves represent tours by successively linking vertices in the order they appear in the permutation and attaching the initial and ending vertices to close up the tour.

Improvement Method. The improvement method used in the initial phase may or may not be the same method used in the scatter search phase. This decision usually depends on the context and on the search strategy one may want to implement. Here we consider the context of the Euclidian TSPs, where distances between vertices are ordinary Euclidian distances in the plane. For instance, since a diversification generator such as the one we are using characteristically generates edges that cross in the initial tours, and such crossings are non-optimal for Euclidian problems, a simple form of improvement method in the initial phase can be one of eliminating possible edge crossings rather than doing extensive local optimization. The objective is to avoid premature convergence and to keep a reference set with diverse and high quality solutions at each

iteration. In this context, the use of a classical k -opt procedure ($k = 2$ or 3) under a *first-improvement strategy*, which performs a move whenever it improves the current tour, may be appropriate for the initial phase of the SS procedure while a more powerful technique such as Lin-Kernighan or Stem-and-Cycle variable depth methods would be appropriate for the scatter search phase.

Reference Set Update Method. This method is used to create and maintain a set of reference solutions. As in any evolutionary method, a set of solutions (population of individuals) containing high evaluation combinations of attributes replaces less promising solutions at each iteration (generation) of the method. In genetic algorithms, for example, the updating process relies on randomized selection rules which select individuals according to their relative fitness value. In scatter search the updating process relies on the use of memory and is organized to maintain a good balance between intensification and diversification of the solution process. In advanced forms of scatter search reference solutions are selected based on the use of memory which operates by reference to different dimensions as defined in tabu search. Depending on the context and the search strategy, different types of memory are called for. As we have seen the term *adaptive memory programming* refers to the general realm of strategies for integrating and managing various types of memory to achieve both intensification and diversification. (See Glover and Laguna [379], and Rego and Alidaee [707], for a detailed explanation of various forms and uses of memory within search processes.) For the purpose of this discussion we consider a simple rule to update the set of reference solutions, where intensification is achieved by selecting high-quality solutions in terms of the objective function value and diversification is induced by including diverse solutions from the current candidate set CS . Thus the reference set RS can be defined by two distinct subsets B and D , representing respectively the subsets of high-quality and diverse solutions, hence $RS = B \cup D$.

Denote the cardinalities of B and D by $|B| = r_1$ and $|D| = r_2$, which do not need to be identical and can vary during the search. For instance, relatively larger sizes of B (D) can be appropriate during a phase that is more strongly characterized by an intensification (diversification) emphasis. Different schemes can be chosen to implement these variations. A dynamic variation of these sizes can be implemented by a perturbation scheme, for example, and a strategic oscillation approach with critical event memory can be used as an indicator of the order of magnitude of the relative variations.

It is important to distinguish the concepts of *difference* and *distance*. In the context of the TSP, the difference between two TSP tours is defined as the number of edges by which the two solutions differ. The distance between two solutions X and Y is defined as the minimum number of steps (or iterations) necessary for the local search algorithm to move from solution X to solution Y . Thus, the distance between two TSP tours depends on the type of neighborhood used in the local search algorithm and may not be directly related to the difference between the two TSP tours.

For a visual representation consider a solution space graph \hat{G} where nodes represent solutions and arcs define direct moves from one solution to another associated with a given neighborhood structure. The distance between two solutions X and Y is given by the shortest path (in terms of the number of arcs) from node X to node Y in the graph \hat{G} . It is easy to see that the distance between solutions is a more accurate measure of diversity than the difference between them. However, for the sake of simplicity it is common to use the difference between solutions as an indicator of their diversity, and for the same reason this measure can be used for the selection of diverse solutions to update D in the reference set.

Let CS denote the set of solutions generated and improved during the method's application. If some of these solutions produced by the diversification generator are not sufficiently distant from each other, it is possible that the improvement method may generate the same solution from several different members of CS . Therefore, it can be valuable to have a fast procedure to identify and eliminate solutions from CS that duplicate or "lie very close" to others before creating or updating the reference set. Such an approach can be facilitated by using an appropriate hash function.

A straightforward way to create a reference set RS consists of selecting the r_1 best solutions from CS to create B , and then to generate the set D of r_2 diverse solutions by successively selecting the solution that differs by the greatest amount from the current members of RS . As a diversity measure we define $d_{ij} = |(S_i \cup S_j) \setminus (S_i \cap S_j)|$ as the difference between solutions S_i and S_j , which identifies the number of edges by which the two solutions differ from each other. The d_{ij} values are computed for each pair of solutions $S_i \in RS$ and $S_j \in CS$.

Candidate solutions are included in RS according to the *Maxmin* criterion which maximizes the minimum distance of each candidate solution to all the solutions currently in the reference set. The method starts with $RS = B$ and at each step extends RS by selecting a solution $S_j \in CS$ and setting $RS = RS \cup \{S_j\}$ and $CS = CS \setminus \{S_j\}$.

More formally, the selection of a candidate solution is given by $S_j = \operatorname{argmax} \min_{i=1, \dots, |RS|} \{d_{ij} : j = 1, \dots, |CS|\}$. The process is repeated until RS is filled to the desired level.

Scatter Search Phase

Subset Generation Method. This method consists of generating subsets of reference solutions to create structured combinations, where subsets of solutions are organized to cover different promising regions of the solution space. In a spatial representation, the convex-hull of each subset delimits the solution space in subregions containing all possible convex combinations of solutions in the subset. In order to achieve a suitable intensification and diversification of the solution space, three types of subsets are organized to consist of:

- 1) subsets containing only solutions in B ,
- 2) subsets containing only solutions in D , and
- 3) subsets that mix solutions in B and D in different proportions.

Subsets defined by solutions of type 1 are conceived to intensify the search in regions of high-quality solutions while subsets of type 2 are created to diversify the search to unexplored regions. Finally, subsets of type 3 integrate both high-quality and diverse solutions with the aim of exploiting solutions across these two types of subregions.

Adaptive memory once again is useful to define combined rules for clustering elements in the various types of subsets. This has the advantage of incorporating additional information about the search space and problem context.

The use of sophisticated memory features is beyond the scope of this discussion. However, for illustrative purposes, we may consider a simple procedure that generates the following types of subsets:

- 1) All 2-element subsets.
- 2) 3-element subsets derived from two element subsets by augmenting each 2-element subset to include the best solution (as measured by the objective function value) not in this subset.
- 3) 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solution (as measured by the objective function value) not in this subset.

- 4) The subsets consisting of the best b elements (as measured by the objective function value), for $b = 5, \dots, |B|$.

Solution Combination Method. The Solution Combination method is designed to explore subregions within the convex-hull of the reference set. We consider solutions encoded as vectors of variables x_{ij} representing edges (v_i, v_j) . New solutions are generated by weighted linear combinations which are structured by the subsets defined in the preceding step. In order to restrict the number of solutions only one solution is generated in each subset by a convex linear combination defined as follows. Let E be a subset of RS , $|E| = r$, and let $H(E)$ denote the convex hull of E . We generate solutions $S \in H(E)$ represented as

$$S = \sum_{t=1}^r \lambda_t S_t$$

$$\sum_{t=1}^r \lambda_t = 1$$

$$\lambda_t \geq 0, \quad t = 1, \dots, r$$

where the multiplier λ_t represents the weight assigned to solution S_t . We compute these multipliers by

$$\lambda_t = \frac{\frac{1}{C(S_t)}}{\sum_{t=1}^r \frac{1}{C(S_t)}}$$

so that the better (lower cost) solutions receive higher weight than less attractive (higher cost) solutions. Then, we calculate the score of each variable x_{ij} relative to the solutions in E by computing

$$score(x_{ij}) = \sum_{t=1}^r (\lambda_t x_{ij}^t)$$

where x_{ij}^t denotes that x_{ij} is an edge in the solution S_t . Finally as variables are required to be binary, the final x_{ij} value is obtained by rounding its score to give $x_{ij} = \lfloor score(x_{ij}) + .5 \rfloor$. The computation of the value for each variable in E results in creating a linear combination of the solutions in E and a new solution can be produced using edges associated with variables $x_{ij} = 1$. Nevertheless, the set of these edges. Nevertheless,

the set of these edges does not necessarily (and usually doesn't) represent a feasible graph structure for a TSP solution, since it typically produces a subgraph containing vertices whose degrees differ from two. Such subgraphs can be viewed as fragments of solutions (or partial tours). When the subgraph resulting from a linear combination contains vertices of degree greater than two, a very straightforward technique consists of successively dropping edges with the smallest scores in the *star* (incident edge set) of these vertices until their degree becomes equal to two. By doing so, the subgraph obtained will be either feasible or fall into the case where some of the vertices have degree 1. At this juncture there are several possibilities to create a feasible solution subgraph and an appropriate tradeoff has to be chosen. For example, a simple possibility is to group vertices of degree 1 and use a heuristic that simply links them two by two according to some distance measure or savings criterion. Another possibility is to solve the linear assignment problem to match each pair of nodes according to their relative distances.

4.3. Path Relinking

Scatter Search (SS) provides a natural evolutionary framework for adaptive memory programming, as we have seen, by its incorporation of strategic principles that are shared with certain components of Tabu Search. Another strategy for integrating SS and TS principles consists of replacing vector spaces with neighborhood spaces as a basis for combining solutions, which gives rise to a TS strategy called Path-Relinking (PR).

More particularly, while SS considers linear combinations of solution vectors, PR combines solutions by generating paths between them using local search neighborhoods, and selecting new solutions encountered along these paths.

This generalization of SS can be described by the same general template outlined in Figure 8.13. Figure 8.14 provides a visual interpretation of the PR process. The lines leaving S in the figure shows an alternative paths traced by the path-relinking strategy having the solutions denoted by S_1 , S_2 and S_3 operate as *guiding solutions*, which collectively determine the path trajectory taken from the *initial solution* S during the local search process. In the simplest case, a single guiding solution can be used.

The process of generating paths between solutions is accomplished by selecting moves that introduce attributes contained in the solutions

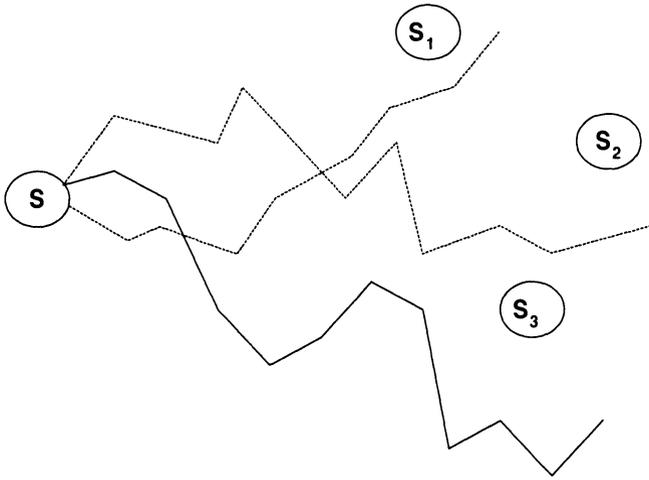


Figure 8.14. Path Relinking

that operate as guiding solutions. In the move generation process, these attributes are weighted to determine which moves are given higher priority. Again, by analogy with the SS design, each intermediate point lying in a path between solution S and a given guiding solution S' can be viewed as the result of a combination of these solutions.

By extension, a number of strategies are possible for a fuller exploration of the solution space in a path-relinking framework. Alternative paths from S under the influence of the guiding solutions can be generated by using memory structures of the type embodied in TS. Also, in a given collection of elite solutions, the roles of initiating solution and guiding solutions can be alternated. That is, a set of current solutions may be generated simultaneously, extending different paths, and allowing an initiating solution to be replaced (as a guiding solution for others) whenever its associated current solution satisfies a sufficiently strong aspiration criterion. Because their roles are interchangeable, the initiating and guiding solutions are collectively called *reference solutions*.

The possibility of exploring different trajectories in the neighborhood space suggests the use of alternative neighborhood structures with the objective of reaching solutions that might otherwise be bypassed. This strategy, denoted in TS terminology by *tunneling*, offers additional possibilities to explore boundaries between regions of feasible and infeasible solutions as a form of *strategic oscillation*.

Path-relinking provides a useful means for integrating intensification and diversification, by reference to groups (or clusters) of elite solutions that are organized according to some measure of relative difference or

distance that gives an indicator of their diversity. Solutions declared “close” to each other according to a given criterion typically are assigned to the same cluster and the objective is to maintain a set of clusters along the search that differ from each other by a significant degree. Here the concept of proximity is broad rather than restrictive in the sense that solutions may be considered close to one another if they share some particular characteristics relevant in the context of the problem under consideration. In the TSP context, for example, proximate solutions may be the ones containing edges that are common to many good solutions. In a path relinking strategy, choosing solutions S and S' from the same cluster stimulates intensification (by preserving common characteristics of these solutions), while choosing them from two different clusters stimulates diversification (by including attributes of the guiding solutions not contained in the initial ones). This approach can go beyond the target solutions by extrapolation, creating an effect analogous to the creation of non-convex linear combinations allowed in the Euclidian space. But if an attractive departure from a guided trajectory is found along the way (using aspiration criteria), then this alternative route can also be explored, providing a dynamic integration of intensification and diversification.

Given that Ejection Chain methods, including the important special case represented by the Lin-Kernighan approach, have provided some of the most efficient algorithms for the TSP, a natural possibility is to join such methods with path relinking to provide a broader range of strategies. Such an approach, which is currently unexplored, can also take advantage of other heuristic processes previously described. For example, a combination of ejection chains and path relinking, can draw upon a sequential fan method to generate paths within the path-relinking procedure. The move components of a sequential fan candidate list can be organized in this setting to include the attributes of the designated guiding solutions. By applying ejection chain constructions to provide a look-ahead process in the sequential fan method, high evaluation trial solutions can be chosen to update the reference set (RS) of guiding solutions for a multi-parent path-relinking strategy. In such a strategy, it is important to consider appropriate measures of distance between the initial solution and the guiding solutions so that solutions in RS differ by approximately the same degree from the initial solution. By extension, if a sufficient number of ejection chain levels is generated to reach solutions that lie at distances beyond those of the current guiding solutions, then high quality solutions found in this extended neighborhood space can be used as guiding points for an extrapolated path-relinking process. Approaches of this form can be relevant not only for TSPs but also for

generalizations that include additional constraints and “embedded TSP” structures.

Finally, we observe that additional metaheuristic approaches exist that offer the potential to create useful hybrid methods for the TSP and its variants. It is beyond the scope of this chapter to provide a detailed description of such methods, but we refer the reader to Glover and Kochenberger [378] for an extensive coverage of these alternative procedures.

5. Conclusions and Research Opportunities

The current state-of-the-art discloses that the key to designing efficient algorithms for large scale traveling salesman problems is to combine powerful neighborhood structures with specialized candidate list strategies, while giving careful attention to appropriate data structures for implementation. As reported in Chapter 9, the Lin-Kernighan (LK) procedure and the Stem-and-Cycle procedure, which represent alternative instances of Ejection Chain (EC) methods, currently provide the most effective algorithms for solving large TSPs. The merit of the EC approaches derives from the use of reference structures to generate compound moves from simpler components, where the evaluation of a move at each level of construction is subdivided into independent operations to gain efficiency. The definition of the reference structure is highly important in these methods, and more advanced reference structures (such as the doubly-rooted loop constructions of [372], for example) invite examination in the future. Such structures provide an opportunity to generate moves with special properties not incorporated in k -opt moves generated by present TSP procedures.

Another potential strategic enhancement comes from the fact that the LK and the Stem-and-Cycle procedures characteristically create paths in neighborhood space by elaborating only a single thread of alternatives throughout successive levels of construction. A more aggressive way to employ such processes is to embed them in special neighborhood search trees, as described by Sequential Fan (SF) and Filter and Fan (F&F) methods. This affords the possibility to go beyond “greedy one-step choices” in the creation of neighborhood paths, while utilizing mechanisms that are highly susceptible to parallel implementation. SF and F&F approaches can also be used to merge neighborhoods of varying characteristics within different stages and threads of the search. Coupling such elements with more carefully designed memory-based strategies, such as those derived from adaptive memory programming considerations, provide additional avenues for future investigation.

Acknowledgements

We are indebted to David Johnson whose perceptive and detailed observations have improved this chapter in a number of places.

This research was supported in part by the Office of Naval Research (ONR) grant N000140110917.

Chapter 9

EXPERIMENTAL ANALYSIS OF HEURISTICS FOR THE STSP

David S. Johnson

AT&T Labs – Research, Room C239, Florham Park, NJ 07932, USA
dsj@research.att.com

Lyle A. McGeoch

Dept. of Mathematics and Computer Science, Amherst College, Amherst, MA 01002
lam@cs.amherst.edu

1. Introduction

In this and the following chapter, we consider what approaches one should take when one is confronted with a real-world application of the TSP. What algorithms should be used under which circumstances? We are in particular interested in the case where instances are too large for optimization to be feasible. Here theoretical results can be a useful initial guide, but the most valuable information will likely come from testing implementations of the heuristics on test beds of relevant instances. This chapter considers the symmetric TSP; the next considers the more general and less well-studied asymmetric case.

For the symmetric case, our main conclusion is that, for the types of instances that tend to arise in practice, heuristics can provide surprisingly good results in reasonable amounts of time. Moreover the large collection of heuristics that have been developed for the STSP offers a broad range of tradeoffs between running time and quality of solution. The heuristics range from those that take little more time than that needed to read an instance and still get within 50% of optimum to those that get within a few percent of optimum for 100,000-city instances in seconds to those that get within fractions of a percent of optimum for instances of this size in a few hours.

The relevant level of performance will of course vary, depending on the application. This chapter provides a tentative characterization of the most promising approaches at many levels of the tradeoff hierarchy. In this way we hope to put previous theoretical and experimental work into a practical perspective.

In order to provide an up-to-date picture of the state of the art, the authors of this chapter, together with Fred Glover and Cesar Rego, organized a DIMACS Implementation Challenge¹ on the STSP. The Challenge began in June 2000 and continued through November, with additional data collected through June 2001. Researchers from all over the world, including all the current top research groups, ran their codes on instances from a collection of test suites, reporting running times and the lengths of the constructed tours. They also reported running times for a special benchmark code distributed by the organizers. These times allowed us to estimate the speeds of their machines (as a function of instance size) and thus to normalize running times to what they might (approximately) have been had the codes all been run on the same fixed machine. We thus can provide detailed comparisons between a wide variety of heuristics and implementations with specific attention to robustness, scalability, and solution quality/running time tradeoffs. In this way we hope to improve on earlier studies, such as those of Golden and Stewart [388], Bentley [103], Reinelt [710, 711], and Johnson-McGeoch [463], which covered fewer heuristics and instances and did not provide as convenient mechanisms for future comparability.

The remainder of this chapter is organized as follows. In Section 2 we provide more details about the Challenge, the testbeds of instances it covered, its participants, our scheme for normalizing running times, and our methods for evaluating tour quality. In Section 3, we describe the various heuristics studied, divided into groups on the basis of approach and speed, and summarize the experimental results obtained for them. Section 4 then presents some overall conclusions and suggestions for further research.

We should note before proceeding that certain heuristics described elsewhere in this book are for various reasons not covered in this chapter. Perhaps our foremost omission is the approximation schemes for geometric STSP's of Arora, Mitchell, et al. [35, 601, 696], as described in Chapter 5. These heuristics, despite their impressive theoretical guar-

¹DIMACS is the Center for Discrete Math and Theoretical Computer Science, a collaboration of Rutgers and Princeton Universities with Bell Labs, AT&T Labs, NEC Labs, and Telcordia Technologies. This was the 8th in the DIMACS Implementation Challenge series. For more information, see <http://dimacs.rutgers.edu/Challenges/>.

antees, have significant drawbacks compared to the competition we shall be describing. Because of the perturbation of the instances that they initially perform, the versions of the heuristics guaranteeing $1 + \epsilon$ worst-case ratios are likely to be off by a significant fraction of ϵ even in the average case. Thus, to be competitive with heuristics that typically get within 1 or 2 percent of optimum in practice, one probably must choose $\epsilon < 0.05$. This is likely to make the running times prohibitive, given the large constant factor overheads involved and the fact that the running times are exponential in $1/\epsilon$. It would be interesting to verify that this is indeed the case, but as of this date we know of no attempt at a serious implementation of any of the schemes.

A second hole in our coverage concerns local-search heuristics based on polynomial-time searchable exponential-size neighborhoods, one of the subjects of Chapter 6. We have results for only one such heuristic. Empirical study of such heuristics is still in its infancy, and so far very little has emerged that is competitive with the best traditional STSP heuristics.

The final hole in our coverage is rather large – much of the burgeoning field of metaheuristics is not represented in our results. Although we do cover one set of tabu search implementations, we cover no heuristics based on simulated annealing, neural nets, classical genetic algorithms, GRASP, etc. The Challenge was advertised to the metaheuristic community and announcements were sent directly to researchers who had previously published papers about heuristics of these sorts for the TSP. For various reasons, however, little was received. Fortunately, we may not be missing much of practical value in the context of the STSP. As reported in the extensive survey [463], as of 1997 all metaheuristic-based codes for the STSP were dominated by 3-Opt, Lin-Kernighan, or Iterated Lin-Kernighan. Metaheuristics, if they are to have a role in this area, are more likely to be useful for *variants* on the TSP. For example, they might well adapt more readily to handling side constraints than would more classical approaches.

2. DIMACS STSP Implementation Challenge

For a full description of the DIMACS Implementation Challenge, see the website at <http://www.research.att.com/~dsj/chtsp/>. In addition to providing input for this chapter, the Challenge is intended to provide a continually updated picture of the state of the art in the area of TSP heuristics (their effectiveness, robustness, scalability, etc.). This should help future algorithm designers to assess how their approaches compare with already existing TSP heuristics.

To this end, the website currently makes a variety of material available for viewing or downloading, including instances and instance generators, benchmark codes, raw data from the participants, and statistics and comparisons derived therefrom. Our intent is to maintain the website indefinitely, updating it as new results are reported and adding new instances/instance classes as interesting ones become available.

This chapter presents a summary and interpretation of the available results as of June 2001, representing code from 15 research groups. Many of the groups reported on implementations of more than one heuristic or variant, thus providing us with fairly comprehensive coverage of the classical heuristics for the STSP along with several promising new approaches. For additional details, the reader is referred to the website and to a forthcoming DIMACS Technical Report that will present the data in a more linear fashion. In the remainder of this section, we describe the Challenge testbeds in more detail, as well as our scheme for normalizing running times.

2.1. Testbeds

In designing the Challenge testbeds, we have chosen to ignore instances with fewer than 1,000 cities. This was done for several reasons. First, as we shall see, currently available optimization codes, in particular the publicly available *Concorde* package² of Applegate, Bixby, Chvátal, and Cook [29], seem to be able to solve typical STSP instances with fewer than 1,000 cities in quite feasible running times. Indeed, *Concorde* was able to solve all the 1,000-city instances in our random testbeds using its default settings. Normalized running times were typically in minutes, and the longest any such instance took was just a little over two hours. Second, if one is only willing to spend seconds rather than minutes, the best of the current heuristics are hard to beat. For instance, the $N/10$ -iteration version of the publicly available LKH code of Keld Helsgaun [446] can get within 0.2% of optimum in no more than 20 seconds (normalized) for each of our 1,000-city random instances and for the six TSPLIB³ instances with between 1,000 and 1,200 cities. Thus it is not clear that heuristics are needed at all for instances with fewer than 1,000 cities, and even if so, high quality solutions can be obtained in practical running times using publicly available codes. The real research question is how heuristic performance scales as instance sizes grow, es-

²Currently available from <http://www.math.princeton.edu/tsp/concorde.html>.

³TSPLIB is a database of instances for the TSP and related problems created and maintained at <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/> by Gerd Reinelt and described in [709].

pecially since many modern applications generate instances with 10,000 cities or more.

Our second decision was to concentrate primarily on geometric instances in two dimensions. Most experimental research on the STSP to date has concentrated on such instances. This is largely because the major applications for the STSP, both industrial and academic, have been of this sort. Consequently, many codes have been written to exploit such structure and only work for such instances. (Limited experimentation with higher dimensional instances suggests that the lessons learned in two dimensions carry over at least to three or four [461, 465].)

A third decision, based on common practice in the literature and an assumption made by many codes, was to restrict attention to instances in which inter-city distances are all integral.

The Challenge test suite contains three classes of geometric instances:

- Random Uniform Euclidean Instances (“Uniform”). Here the cities are points whose two coordinates are each integers chosen randomly from the interval $[0, 10^6]$, with instance sizes increasing roughly by factors of $\sqrt{10}$ from $N = 1,000$ to $N = 10,000,000$. Distances are the Euclidean distance rounded to the nearest integer. There are ten instances with 1,000 cities, five with 3,162, three of size 10,000, two each of sizes 31,623 and 100,000, and one each of sizes 316,228, 1,000,000, 3,162,278, and 10,000,000. Instances of this type have been widely studied and yield an interesting view on asymptotic performance.
- Random Clustered Euclidean Instances (“Clustered”). Here we choose $N/100$ cluster centers with coordinates chosen uniformly in $[0, 10^6]$, and then for each of the N cities we randomly choose a center and two normally distributed variables, each of which is then multiplied by $10^6/\sqrt{N}$, rounded, and added to the corresponding coordinate of the chosen center. Distances are again the Euclidean distance rounded to the nearest integer. For this class there are ten instances with 1,000 cities, five with 3,162, three of size 10,000, two each of sizes 31,623 and 100,000, and one of size 316,228. These were designed to be challenging for local search heuristics.
- All 33 geometric instances in TSPLIB with 1,000 or more cities as of June 2001. These instances range in size from 1,000 cities to 85,900. All are 2-dimensional with rounded Euclidean distances (either rounded up or to the nearest integer). Most come either from geography (coordinates of actual cities, with the earth viewed as planar) or from industrial applications involving circuit boards, printed circuits, or programmable gate arrays.

As to applications with non-geometric instances, these tend to be asymmetric as well as non-geometric and hence will be covered in the next chapter. For the STSP Challenge, our main source of non-geometric instances consisted of random symmetric distance matrices, the only non-geometric class that has previously been widely studied in the context of the STSP. Although such instances have no direct relevance to practice, they do offer a substantial challenge to many heuristics and thus are useful in studying the robustness of various approaches. For our Random Matrix testbed, distances were independently chosen integers distributed uniformly in the interval $[0, 10^6]$. We include four instances of size 1,000, two of size 3,162, and one of size 10,000. (Since an instance of this type consists of roughly $N^2/2$ integers, storage can become a problem for larger N .) In addition, our testbed contains the one instance from TSPLIB that is given by its distance matrix and has 1,000 or more cities (si1032).

Although participants were encouraged to run their codes on as many of the testbed instances as possible, this was not always possible. There were four main reasons why participants could not handle the entire test suite:

1. The participant's code was too slow to handle the largest instances on the participant's machine.
2. The code was fast enough, but required too much memory to handle the largest instances on the participant's machine.
3. The participant's code was designed to handle geometric instances and so could not handle instances given by distance matrices.
4. The participant's code was not designed to handle instances with fractional coordinates. Despite the fact that the TSPLIB instances all have integral inter-city distances, 13 of the 33 geometric TSPLIB instances in our test suite have fractional coordinates.

Not all of these need be defects of the underlying *heuristic*. In particular, (4) can typically be circumvented by additional coding, as several of our participants have shown, and (1) and (2) can often be ameliorated by cleverer code-tuning and memory management (or more powerful machines). Reason (3) may be less forgiving, however: Some heuristics are geometric by definition (e.g., Convex Hull Cheapest Insertion), and others will experience substantial slowdowns if they are unable to exploit geometric structure. In any case, we can only report on the results for the implementations we have, although where relevant we will try to identify those heuristics for which faster or more robust implementations may well be possible.

2.2. Running Time Normalization

Running time comparisons are notoriously difficult to make with any precision, even when all codes are compiled using the same compiler and compiler options and run on the same machine. By allowing participants to compile their own codes and run them on their own machines, we have made the problem substantially more difficult. However, since we did not wish to restrict participation to those who were willing to share their source codes, and we wanted to establish a record of the state of the art that might still be meaningful after the machines we currently have are obsolete and forgotten, there seemed to be no other choice.

In order to provide some basis for comparison, we thus have distributed a benchmark STSP code, an implementation of the Greedy (or Multi-Fragment) heuristic that uses K - d trees to speed up its operation on geometric instances. Participants were asked to run this code on their machines for a set of instances covering the whole range of sizes in the Challenge test suite and to report the resulting running times. Note that one cannot accurately quantify the difference in speeds between two machines by a single number. Because of various memory hierarchy effects, the relative speeds of two machines may vary significantly as a function of the size of the input instances. Figure 9.1, which graphs the running time of the benchmark code as a function of instance size for a variety of machines, shows how widely relative machine speeds can vary as a function of N . (In the chart, running times are divided by $N \log N$, the approximate number of basic operations performed by the heuristic.)

Using these reports, we can normalize running times to approximately what they would have been on a specific benchmark machine: a Compaq ES40 with 500-Mhz Alpha processors and 2 Gigabytes of main memory. The basic plan is to compute a normalization factor as a function of N . For N equal to one of the instance sizes in our Uniform test suite, we simply use the ratio between the benchmark code's time on the source machine and on the ES40 for the test instance of that size (assuming the benchmark code could be run on the source machine for instances that large). For other values of N , we interpolate to find the appropriate normalization factor.

There are multiple sources of potential inaccuracy in this process. Linear interpolation is an inexact approach to getting intermediate normalization factors. A particular code may require more (or less) memory for a given value of N than does the benchmark Greedy code. It may make more (or less) efficient use of instruction and data caches than the benchmark code. Also, our normalization process de-emphasizes the time to read an instance. Reading times do not necessarily differ by the

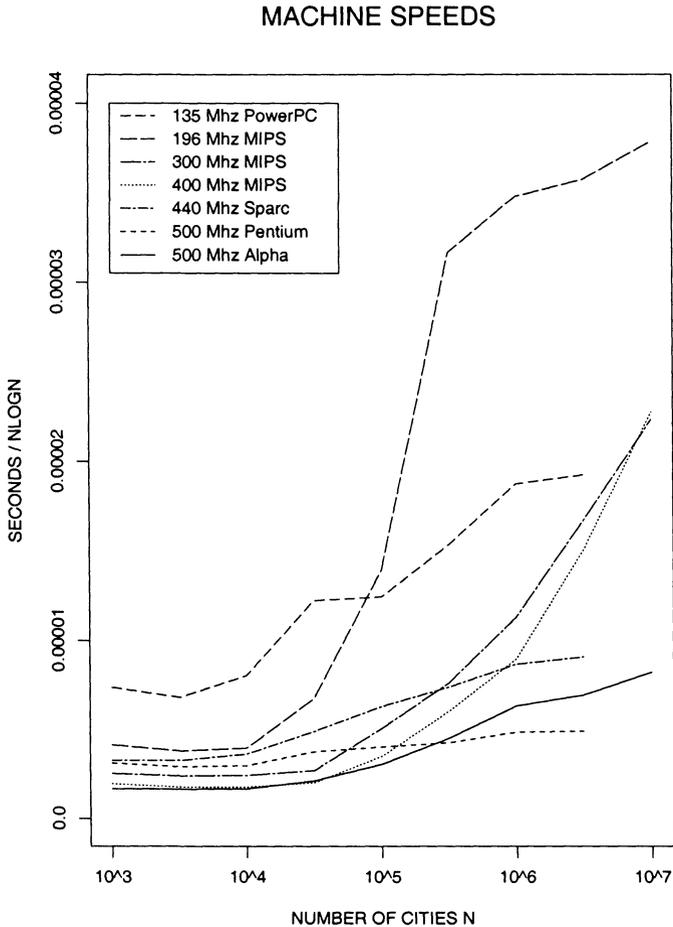


Figure 9.1. Running times for benchmark greedy code as a function of instance size for a variety of microprocessor-based machines. The microprocessors are listed in the order of their average times for 1000-city instances.

same factors as do CPU times, and they can be a significant component of the running time for the faster codes in our study, especially on smaller instances and on instances given by full distance matrices.

The de-emphasis arises because of the way we deal with the fact that systems typically only report running times in increments of 0.01 seconds. The benchmark Greedy code is so fast that its running time is typically 0.00, 0.01, or 0.02 for 1,000-city instances. This makes it difficult to derive precise normalization factors based on a single run. Thus, when we perform the benchmark runs on the smaller instances, we re-

port the total time over a series of runs on the same instance. The number of runs is chosen so that the product of the number of runs and N is roughly 1,000,000, with just one run performed for each instance with 1,000,000 or more cities. Although the basic data structures are rebuilt in each run, the instance itself is read only once. A single read makes sense since the heuristics we are testing only read the instance once. However, by reducing the proportion of the total time devoted to reading, this approach may misrepresent the impact of reading time on heuristics for which it is a major component of the total time.

To get a feel for the typical accuracy of our normalization procedure, see Figure 9.2 which charts, for the benchmark **Greedy** code and a Johnson-McGeoch implementation of the Lin-Kernighan heuristic the ratio between the actual time on the target ES40 machine and the normalized time based on compiling the code and running it on a 196-Mhz MIPS R10000 processor in an SGI Challenge machine. Note that for each heuristic, the error is somewhat systematic as a function of N , but the error is not consistent between heuristics. For **Greedy** the tendency is to go from underestimate to overestimate as N increases, possibly reflecting the reading time underestimate mentioned above. For Lin-Kernighan, on the other hand, read time is not a major component of running time on geometric instances, and for these the tendency is to go from overestimate to underestimate, possibly because this code needs substantially more memory than **Greedy** and because the MIPS machine has larger 2nd level caches than does the ES40. It is worth noting, however, that for both codes the estimate is still typically within a factor of two of the correct time.

Unfortunately, even if we can estimate running times for specific codes to within a factor of two, this may not imply anything so precise when talking about *heuristics*. Differing amounts of low-level machine-specific code tuning can yield running time differences of a factor of two or more, even for implementations that supposedly use the same data structures and heuristic speedup tricks. And the latter can cause even greater changes in speed, even though they are not always specified in a high-level description of a heuristic. Thus, unless one sees order-of-magnitude differences in running times, or clear distinctions in running time growth rates, it is difficult to draw definitive conclusions about the relative efficiency of heuristics implemented by different people on different machines. Fortunately, there *are* orders-of-magnitude differences in running time within the realm of TSP heuristics, so some conclusions about relative efficiency will be possible.

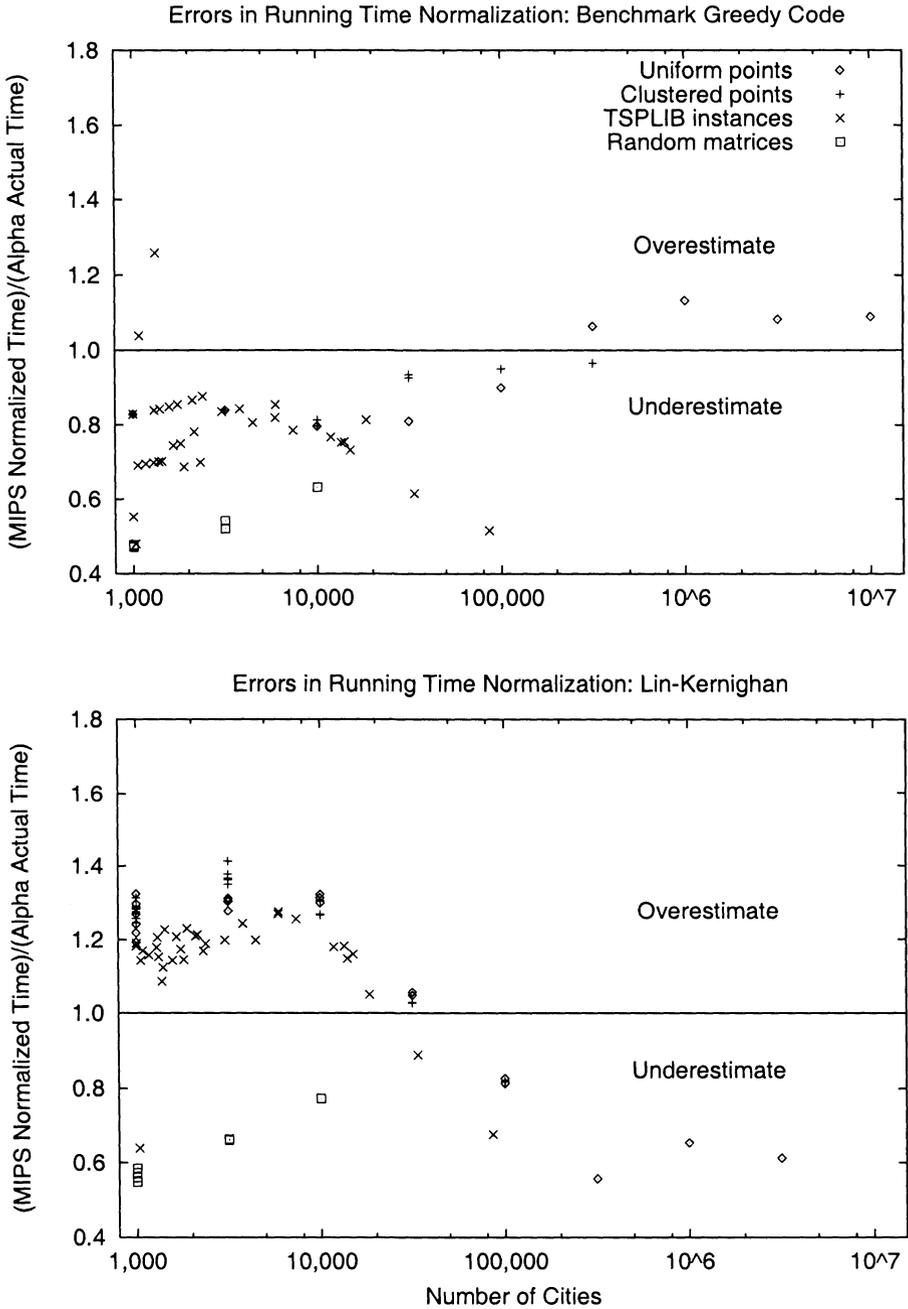


Figure 9.2. Ratios of predicted running time to actual running time on a Compaq 500-Mhz Alpha processor for the Benchmark Greedy code and for the Johnson-McGeoch implementation of Lin-Kernighan.

2.3. Evaluating Tour Quality

The gold standard for tour quality is of course the distance from the optimal solution, typically measured as the percentage by which the tour's length exceeds the length of an optimum tour. In order to use this standard, one unfortunately must *know* the optimal solution value.

Modern optimization technology is surprisingly effective: provably optimal solutions have been found for all but one of the instances in TSPLIB with 15,112 cities or fewer, and the *Concorde* code of Applegate, Bixby, Chvátal, and Cook [29] is able to solve all the random instances in the Challenge test suite with 3,162 or fewer cities (and the one 10,000-city Random Matrix instance). However, a prime reason for using heuristics is to get reasonable results for instances that are too difficult for current optimization algorithms to work. For this reason our test suite contains many instances for which optimal tour lengths are not yet known.

In order to provide a point of reference that is similar across all instances, our default comparison is thus to the Held-Karp lower bound on the optimal solution [444, 445]. This is the linear programming relaxation of the standard integer programming formulation for the STSP, as described in Chapter 2. Johnson et al. [465] argue that this bound is a good surrogate for the optimal solution value. For Random Uniform Euclidean instances in particular, they conjecture that the expected gap between the optimal tour length and the Held-Karp bound is asymptotically less than 0.65% and they provide extensive experimental evidence supporting this conjecture.

Table 9.1 shows the percent by which the optimal tour length exceeds the Held-Karp bound for all the instances in our test suite where the optimal is known. Note that the typical excess is less than 1% and the maximum excess observed is 1.74%. Moreover, although the optimal tour length is not yet known for four of the largest TSPLIB instances, for each one a tour is known that is within .54% of the Held-Karp bound.

The table also includes the normalized running times for computing the optimal tour lengths and for computing the Held-Karp bounds, which is typically much easier. When a running time is reported for an optimal tour length computation, it represents the time taken by *Concorde* using its default settings. For the random instances *Concorde* was run on our 196-Mhz MIPS processors. Times for the TSPLIB instances are those reported by Applegate et al. on their TSP webpage (<http://www.math.princeton.edu/tsp/>) for the same 500-Mhz Alpha processor used in our benchmark machine. For those instances with known optima but no quoted running time, additional expertise was needed (and running time – more than 22 CPU years for d15112).

Random Uniform Euclidean				TSPLIB			
Name	%Gap	Opttime	HKtime	Name	%Gap	Opttime	HKtime
E1k.0	0.77	1406	2.13	dsj1000	0.61	410	3.68
E1k.1	0.64	3855	2.15	pr1002	0.89	34	2.40
E1k.2	0.72	1211	2.02	si1032	0.08	25	11.32
E1k.3	0.62	956	1.92	u1060	0.65	571	3.62
E1k.4	0.69	330	1.69	vm1084	1.33	605	2.40
E1k.5	0.59	233	2.42	pcb1173	0.96	468	1.70
E1k.6	0.79	2940	1.67	d1291	1.18	27394	4.54
E1k.7	0.94	8003	1.95	rl1304	1.55	189	4.08
E1k.8	1.01	4347	1.65	rl1323	1.65	3742	4.49
E1k.9	0.61	189	2.14	nrv1379	0.43	578	2.40
E3k.0	0.71	533368	9.57	fl1400	1.74	1549	9.83
E3k.1	0.67	425631	10.54	u1432	0.29	224	2.42
E3k.2	0.74	342370	9.41	fl1577	1.66	6705	38.19
E3k.3	0.67	147135	10.30	d1655	0.94	263	6.51
E3k.4	0.73	–	8.07	vm1748	1.35	2224	4.43
Random Clustered Euclidean				u1817	0.90	449231	5.01
C1k.0	0.54	337	9.83	rl1889	1.55	10023	11.45
C1k.1	0.41	534	10.84	d2103	1.44	–	8.19
C1k.2	0.42	320	8.79	u2152	0.62	45205	8.10
C1k.3	0.53	214	7.63	u2319	0.02	7068	3.16
C1k.4	0.58	768	9.36	pr2392	1.22	117	5.75
C1k.5	0.58	139	9.29	pcb3038	0.81	80829	7.26
C1k.6	0.73	1247	7.07	fl3795	1.04	69886	123.66
C1k.7	0.58	449	13.24	fnl4461	0.55	–	12.47
C1k.8	0.34	140	10.40	rl5915	1.56	–	42.00
C1k.9	0.66	703	9.61	rl5934	1.38	–	56.15
C3k.0	0.62	16009	53.03	pla7397	0.58	–	55.42
C3k.1	0.61	17754	126.49	rl11849	1.02	–	102.41
C3k.2	0.70	18237	80.39	usa13509	0.66	–	120.20
C3k.3	0.57	6349	71.57	d15112	0.52	–	90.13
C3k.4	0.57	4845	44.02				
Random Matrices							
M1k.0	0.01	60	5.47	M3k.0	0.00	612	40.35
M1k.1	0.03	137	5.51	M3k.1	0.01	546	39.52
M1k.2	0.01	151	5.63	M10k.0	0.00	1377	367.84
M1k.3	0.01	169	5.26				

Table 9.1. For instances in the Challenge test suite that have known optimal solutions, the percent by which the optimal tour length exceeds the Held-Karp bound and the normalized running times in seconds for computing each using *Concorde* with its default settings. (“–” indicates that the default settings did not suffice.) For random instances, suffixes 1k, 3k, and 10k stand for 1,000, 3,162, and 10,000 cities respectively. The number of cities in a TSPLIB instance is given by its numerical suffix.

The Held-Karp times reported are also for *Concorde*, which contains a flag for computing the Held-Karp bound. Although the linear program that defines the bound involves an exponential number of “subtour” constraints, there are simple routines for finding violated constraints of this type, and typically not many of these need be found in order to solve the LP exactly. This is much more effective (and accurate) than the Lagrangean relaxation approach originally suggested by Held and Karp. *Concorde* was able to compute the bound using its default settings on our local SGI machine for all the instances in the Challenge test suite with less than a million cities, with the maximum normalized running time being roughly 4 hours for the 316,228-city random clustered Euclidean instance. Using more powerful machines at Rice University, Bill Cook used the code to compute the bound for our million-city instance. For the two instances in our test suite with more than a million cities, we relied on the empirical formula derived in [465], which was off by less than .02% for the million-city instance.

3. Heuristics and Results

As noted in the Introduction, currently available heuristics for the STSP provide a wide variety of tradeoffs between solution quality and running time. For example, Figure 9.3 illustrates the average performance of a collection of heuristics on the three 10,000-city instances in our testbed of Uniform instances. The underlying data is presented in Table 9.2. Details on the heuristics/implementations represented in the chart and table will be presented later in this section.

The normalized running times range from 0.01 seconds to over 5 hours, while the percentage excess over the Held-Karp bound ranges from about 35% down to 0.69% (which is probably within 0.1% of optimum). There is not, however, a complete correlation between increased running time and improved quality. Some heuristics appear to be *dominated*, in that another heuristic can provide equivalently good tours in less time or can provide better tours in the same time or less. For example, Bentley’s implementation of Nearest Insertion (NI) from [103] is dominated by his implementation of Nearest Neighbor (NN), and the Tabu Search implementation *Tabu-SC-SC* is dominated by three sophisticated iterated variants on Lin-Kernighan (*MLLK-.1N*, *CLK-ABCC-N*, and *Helsgaun-.1N*).

In this chapter, we shall separately consider groups of heuristics clustered in different regions of this trade-off spectrum, attempting to identify the most robust undominated heuristics in each class. Although we shall concentrate primarily on undominated heuristics, we will not do so exclusively. Dominated heuristics for which theoretical results have

10,000-City Random Uniform Euclidean Instances

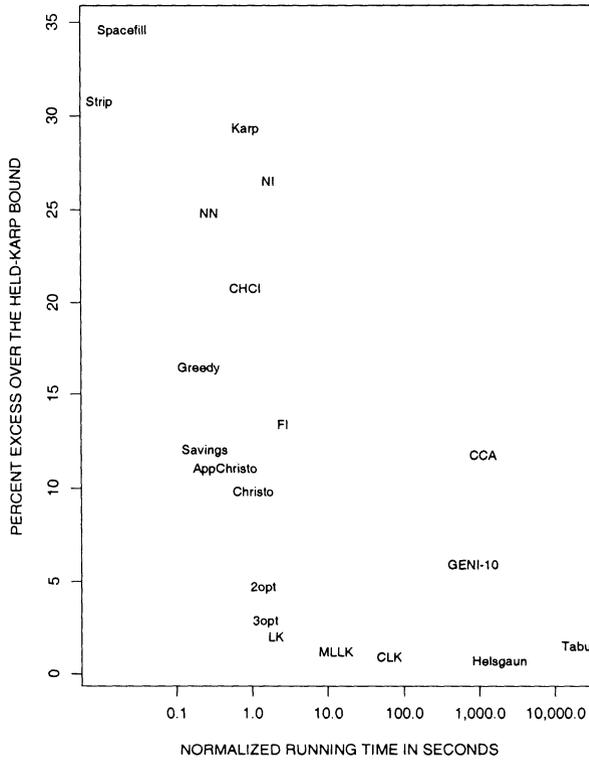


Figure 9.3. Average tradeoffs between tour quality and normalized running time for a variety of heuristics applied to 10,000-city Random Uniform Euclidean Instances. The full abbreviations for the heuristic names are given in Table 9.2 below and explained in Table 9.19 at the end of this chapter.

Heuristic	Excess Over HK	Time (Seconds)	Heuristic	Excess Over HK	Time (Seconds)
Spacefill	34.56	0.02	AppChristo	11.05	0.44
Strip	30.75	0.01	Christo-S	9.81	1.04
Karp-20	29.34	0.85	GENI-10	5.89	823.00
NI	26.50	1.71	2opt-JM	4.70	1.41
NN	24.79	0.28	3opt-JM	2.88	1.50
CHCI	20.73	0.83	LK-JM	2.00	2.06
Greedy	16.42	0.20	Tabu-SC-SC	1.48	18830.00
FI	13.35	2.59	MLLK-.1N	1.18	12.75
Savings	12.03	0.24	CLK-ABCC-N	0.90	63.91
CCA	11.73	1129.00	Helsgaun-.1N	0.69	1840.00

Table 9.2. Average tour quality and normalized running times for various heuristics on the 10,000-city instances in our Random Uniform Euclidean testbed.

been proven or which have received significant publicity will also be covered, since in these cases the very fact that they are dominated becomes interesting. For example, assuming that the triangle inequality holds, Nearest Insertion can never produce a tour longer than twice optimum, whereas NN can be off by a factor of $\Theta(\log N)$ [730], which makes the fact that the latter can be better in practice somewhat surprising.

Moreover, domination for one class of instances need not tell the full story. Table 9.3 summarizes the relative performances of Bentley’s implementations of Nearest Insertion and Nearest Neighbor as a function of instance size for our three geometric instance classes, represented by the shorthands U (Random Uniform Geometric instances), C (Random Clustered Geometric instances) and T (TSPLIB instances). Figure 9.4 presents a more detailed picture, with charts that depict the relative solution quality and running times of the two implementations for each of the geometric instances in our testbeds to which both could be applied. (The implementations were designed to exploit geometry as much as possible, but do not handle fractional coordinates.) Analogous tables and charts for other pairs of heuristics can be generated and viewed online via “Comparisons” page at the Challenge website. One can also generate charts in which the running time for a single heuristic is compared to various growth rates, just as the running times for Greedy were compared to $N \log N$ in Figure 9.1.

Average Percent Excess: NI over NN

	N=1000	3162	10K	31K	100K	316K	1M	3M	10M
U	-0.55	-0.15	1.37	2.22	2.86	2.85	2.95	3.31	3.31
C	-4.47	-3.89	-2.42	-2.91	-2.60	-3.04			
T	-1.41	-3.44	-2.73	-1.27	0.35				

Average Running Time Ratio: NI/NN

U	5.5	5.2	5.6	5.6	5.7	5.7	5.1	4.9	5.0
C	8.5	9.7	13.0	11.5	13.1	11.9			
T	6.2	5.7	6.5	8.5	10.0				

Table 9.3. Average comparisons between Nearest Insertion (NI) and Nearest Neighbor (NN) on our geometric testbeds. Bentley’s implementations [103] of both heuristics were used. A positive entry in the “Excess” table indicates that the NI tours are longer by the indicated percentage on average. As in all subsequent tables of this sort, the TSPLIB averages are over the following instances: `pr1002`, `pcb1173`, `r11304`, and `nrw1379` for $N = 1,000$, `pr2392`, `pcb3038`, and `fn14461` for $N = 3162$, `pla7397` and `bfd14051` for $N = 10K$, `pla33810` for $N = 31K$, and `pla85900` for $N = 100K$. These may not be completely typical samples as we had to pick instances that most codes could handle, thus ruling out the many TSPLIB instances with fractional coordinates.

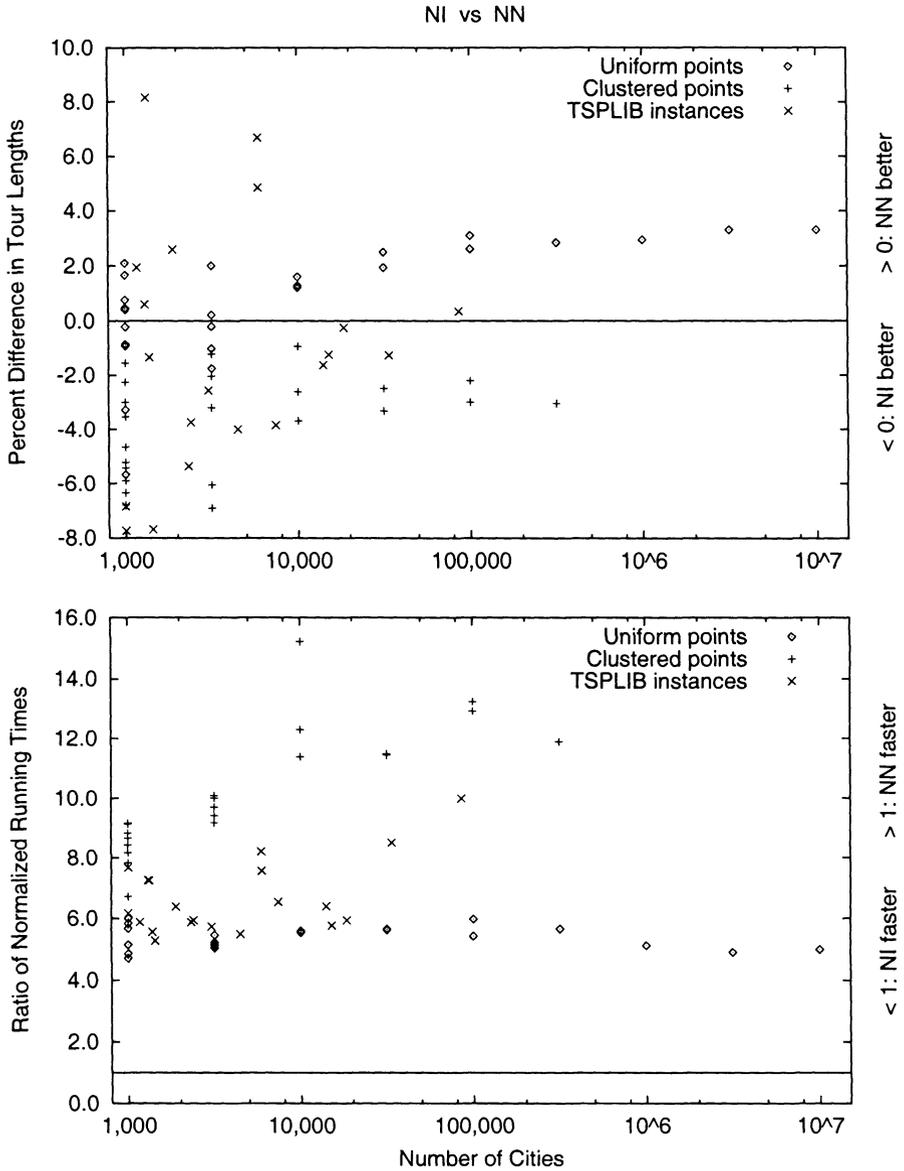


Figure 9.4. Tour length and normalized running time comparisons: Nearest Insertion versus Nearest Neighbor.

As to the Nearest Insertion versus NN comparison, we see that the tour length results for 10,000-city Uniform instances are echoed for larger instances from that class, but do not predict results for the other instance classes. Indeed, NI consistently provides better tours than NN for Clustered instances and is also better for a majority of the TSPLIB instances. NI *does* remain slower than NN (by a factor of 4 or more on the same machine for all instances), but for certain instance classes one might be willing to pay this price to get better tours. We thus cannot say that NI is consistently dominated by NN, although we will see many examples of consistent domination in what follows.

The body of this section is divided into seven parts, each covering a group of related heuristics. The first three subsections cover what are typically called *tour construction* heuristics, i.e., heuristics that incrementally construct a tour and stop as soon as a valid tour is created. The remaining sections concern heuristics with a *local search* component, i.e., heuristics that repeatedly modify their current tour in hopes of finding something better.

In Section 3.1, we consider tour construction heuristics designed more for speed than for quality. The Strip heuristic and the Spacefilling Curve heuristic, for example, do little more than read the instance and sort. Sections 3.2 and 3.3 cover the remainder of the classical tour construction heuristics, divided somewhat arbitrarily into those that build tours by adding edges one at a time, as in NN (Section 3.2), and those where the augmentation may involve *replacing* edges, as in Nearest Insertion and Christofides (Section 3.3). Since tour construction heuristics for the STSP are not covered in detail elsewhere in this book, we shall in these sections summarize what is known theoretically about these heuristics as well as discussing their empirical behavior.

The remaining sections cover local search heuristics, the subject of Chapter 8. Section 3.4 covers simple local search heuristics like 2-Opt and 3-Opt. Section 3.5 covers the famous Lin-Kernighan heuristic and its variants. Section 3.6 discusses various heuristics that involve repeated calls to a local search heuristic as a subroutine, such as the Chained Lin-Kernighan heuristic introduced by [563]. It also covers our one set of Tabu Search implementations, which operate in a similar fashion. The final Section 3.7 considers heuristics that take this one step further and use a heuristic like Chained Lin-Kernighan as a subroutine.

Although we do not have room to provide full descriptions of all the heuristics we cover, we present at least a high-level summary of each, mentioning relevant theoretical results and, where possible, pointers to sources of more detailed information. If implementation details can have a major impact on performance, we say something about these as well.

3.1. Heuristics Designed for Speed

In this section we cover heuristics for geometric instances of the STSP that are designed for speed rather than for the quality of the tour they construct. In particular, we restrict attention to heuristics whose observed total running time is within a small factor (10 or less) of the time simply to read the (x, y) coordinates for N cities using standard formatted I/O routines. The normalized times for the latter are shown in Table 9.4. Note that one can read instances much faster than this by using lower-level routines to exploit the fact that coordinates come in a known format. Using such an approach, one can speedup the reading of our 10,000,000-city instance by a factor of 80 or more [26]. This would have a significant impact on the overall speed of our fastest heuristics, which currently do not take this approach. The restriction to geometric instances, i.e., ones given by tuples of coordinates, is important: If one required the instance to be given by its full distance matrix, *many* of our heuristics would satisfy the above speed criterion, but could hardly be called “fast” given that instance reading itself would take $\Theta(N^2)$ time.

At present, three heuristics meeting the above criteria have received significant coverage in the literature: the Strip, Spacefilling Curve, and Fast Recursive Partitioning heuristics. In this section we cover all three, plus an obvious enhancement to the first. All are defined in terms of 2-dimensional instances but could in principle be generalized to geometric instances in higher dimensions. The results we report were all obtained on the same machine (as were the reading times mentioned above), which removes running-time normalization as an extra source of inter-heuristic variability. All the heuristics begin by making one pass through the data to determine minimum and maximum x and y coordinates and thus the minimum enclosing rectangle for the point set.

Strip. In this heuristic, we begin by dividing the minimum enclosing rectangle into $\sqrt{N/3}$ equal-width vertical strips and sorting the cities in each strip from top to bottom. We then construct a tour by proceeding from the leftmost strip to the rightmost, alternately traveling up one

Average Normalized Running Time in Seconds: **Read**

	N=1000	3162	10K	31K	100K	316K	1M	3M	10M
U	0.00	0.01	0.02	0.06	0.13	0.25	1.0	3.4	12
C	0.00	0.01	0.03	0.06	0.12	0.25			
T	0.00	0.00	0.03	0.06	0.12				

Table 9.4. Average normalized times for reading instances using the standard I/O routines of C, compiled for MIPS processors using gcc.

strip and down the next, with one final (long) edge back from last city in the rightmost strip to the first in the leftmost.

This heuristic can be traced back to 1959, when Beardwood, Halton, and Hammersley [94] introduced it as a tool in a proof about the average-case behavior of the optimal tour length. It is easy to see that Strip's tours can be as much as $\Omega(\sqrt{N})$ times optimum in the worst case. However, for points uniformly distributed in the unit square (a continuous version of our Uniform instance class), the expected length of the Strip tour length can be shown to be no more than $0.93\sqrt{N}$ [500]. Given that the expected Held-Karp bound for such instances is empirically asymptotic to $0.71\sqrt{N}$ [465], this means that Strip's expected excess for such instances should be less than 31%.

Results for Strip are summarized in Table 9.5. Note that for Uniform instances, the upper bound on average case excess mentioned above is close to Strip's actual behavior, but Strip's tours are much worse for the other two classes. Strip is fast, however: Even for the largest instances, its running time averages less than 3.5 times that for just reading the instance, and the time is basically independent of instance class (as are the times for all the heuristics covered in this section). Since most of Strip's computation is devoted to sorting, this implementation uses a variety of sorting routines, depending on instance size. For the largest instances, a 2-pass bucket sort using 2^{16} buckets is used. This means that theoretically the implementation should run in linear time for our instances, although in practice it appears to be a bit slower, presumably because of memory hierarchy effects.

It is fairly easy to see why Strip's tours for Clustered instances are poor: They jump between clusters far too frequently. For instances in TSPLIB something similar might be going on, but one might wonder whether some of its poor performance is just an artifact of the fact that

Average Percent Excess over the HK Bound: Strip

	N=1000	3162	10K	31K	100K	316K	1M	3M	10M
U	31.94	32.23	30.75	30.16	30.36	30.22	30.10	30.10	30.09
C	115.61	160.82	174.39	190.62	198.05	201.76			
T	61.33	36.26	73.03	91.86	73.28				

Average Normalized Running Time in Seconds

U	0.00	0.02	0.03	0.09	0.20	0.53	2.8	10.4	41
C	0.00	0.02	0.04	0.09	0.19	0.53			
T	0.01	0.01	0.04	0.09	0.18				

Table 9.5. Average performance of the Strip heuristic.

we chose *vertical* strips. To examine this question, we implemented a composite heuristic that applies both the original Strip heuristic and the variant that uses *horizontal* strips and returns the better result (*2-Way Strip*). Given that reading time is amortized across the two runs of Strip, the overall running time only goes up by a factor of 1.3 to 1.7. Unfortunately the average improvements are minor, with a few individual exceptions, such as an improvement from an excess of about 119% to one of 52% for the TSPLIB instance `r15915`. Details can be explored on the Challenge website. A more promising competitor to Strip is the following.

Spacefilling Curve (Spacefill). This heuristic was invented by Platzmann and Bartholdi [671]. The cities are visited in the order in which they would be encountered while traversing a spacefilling curve for the minimum enclosing rectangle. As with Strip, most of the time is spent simply in sorting. For full details see [671]. Platzmann and Bartholdi prove that the Spacefilling Curve heuristic can never produce tours that are worse than $O(\log N)$ times optimum. Bertsimas and Grigni [108] exhibit pointsets for which Spacefill is this bad. Again, however, one can get bounded average-case ratios. A probabilistic analysis in [671] shows that when cities are uniformly distributed in the unit square the asymptotic expected tour length is approximately 35% above the empirical estimate of the expected Held-Karp bound. (Interestingly, the ratio of the heuristic's tour length to \sqrt{N} does not go to a limit as $N \rightarrow \infty$, although the \liminf and \limsup are extremely close [671].) Table 9.6 presents results for the inventors' implementation.

As with Strip, the overall running time for Spacefill stays within a factor of 3.5 of that for merely reading an instance. Moreover, although Spacefill's average excess for Uniform instances matches the theoretical prediction and hence is 4-5 percentage points worse than that for

Average Percent Excess over the HK Bound: Spacefill

	N=1000	3162	10K	31K	100K	316K	1M	3M	10M
U	32.25	33.40	34.56	34.71	34.94	35.00	35.09	35.06	35.08
C	41.08	60.74	72.85	95.48	76.81	51.68			
T	45.36	40.27	36.03	40.97	37.39				

Average Normalized Running Time in Seconds

U	0.00	0.01	0.04	0.11	0.24	0.64	3.0	10.6	39
C	0.00	0.01	0.04	0.11	0.24	0.62			
T	0.00	0.01	0.04	0.11	0.23				

Table 9.6. Average performance of the Spacefilling Curve heuristic.

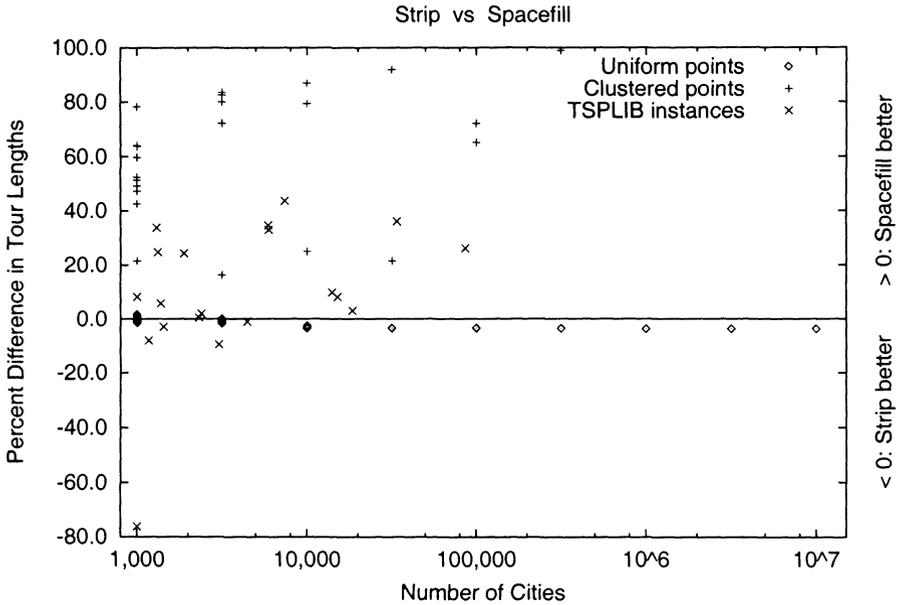


Figure 9.5. Tour quality comparisons for the Strip and Spacefilling Curve heuristics.

Strip, it is substantially better for the other two classes. Figure 9.5 provides a more detailed picture of the comparison. Based on these results, the Spacefilling Curve heuristic would seem to be the preferred choice, if one must choose only one of the two heuristics. It also would be preferred over our final candidate.

Fast Recursive Partitioning (FRP). In this heuristic, proposed by Bentley in [103], we begin by hierarchically partitioning the cities as in a K - d tree. This starts with the minimum enclosing rectangle and then recursively splits each rectangle containing more than $B = 15$ cities into two rectangles with roughly half as many cities. If the parent rectangle is longer than it is wide, the median x -coordinate for cities in the rectangle is found and a vertical split is made at this x value; otherwise the median y -coordinate is found and a horizontal split is made at this value of y . Call the final rectangles, all containing 15 or fewer cities, *buckets*. Nearest Neighbor tours are constructed for all the buckets, and these are then patched together to make an overall tour. FRP is effectively dominated by Spacefill, which is on average 2.5-3 times faster and is better for all but 4 instances in our testbed, usually by more than 10%. (The four exceptions are three of the 23 clustered instances and the TSPLIB instance dsj1000 which is itself a clustered instance, produced by an earlier version of our generator.)

3.2. Tour Construction by Pure Augmentation

In this section we cover heuristics that build their tours by adding one edge at a time, making each choice based on the length of the edge to be added. This is in contrast to **Strip** and **Spacefill**, which can be viewed as building their tours one edge at a time, but with choices based only on simple directional constraints. The class includes the Nearest Neighbor heuristic as well as the Greedy heuristic and several variants, including the lesser-known but quite effective “Savings” heuristic of [201].

As in the previous section, all the results we report were generated on same machine (using 196-Mhz R10000 MIPS processors), thus ensuring that running time comparisons will not be biased by normalization errors. However, it still may be dangerous to draw conclusions about the relative speeds of closely matched heuristics, since these may be highly implementation-dependent. We will illustrate this by presenting results for multiple implementations of the same heuristics (implemented by different programmers). These differ significantly in constant factors and asymptotic growth rates even though all follow the recommendations of Bentley’s influential papers [102, 103] that promoted the use of K - d trees (short for “ K -dimensional binary search tree” [101, 102]) and lazily updated priority queues for exploiting the geometric structure of instances and avoiding unnecessary work. These two are such a significant component of many of the implementations described in this section and later that they are worth a few more words.

K - d Trees. In defining the FRP heuristic in the previous section, we introduced the fundamental hierarchical partition of the instance space that underlies the K - d tree. (For K - d trees, however, we typically split any rectangle that contains more than 8 cities, as opposed to the bound of 15 used in FRP.) This partition is represented by a tree, with a vertex for each rectangle. For each vertex that represents a split rectangle, we store the coordinate of the median point that was used in splitting the rectangle (x if the split was left-right, y if the split was top-bottom), together with pointers to the vertices representing the two subrectangles into which it was split. (In *bottom-up* K - d trees, we also store a pointer to the parent of the given rectangle.) For a vertex corresponding to a final unsplit rectangle, we store a list of the cities in that rectangle. The partition and associated tree can be constructed in $O(N \log N)$ time.

Simple recursive routines can be used to search a K - d tree in various ways. We here mention three important ones. These all assume the existence of an auxiliary array `present[]` that tells us which of the cities are relevant to the current search. First, there is the *nearest neighbor* search: Given a city c , find the present city that is nearest to c . Second,

there is the *fixed-radius near neighbor* search: given a city c and a radius r , return (in some order) all those present cities c' such that $d(c, c') \leq r$. The third is *ball* search from city c , which assumes an additional array $\text{rad}[]$ of radii for all the cities and returns all those present cities c' for which $d(c, c') \leq \text{rad}[c']$, i.e., all those cities c' for which the ball of radius $\text{rad}[c']$ around c' contains c . For details on how these can be efficiently implemented, see [101, 102]. The first two searches involve the execution of $O(\log N)$ computer instructions for most data sets, while the third may take somewhat longer, depending on the number of relevant balls. The speed of all three can vary depending on the sophistication of the implementation and its interaction with the memory hierarchy of the machine on which the heuristic is run.

This section's simple heuristics require only the first of these three operations (or a slight variant on it). The others come into play for the more complicated heuristics of the next section. (An alternative to K - d trees, the Delaunay triangulation, was exploited by Reinelt in [710, 711]. This appears to be a competitive approach, but the results presented in [710, 711] are not sufficiently comparable to ours to yield firm conclusions. K - d trees, at any rate, offer substantially more power and flexibility.)

Lazily Updated Priority Queues. The use of this data structure in TSP heuristics was first suggested in [102, 103]. A priority queue contains items with associated values (the *priorities*) and supports operations that (1) remove the highest priority item from the queue and deliver it to the user (a “pop”), (2) insert a new item, (3) delete an item, and (4) modify the priority of an item (an “update”). Algorithms textbooks contain a variety of implementations for this data structure, most of which support all the operations in time $O(\log N)$ or less, but with different tradeoffs and constant factors. The choice can have a significant effect on running time. A major additional savings is possible if we can reduce the number of updates actually performed, which is what happens with lazy evaluation. This technique can be used if we know that no update will ever *increase* a priority. In this case, we need not perform an update when it first takes effect, but only when the popped (highest priority) item has an outdated priority. In this case, that item's priority is reevaluated and it is reinserted into the queue.

We are now prepared to describe this section's heuristics and how they are implemented.

Nearest Neighbor (NN). We start by picking a initial city c_0 . Inductively, suppose $i < N - 1$ and $c_0, c_1 \dots, c_i$ is the current partial tour. We then choose c_{i+1} to be the nearest city to c_i among all those cities not yet present in the tour. If $i = N - 1$ we add the edge $\{c_{N-1}, c_0\}$, thus

completing the tour. For non-geometric instances, this heuristic would take time $\Theta(N^2)$, but for geometric instances the use of K - d trees can reduce this to something like $O(N \log N)$ in practice.

Double-Ended Nearest Neighbor (DENN). We start by picking an initial city c_0 . Inductively, suppose a and b are the endpoints of the current partial tour and that it does not yet contain all the cities. If a' and b' are the nearest non-tour cities to a and b respectively, we choose the one that is closest to its respective endpoint and add the corresponding edge to the tour. If all the cities *are* in the tour, we add the edge $\{a, b\}$ and are done. This heuristic can be implemented to run almost as fast as NN in practice, since we only need to compute a nearest neighbor when the tour gains a new endpoint or when an endpoint's previous nearest neighbor is added to the other end of the tour.

Greedy. We start by sorting all the potential tour edges $\{c, c'\}$ in order of increasing length. We then build a tour, viewed as a set of edges, by going through the edges in order, starting with the shortest, and adding $\{c, c'\}$ so long as neither c nor c' already has degree 2 and the new edge does not complete a cycle with fewer than N vertices.

As described, the implementation would take time $\Theta(N^2 \log N)$, and that is the time that would be required for non-geometric instances. For geometric instances, this can be reduced by combining K - d trees and nearest neighbor searches with a lazily updated priority queue, as suggested by [102, 103]. This is done as follows. After first constructing the K - d tree, we find the nearest neighbor c' for each city c and put the ordered pair (c, c') in the priority queue with priority $-d(c, c')$. Thus the queue contains only N entries and the highest priority entry corresponds to the shortest edge, i.e., the first that **Greedy** would add to its tour. As we proceed, we will mark a city as *present* if it does not have degree 2 in the current tour. If c is a city with degree 1 in the tour, we will let $\text{end}[c]$ denote the city at the other end of the tour path starting with c . Note that we could build the **Greedy** tour in just $N - 1$ pops if we maintained the property that at all times the priority queue contained, for each city c that is currently present, the nearest *eligible* neighbor, i.e., the nearest present city other than $\text{end}[c]$.

Unfortunately, maintaining this property might require many updates after each pop. A single city c' can be the nearest eligible neighbor for many other present cities. When c' attains degree 2, it will no longer be eligible and each city c that thought c' was its nearest eligible neighbor will have to find a new partner. Note, however, that whenever the nearest neighbor of a city c needs to be updated, it will be replaced by a new city that is at least as far away from c as the city it replaced. So we can do lazy updating. When we pop the highest priority item in

the queue (c, c') , there are two cases. If c already has degree 2 in the tour, we simply discard this pair and pop the next one. If c has degree at most 1 in the tour and c' is present and not equal to $\text{end}[c]$, we can add edge $\{c, c'\}$ to the tour. Otherwise, we temporarily mark $\text{end}[c]$ (if it exists) as “not present,” find a new nearest present neighbor c'' for c , insert (c, c'') in the queue, reset $\text{end}[c]$ (if it exists) to “present,” and pop the new highest priority pair.

Boruvka. This heuristic is a variant on **Greedy** devised by Applegate, Bixby, Chvátal, and Cook in analogy with the classic minimum spanning tree of O. Borůvka [132]. As with the above **Greedy** implementation, we start by computing the nearest neighbor for each city. Instead of putting the resulting pairs into a priority queue, however, we simply sort them in order of increasing edge length. We then go through the list, edge by edge, adding each to the tour we are building so long as we legally can do so. In other words, when a pair (c, c') is encountered where c' is no longer eligible, we discard the pair without updating even if c still hasn't attained degree 2. After we have gone through the whole list, we probably won't yet have a tour, so we repeat the process again, this time restricting attention to cities that do not yet have degree 2 and eligible neighbors. We continue to repeat the process until a tour is constructed. In comparison to **Greedy**, this heuristic replaces priority queue overhead with simple sorting, but may have to do more nearest neighbor searches. It is not *a priori* evident whether tours should be better or worse.

Quick Boruvka (Q-Boruvka). This variant, also due to Applegate, Bixby, Chvátal, and Cook, dispenses with the sorting step in **Boruvka**, presumably trading tour quality for an increase in speed. We go through the cities in some arbitrary fixed order, skipping a city if it already has degree 2 and otherwise adding an edge to the nearest eligible city. At most two passes through the set of cities will be required.

Savings. This is a specialization to the STSP of a more general vehicle routing heuristic proposed by Clarke and Wright in [201]. Informally, it works by starting with a pseudo-tour, consisting of a multigraph that has two edges from an arbitrary *central city* c_0 to each of the other cities. We then successively look for the best way to “shortcut” this graph by replacing a length-2 path from one (non-central) city to another by a direct link. In practice, the Savings heuristic works like **Greedy**, except with a surrogate distance function. For any pair of cities c, c' other than c_0 , the surrogate distance function is $D(c, c') = d(c, c') - d(c, c_0) - d(c_0, c')$. Given a K - d tree, nearest neighbors under this surrogate distance function can be computed using a slightly more complicated version of the standard nearest neighbor search, as shown in [461]. The only other

difference from **Greedy** is that c_0 is not put in the priority queue, and we stop growing the tour when it contains $N - 2$ edges, at which point it must be a path, which we can complete to a tour by adding the edges from c_0 to its two endpoints.

Theoretical worst-case results have been proved for several of these heuristics, assuming the triangle inequality, i.e., that for all triples of cities (c_1, c_2, c_3) , $d(c_1, c_2) \leq d(c_1, c_3) + d(c_3, c_2)$. For none of these heuristics are the tours guaranteed to be within a constant factor of optimum, but we can provide some bounds. **NN** can be shown never to produce a tour longer than $(1 + \lceil \log N \rceil)/2$ times optimum, and there are instances that force it to generate tours roughly $2/3$ that long [730]. **Greedy** never produces a tour longer than $(1 + \log N)/2$ times optimum [633] (roughly the same upper bound as for **NN**), but the worst instances known only cause it to produce tours that are $(\log N/3 \log \log N)$ times optimum [325]. **Savings** never produces a tour longer than $(1 + \log N)$ times optimum [633] (a weaker bound than for the other two heuristics), but the worst examples known produce tours that are again only $(\log N/3 \log \log N)$ times optimum [325]. We are unaware of worst-case results for the relatively more recent **Boruvka** variants, but the bounds for these are likely to be no better than those for **Greedy**.

Figure 9.6 graphs the average tour quality as a function of N for the six heuristics described in this section and our two classes of random geometric instances. For both classes it is typical of most of the heuristics we cover that the average percentage excess over the Held-Karp bound appears to approach an asymptotic limiting value, although those limits are usually different for the two classes. For Uniform instances, the limiting values for **NN** and **DENN** appear to be roughly 23% above the Held-Karp bound, compared to 15% for **Q-Boruvka**, 14% for **Greedy** and **Boruvka**, and 12% for **Savings**. **DENN** appears to yield slightly better averages than **NN** for the smaller instances but its advantage vanishes once $N > 10,000$. (Variations after that point are attributable to the small number of instances in our samples). **Greedy** appears to be slightly better than **Boruvka** for the smaller instances, but this advantage disappears by the time $N = 100,000$. All the heuristics perform significantly more poorly for the Clustered instances, but the relative asymptotic ranking remains the same. For **TSPLIB** instances, the tour quality tends to lie between these two extremes, except that **Savings** is typically 1-2% better on the larger **TSPLIB** instances than it is even for Uniform instances of similar size.

When we order the heuristics by running time, they appear in roughly reverse order, which implies that no one of them is dominated by any of the others. Table 9.7 lists normalized running times for Uniform in-

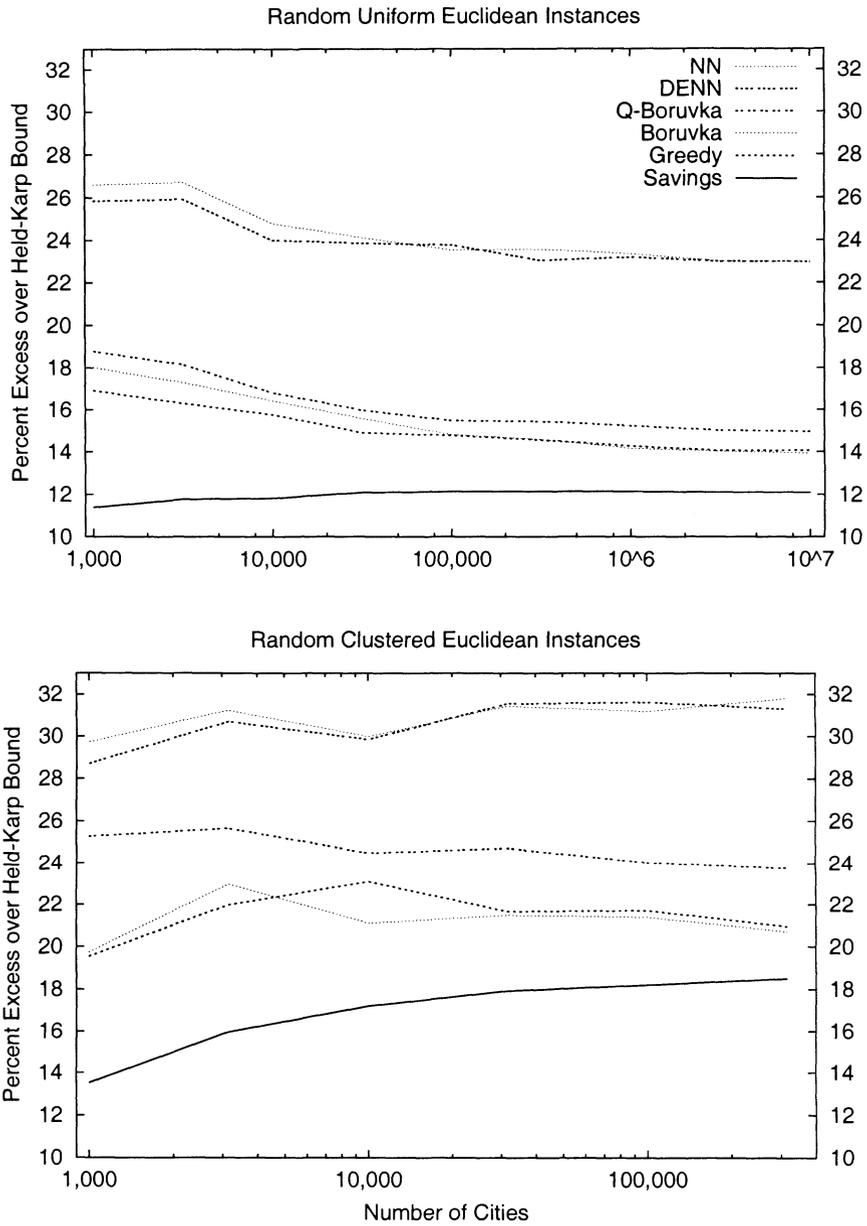


Figure 9.6. Average percentage excess for pure augmentation heuristics. (For an explanation of the abbreviations, see the text or Table 9.19.) Note that the ranges of N are different for the two classes of instances.

stances as a function of N . Times for Clustered instances are roughly the same. Those for TSPLIB instances tend to be faster, possibly because the added structure of these instances limits the breadth of the K - d tree nearest-neighbor searches. The table covers three families of implementations: Bentley's (-B) implementations of NN and Greedy, the Johnson-McGeoch (-JM) implementations of the same heuristics plus Savings, and the Concorde (-ABCC) implementations of those two plus the two Boruvka variants. (The suffixes are the implementers' initials.)

For codes with common implementers, the code that produces better tours typically takes longer time for all values of N , with two exceptions: DENN takes about the same time as NN (as predicted), and the Johnson-McGeoch implementation of Savings sometimes beats their implementation of Greedy. Although the nearest-neighbor searches are more complicated under Savings than under Greedy, this is balanced by the fact that far fewer of them need to be made and the two heuristics end up taking roughly the same overall time. Generally, the time for Greedy/Savings is 2 to 5 times that for NN, with the biggest differences occurring for the Concorde implementations.

Cross-family comparisons are more problematic, presumably because of implementation differences. The Bentley and Concorde implementations exploit up-pointers in their K - d trees, whereas the Johnson-McGeoch implementations do not. Up-pointers add constant-factor overhead but can greatly reduce the depth of searching. As a result, the Johnson-McGeoch implementations are faster than other two when $N \leq 100,000$ but slower when N is larger. Bentley's implementations are in C++ while the other two are in C, which might explain in part why Bentley's implementations lose to Concorde on both NN and Greedy.

The observed running times for all the implementations appear to have two components: one that grows more slowly than $N \log N$ and

N =	1000	3162	10K	31K	100K	316K	1M	3M	10M
NN-ABCC	0.01	0.03	0.10	0.27	0.88	2.73	14.2	58.6	247
NN-B	0.02	0.10	0.32	0.82	2.17	5.13	25.5	100.9	400
NN-JM	0.01	0.02	0.08	0.22	0.53	5.34	25.6	103.5	453
DENN-B	0.02	0.10	0.32	0.83	2.08	5.18	25.8	102.2	405
Q-Boruvka-ABCC	0.01	0.04	0.12	0.31	1.13	4.02	22.4	96.2	404
Boruvka-ABCC	0.02	0.05	0.18	0.65	2.60	7.46	36.9	151.4	597
Greedy-ABCC	0.02	0.07	0.27	1.12	4.55	12.64	59.2	221.8	863
Greedy-B	0.05	0.19	0.62	1.77	5.14	12.77	60.4	232.5	930
Greedy-JM	0.02	0.06	0.20	0.81	4.05	21.28	100.8	357.3	1450
Savings-JM	0.02	0.08	0.26	0.83	3.13	21.02	99.6	385.5	1604

Table 9.7. Normalized running times in seconds for Pure Augmentation heuristics and Random Uniform Euclidean instances.

dominates when $N \leq 100,000$, and one that grows faster than $N \log N$ and dominates once $N > 100,000$. This latter component in fact appears to be growing faster than $N \log^2 N$, although no worse than $O(N^{1.25})$. The relative importance of these two components and their crossover point depend on the heuristic and the implementation. Determining the causes of this behavior is an interesting question for future research.

With respect to tour quality, there is no appreciable difference between the various implementations of NN and Greedy. This is as should be expected, given the well-defined nature of those heuristics. Different implementations do not, however, always yield the same tours. This is because of different tie-breaking rules and because the output of NN depends on the starting city chosen.

Our overall conclusion is that, although there are few cases of pure domination here, three of the six heuristics adequately cover the range of trade-offs: DENN, Boruvka, and Savings (with the K - d tree implementation chosen based on the expected size of the instances to be handled). In most real-world applications, we would expect Savings to be fast enough to supplant the other two. The above conclusions assume that one is looking for a stand-alone heuristic. As we shall see in Sections 3.4 and 3.5, different conclusions may hold if one is choosing a method for generating starting tours in a local search heuristic.

The story for non-geometric applications may also differ, and we are less able to provide insight here. The only non-geometric implementations we have are for Greedy and NN, and our testbed of non-geometric instances consists mostly of Random Matrices, whose relevance to practice is suspect. For what it is worth, Greedy continues to provide substantially better tours than NN for these instances and now takes roughly the same time. Unfortunately, that time is now $\Theta(N^2)$, and both heuristics produce tour lengths whose average ratio to the optimum appears to grow with N and exceeds 2 by the time $N = 10,000$. See [461].

3.3. More Complex Tour Construction

In this section we consider somewhat more complicated heuristics, but ones that still build tours incrementally. Many of these heuristics, even ones with appealing theoretical performance guarantees, are dominated by Savings. Results for the dominated heuristics will not be covered in full detail here, although they can be viewed at the Challenge website.

Nearest Insertion and its Variants (NI,NA,NA⁺). Start with a partial tour consisting of some chosen city and its nearest neighbor. Then repeatedly choose a non-tour city c whose distance to its nearest neighbor among the tour cities is minimum, and insert it as follows:

- **Insertion Rule (NI)**. Insert c between the two consecutive tour cities for which such an insertion causes the minimum increase in tour length.
- **Addition Rule (NA)**. Insert c next to its nearest neighbor in the tour on the side (before or after) that yields the minimum increase in the tour length.
- **Augmented Addition Rule (NA⁺)**. Insert c as in NI, but restrict attention to pairs of consecutive tour cities at least one of which is no further from c than twice the distance to c 's nearest neighbor in the tour.

NI, NA, and NA⁺ are all guaranteed to produce tours that are no longer than $(2 - \frac{2}{N})$ times optimum assuming the triangle inequality holds, and all can produce tours that bad [730]. As explained by Bentley in [103], which introduced the augmented addition rule, they can all be implemented to exploit geometry, although the process is complicated. (NI requires a ball search and NA⁺ requires a fixed-radius near neighbor search.) As might be expected, this added complexity (even for NA) means that Bentley's implementations of all three heuristics are substantially slower than *Savings*. They also produce worse tours for all instances in our geometric testbeds. For Uniform instances NI and NA⁺ have an average percentage excess over the Held-Karp bound that approaches 27% as compared to 12% for *Savings*, while the limiting percentage for NA is 32.5%. Thus all three variants are dominated by *Savings*. The same holds for the following family of theoretically interesting heuristics.

Cheapest Insertion and its Variants (CI,CHCI). In Cheapest Insertion (CI), we start with a partial tour consisting of a chosen city and its nearest neighbor. We then repeatedly choose a triple a, b, c of cities such that a and b are adjacent in the current tour, c is a non-tour city, and the increase in tour length that would occur if c were inserted between a and b is minimized, and perform that insertion. Assuming the triangle inequality, CI obeys the same $2 - (2/N)$ times optimum bound as Nearest Insertion. The "Convex Hull" variant CHCI starts by computing the convex hull of the cities and creating a starting tour consisting of these in radial order. CHCI trivially obeys a $3 - (2/N)$ bound, given the result for CI.

Implementations can again take advantage of geometry, as explained in [461]. For CHCI, the convex hull can be found by a linear-time algorithm such as Graham's [394]. Even so, the Johnson-McGeoch implementations of CI and CHCI remain substantially slower than their implementation of *Savings* and are almost universally worse. (CHCI is

slightly better than **Savings** on one 1,000-city Clustered instance.) **CHCI** tends to produce better tours than **CI**, but the advantage shrinks as N grows. For Uniform instances, the average percentage excess over the Held-Karp bound for both **CHCI** and **CI** tends to about 22% versus 12% for **Savings**.

Double Minimum Spanning Tree (DMST). Construct a multigraph consisting of two copies of a minimum spanning tree for the cities. This graph must have an Euler tour, i.e., a (not necessarily simple) cycle that includes every edge exactly once. Construct one and convert it into a Hamiltonian cycle by taking shortcuts to avoid visiting cities more than once. Assuming the triangle inequality holds, this heuristic obeys the same $2 - (2/N)$ worst-case bound as do **Nearest** and **Cheapest Insertion**.

Again, geometry can be exploited in implementing **DMST**, in particular for constructing the initial **MST**. The Euler tour can be found in linear time, as can the shortcuts needed to produce the tour. Unfortunately, we still end up slower than **Savings**, and even if we use the “greedy shortcut” procedure described below in the context of the **Christofides** heuristic, **DMST** still produces substantially worse tours than those for **Savings**. For Uniform instances the average percentage excess tends toward 40%, and the results for the other classes are comparable.

Karp’s Partitioning Heuristic (Karp). As in K - d tree construction (and in the **FRP** heuristic of Section 3.1), we begin by recursively partitioning the cities by horizontal and vertical cuts through median cities, although now the median city is included in *both* of the subsets of cities created by the cut through it. This process is continued until no more than C cities are in any set of the partition (C is a parameter). Using the dynamic programming algorithm of Bellman [95], we then optimally solve the subproblems induced by the sets of cities in the final partition. Finally, we recursively patch the solutions together by means of their shared medians. For fixed C , this takes $O(N \log N)$ time.

This heuristic was proposed by Karp in his paper [497], which analyzed the average-case behavior of a closely related, non-adaptive heuristic. For this non-adaptive variant and any $\epsilon > 0$, there exists a C_ϵ such that for Uniform instances the expected ratio of the heuristic’s tour length to the optimal tour length is asymptotically no more than $1 + \epsilon$. Unfortunately, C_ϵ grows linearly with $1/\epsilon$, and the running time and space requirements of the dynamic programming subroutine are both exponential in C . (See also [500] and Chapter 7.)

The adaptive version of the heuristic we test here is likely to produce better tours and be more robust in the presence of non-uniform data, but this has not been rigorously proved. It suffers from the same drawbacks as far as C is concerned, however, with the largest value that has proved

feasible being $C = 20$. Given that the average number of cities in the final partitions can vary from 10 to 20 depending on the value of N , this heuristic has a wildly varying running time as a function of N . The average quality of the tours it produces for Uniform instances also fails to go to a limit as $N \rightarrow \infty$. As might be expected, the best results correspond to the worst running times, which themselves can be hundreds of times worse than those for **Savings**. However, even those best results are far worse than those for **Savings**: the liminf of the Uniform instance excesses is larger than 20% and the results for Clustered and TSPLIB instances are substantially worse.

The failings of this approach can be ameliorated if one settles for near-optimal rather than optimal solutions to the final subproblems. This was the approach taken by FRP, but it used a small value for C and a poor heuristic (NN). If one instead uses a large value for C and one of the much more powerful heuristics we describe later in this chapter, one could do much better. Indeed, this might be a plausible first choice for coping with instances that are too big to be handled all-at-once in main memory.

Many other tour construction heuristics that have been proposed in the literature are also dominated by **Savings** (for example, Litke's recursive clustering heuristic [564] and the Greatest Angle Insertion heuristic of Golden and Stewart [388], both implemented to exploit geometry in [461] and covered on the Challenge website). However, none of these are of independent theoretical interest. For the remainder of this section, we concentrate on heuristics that are *not* dominated by **Savings**. We first consider variants on Nearest Insertion that lack its strong theoretical guarantees but perform much better in practice.

Random and Farthest Insertion Variants (RI,RA,RA⁺,FI,FA,FA⁺). These heuristics differ from their "Nearest" variants mainly in the choice of city to add. In the "Random" variants the city is simply chosen randomly. In the "Farthest" variants we add the city c with the largest value of $\min\{d(c, c') : c' \text{ is in the tour}\}$. For both sets of variants, we start with a tour consisting of the two maximally distant cities.

The best guarantee currently provable for these heuristics (assuming the triangle inequality) is that all provide tours that are no more than $O(\log N)$ times optimum. At present we do not know whether this bound is tight. The worst examples known for the Random variants were obtained by Azar [50]: Euclidean instances for which with high probability the heuristics produce tours of length $\Theta(\log \log N / \log \log \log N)$ times optimum. The worst examples known for the Farthest variants were obtained by Hurkens [455] and only yield ratios to optimum that approach 6.5 (triangle inequality) or 2.43 (2-dimensional Euclidean).

As with the Nearest variants, these heuristics can be implemented to exploit K - d trees. The Random variants save work in identifying the city to insert and so are fastest. The Farthest variants require additional work in order to find the point to be inserted, but this can be done using a K - d tree on the tour cities and a lazily updated priority queue that for each non-tour city lists the distance to the closest tour city (at the time the entry was computed). As an indication of the relative asymptotic performance of all these variants, see Table 9.8 which summarizes the average results for Bentley’s implementations of them on 100,000 Uniform instances. For comparison purposes the results for the Johnson-McGeoch implementation of **Savings** are also included.

Note first that in each family Augmented Addition takes less than twice as much time as Addition, but provides substantially better tours, especially in the cases of the Random and Farthest families. Second, note that in each family the Augmented Addition variants produce nearly as good tours as do their Insertion siblings at a fraction of the running time cost. Unfortunately, the only one of these heuristics that is clearly competitive with **Savings** in running time (**RA**) produces very poor tours.

Uniform instances, however, don’t tell the full story. As an illustration of the total picture, see Figure 9.7, which for all instances in our geometric testbeds without fractional coordinates compares the tour lengths found by **FI** and **Savings**. Although **Savings** typically has an even greater advantage for **TSPLIB** instances than for Uniform ones, a different story holds for the **Clustered** instance class. For these instances, **RA**⁺, **RI**, **FA**⁺, and **FI** all find better tours on average than does **Savings**, ranging from roughly a 1.5% improvement under **RA**⁺ to 3.0% under **RI** and 3.5% improvement under **FA**⁺ and **FI**. We should also point out that **RA**⁺ has another advantage. Although it is slower than **Savings** when $N \leq 100,000$, its running time is similar to that of Bentley’s implementation of **Greedy**, in that it becomes faster than **Savings** for larger N (for roughly the same implementation-dependent reasons).

Heuristic	NA	NA ⁺	NI	RA	RA ⁺	RI	FA	FA ⁺	FI	Sav
Excess %	32.5	27.1	27.1	40.5	15.4	15.0	43.7	13.6	13.4	12.1
Seconds	6.6	8.6	12.3	3.2	5.7	20.3	10.7	13.6	27.7	3.1

Table 9.8. Average percentage excesses over the Held-Karp bound and normalized running times for Bentley’s implementations of Insertion, Addition, and Augmented Addition heuristics applied to 100,000-city Random Uniform Euclidean instances. For comparison purposes, the last column gives results for the Johnson-McGeoch implementation of **Savings**.

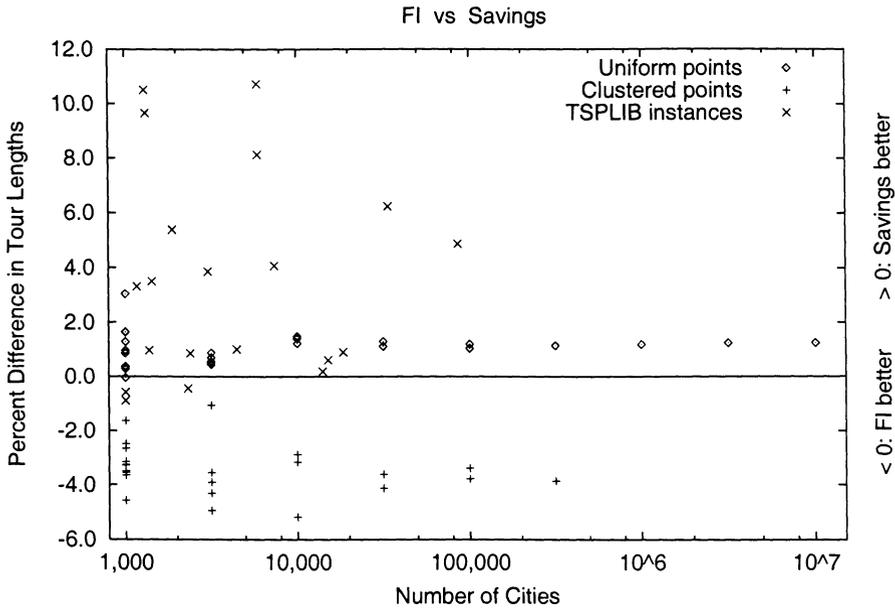


Figure 9.7. Tour quality comparisons for Farthest Insertion and Savings heuristics.

We thus can conclude that these heuristics and **Savings** are all technically incomparable in that none totally dominates any of the others. Given the artificial nature of the Clustered instances, however, one would probably still choose **Savings** if one could use only one heuristic.

CCA. This is an abbreviation for “Convex Hull, Cheapest Insertion, Angle Selection,” a heuristic proposed by Golden and Stewart in [388] and claimed to be the best tour construction heuristic in that study. As in **CHCI**, one starts by constructing the convex hull of the cities and proceeds by successively inserting the remaining cities. The choice of insertion is more complicated however. For each non-tour city c , one determines the pair (a_c, b_c) of adjacent tour cities between which c could be inserted with the least increase in overall tour length. We then select that c that maximizes the angle between the edges $\{a_c, c\}$ and $\{c, b_c\}$ and insert it between a_c and b_c in the tour.

Nothing is known theoretically about this heuristic, and its complexity makes it difficult (if not impossible) to exploit geometry when implementing it. However, the results reported in [388] were impressive, even if the largest instance considered had only 318 cities. To see how it handles larger instances, Johnson and McGeoch [461] constructed a non-geometric implementation, which we tested. Running times as expected are non-competitive, growing at a rate somewhere between $\Theta(N^2)$ and

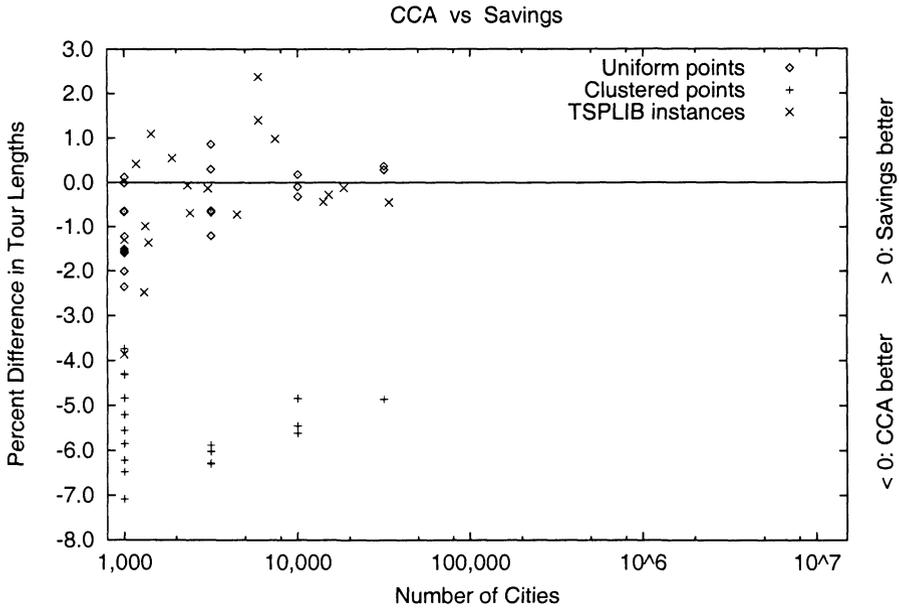


Figure 9.8. Tour quality comparisons for CCA and Savings heuristics.

$\Theta(N^{2.5})$ and taking over 3 normalized hours for $N = 33,810$ cities (the largest instance we tried) versus 0.44 seconds for **Savings**. However, as seen in Figure 9.8, **CCA** does find better tours for instances of all three types, even if its advantage for Uniform and TSPLIB instances seems to be vanishing as N grows. In particular, the limiting value for the average percentage excess on Uniform instances seems likely to exceed 12.5% (its value at $N = 31,623$), whereas that for **Savings** is 12.1%

The Christofides Heuristic and its Variants. For our final collection of tour construction heuristics, we consider variants on the famous heuristic of Christofides [189], which currently has the best worst-case guarantee known for any polynomial-time TSP heuristic, assuming only the triangle inequality.

The Christofides heuristic is a clever improvement on the Double Minimum Spanning tree (**DMST**) heuristic described earlier. In the standard version of Christofides (**Christo-S**), we start by computing a minimum spanning tree, as in **DMST**. However, instead of adding a second copy of the MST to get an Eulerian multigraph, here we add a minimum weight matching on the odd-degree vertices of the MST, which optimally grows the MST to an Eulerian multigraph. We then find an Euler tour and traverse it, shortcutting past previously visited vertices as in **DMST**. This leads to an improved guarantee: assuming the triangle inequality, the tour produced will never be more than $3/2$ times the optimal tour length

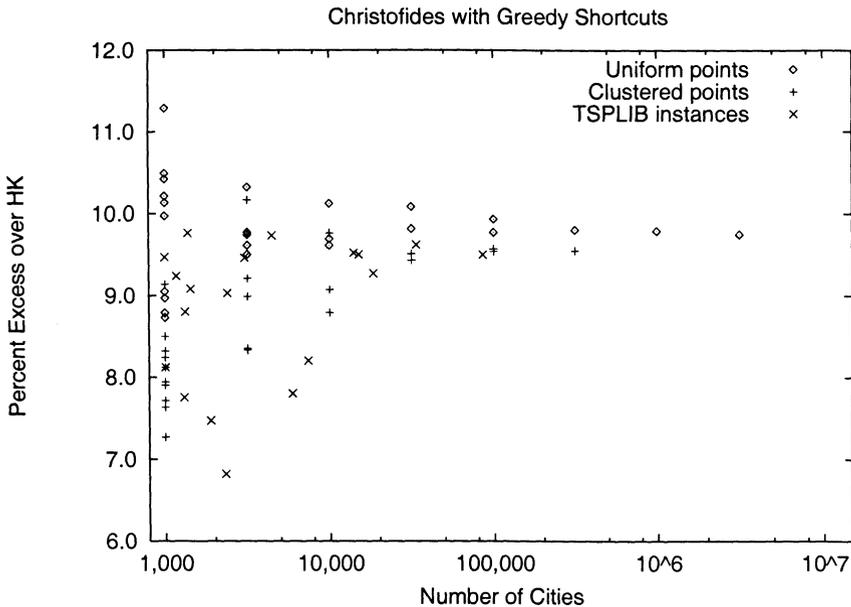


Figure 9.9. Tour quality for Christofides with greedy shortcuts.

(a bound that is asymptotically attainable by 2-dimensional Euclidean instances, as shown by Cornuejols and Nemhauser [222]).

Unfortunately, this improvement in worst-case behavior comes at a price. The running time for Christofides is dominated by that for computing the minimum weight matching, and the best algorithms known for this have $O(N^3)$ running times, as compared to the (non-geometric) worst-case running time of $O(N^2 \log N)$ for Savings. Fortunately, in practice we can use matching codes that exploit geometry to run much more quickly. For the implementations of Christofides studied here, we used the code of Cook and Rohe [207], together with a K - d tree based minimum spanning tree algorithm. With these, the observed running time appeared to be $O(N^{1.25})$ and we were able to handle instances with 3 million cities in normalized time of about an hour, only 10 times longer than for Savings. (Memory problems prevented us from successfully running the Christofides code on larger instances.)

The tour quality results for this standard version of Christofides are disappointing, however. Like Farthest Insertion it beats Savings on the Clustered instances, but it does worse for Uniform instances and most of the TSPLIB instances. Indeed, its average percentage excess for Uniform instances appears to approach 14.5%, which is worse than the limits for FA and FI as well as the 12.1% for Savings.

This is not the end of the story, however. A natural question is whether we might somehow do a better job of shortcutting in the final phase of the heuristic. As shown by Papadimitriou and Vazirani in [657], it is NP-hard to find the optimal way to shortcut the Euler tour. However, there are heuristics that do significantly better than the naive approach taken in the standard implementation. The “greedy shortcut” version of Christofides (**Christo-G**) examines the multiply visited cities in some arbitrary order and for each chooses the current best of the possible shortcuts. This version runs in essentially the same time as the standard one and yet finds better tours than both **Savings** and **Farthest Insertion** on all but one instance each (and on those two instances, it is only 0.06% worse). It is also more consistent. Figure 9.9 plots the percentage excess above the Held-Karp bound for **Christo-G** on all the integral-coordinate geometric instances in our testbeds (except the 10,000,000-city instance which we couldn’t run). For all three classes **Christo-G**’s excesses for larger instances lie between 9 and 10%. The limiting percentage for Uniform instances appears to be about 9.8%, a substantial improvement over any of the other heuristics we have covered. **Christo-G** also outperforms **CCA** on all instances with more than 3,162 cities and performs better on average except in the case of **Clustered** instances with 1,000 or 3,162 cities (and is of course *much* faster).

Another modification of Christofides that has been proposed is to replace the initial MST with the one-tree obtained in the process of computing the Held-Karp bound using Lagrangean relaxation [444, 445]. This approach was combined with greedy shortcuts by Andre Rohe [729] in an implementation we shall call **Christo-HK**. The Lagrangean relaxation scheme used by Rohe involves many spanning tree computations over weighted sparse graphs derived from the Delaunay triangulation of the cities. Although no attempt is made to run this process to convergence, it still takes substantially longer than simply computing a single MST for the cities, so that asymptotically **Christo-HK** seems to be some 4-8 times slower than **Christo-G**. It does find significantly better tours, however: Its average excess over the HK bound appears to go toward 6.9% for Uniform instances and 8.6% for Clustered instances.

If one is unwilling to pay the running time penalty of **Christo-G** (much less that of **Christo-HK**), it is natural to ask how well one might do if one sped up the bottleneck matching phase of Christofides’ algorithm by using a fast heuristic to get a good but not-necessarily-optimal matching (while still using greedy shortcuts). We have implemented such a heuristic using a K - d tree based greedy matching procedure followed by 2-opting, i.e., looking for pairs $\{a, b\}$, $\{c, d\}$ of matched cities for which changing partners (to $\{a, c\}$, $\{b, d\}$) shortens the matching, until

Average Percent Excess over the HK Bound: Uniform Instances

N =	1000	3162	10K	31K	100K	316K	1M	3M	10M
RA ⁺	13.96	15.25	15.04	15.49	15.43	15.42	15.48	15.47	15.50
Chr-S	14.48	14.61	14.81	14.67	14.70	14.49	14.59	14.51	–
FI	12.54	12.47	13.35	13.44	13.39	13.43	13.47	13.49	13.49
CCA	10.11	11.47	11.73	12.46	–	–	–	–	–
Sav	11.38	11.78	11.82	12.09	12.14	12.14	12.14	12.10	12.10
ACh	11.13	11.00	11.05	11.39	11.24	11.19	11.18	11.11	11.11
Chr-G	9.80	9.79	9.81	9.95	9.85	9.80	9.79	9.75	–
Chr-HK	7.55	7.33	7.30	6.74	6.86	6.90	6.79	–	–

Average Normalized Running Time in Seconds

RA ⁺	0.06	0.23	0.71	1.9	5.7	13	60	222	852
Chr-S	0.06	0.26	1.00	4.8	21.3	99	469	3636	–
FI	0.19	0.76	2.62	9.3	27.7	65	316	1301	5345
CCA	4.88	82.09	1129.85	14015	–	–	–	–	–
Sav	0.02	0.08	0.26	0.8	3.1	21	100	386	1604
ACh	0.03	0.12	0.44	1.3	3.8	28	134	477	2036
Chr-G	0.06	0.27	1.04	5.1	21.3	121	423	3326	–
Chr-HK	1.00	3.96	14.73	51.4	247.2	971	3060	–	–

Table 9.9. Results for the more powerful tour construction heuristics on Random Uniform Euclidean instances. Sav, ACh, and Chr stand for Savings, AppChristo, and Christo, respectively.

no more can be found. Our 2-opting procedure uses the speedup tricks for the 2-opt TSP heuristic described in the next section. The resulting “Approximate Christofides” heuristic (AppChristo) is from 2 to 7 or more times faster than Christo-G, with average tour lengths increasing between 1 and 2%, the higher figure being for Clustered instances. For Uniform instances, the limiting percentage excess for AppChristo appears to be 11.1%, compared to 12.1% for Savings, and AppChristo is typically only 1.2 to 3 times slower.

Table 9.9 summarizes the tour quality and running time results on Uniform instances for the best of the “more complex tour construction heuristics” of this section, with Savings included for comparison purposes. Times for similarly sized instances of the other two geometric classes are roughly the same except in the case of Clustered instances. AppChristo is typically almost twice as slow for such instances, while Christo-S and Christo-G are almost 3 times slower for the smaller ones, improving to 25-50% slower when $N = 316, 228$.

3.4. Simple Local Search Heuristics

In this and the next three sections we cover various local search heuristics for the STSP, many of which are described in more detail in Chapter 8. In a local search heuristic for the TSP, one defines a neighborhood structure on the set of tours, where a tour T' is declared to be a *neighbor* of a tour T if it differs from it in some specified way. The classic neighborhoods of this type are the k -Opt neighborhoods, where T' is obtained from T by deleting k edges and replacing them with a different set of k edges (a k -Opt move). For $k > 2$, the sets need not be disjoint, so in particular a k -Opt move is a special case of a $(k + 1)$ -Opt move.

Given a neighborhood structure, a standard local search heuristic operates in two phases. First, it uses some tour construction heuristic to generate a *starting tour*. Then it repeatedly replaces its current tour by a neighboring tour of shorter length until no such tour can be found (either because none exists, in which case the tour is “locally optimal,” or because the heuristic does not explore its neighborhoods exhaustively). A local search heuristic that uses the k -Opt neighborhood is usually called simply “ k -Opt,” and in this section we study various pure and restricted heuristics of this kind.

Currently, 2-Opt and 3-Opt are the main k -Opt heuristics used in practice, introduced respectively by Flood and Croes [314, 228] and by Bock [115]. In Shen Lin’s influential 1965 study of 3-Opt [562], he concluded that the extra time required for 4-Opt was not worth the small improvement in tour quality it yielded, and no results have appeared since then to contradict this conclusion. In contrast, there have been several attempts to trade tour quality for improved running time in 3-Opt by exploiting restricted versions of the 3-Opt neighborhood, as in the *Or-Opt* heuristic of Or [635] and the 2.5-Opt heuristic of Bentley [103].

Implementation Details. Simply stating the neighborhood structure used does not completely specify a local search heuristic. In order to determine the tours generated by the heuristic one needs to provide such additional details as (a) the tour construction heuristic used, (b) the rule for choosing the improving move to make when there are more than one, and (c) the method used to look for improving moves (when the rule specified in (b) depends on the order in which moves are examined). Moreover, the heuristic’s running time will depend on additional implementation details. Although naively one might expect 2-Opt and 3-Opt to require $\Omega(N^2)$ and $\Omega(N^3)$ time respectively, in practice they can be implemented to run much more quickly for geometric instances. 3-Opt can be implemented to run much more quickly than $\Omega(N^3)$ even

for non-geometric instances. Although many factors are involved in these speedups, there are perhaps four key ones.

1. Avoiding Search Space Redundancy
2. Bounded Neighbor Lists
3. Don't-Look Bits
4. Tree-Based Tour Representation

The second and third of these trade a potential slight degradation in tour quality for improvements in running time. In particular, their use leaves open a slight possibility that the final tour may not be 2-Optimal (3-Optimal), i.e., it may have a better neighboring tour that we failed to notice. We now describe each of the four factors individually, as they are relevant not only the simple local search heuristics of this section but to the more sophisticated heuristics of later sections as well.

Avoiding Search Space Redundancy. We illustrate this in the context of 2-Opt. Each possible 2-Opt move can be viewed as corresponding to a 4-tuple of cities $\langle a, b, c, d \rangle$, where $\{a, b\}$ and $\{c, d\}$ are tour edges deleted and $\{a, c\}$ and $\{b, d\}$ are the edges that replace them. Suppose we intend to search through all the possibilities as follows: Let t_1 range over the N possibilities for a ; given t_1 , let t_2 range over the two possibilities for b ; and given t_1 and t_2 , let t_3 range over the possibilities for c (the choice of d is then forced). Note that, as stated, a given move would be examined four times, depending on whether $\langle t_1, t_2 \rangle$ is $\langle a, b \rangle$, $\langle b, a \rangle$, $\langle c, d \rangle$, or $\langle d, c \rangle$. This redundancy can be exploited as follows: Never consider a city t for t_3 unless $d(t_1, t) < d(t_1, t_2)$. Note that if $\langle a, b, c, d \rangle$ is never examined under this regimen, we must have both $d(a, c) \geq d(a, b)$ and $d(b, d) \geq d(c, d)$, and so it cannot be an improving move. Hence no improving move will be missed. This restriction typically strongly limits the possibilities for t_3 as the heuristic proceeds. A generalization to 3-Opt limits choices for both t_3 and the analogous final choice t_5 .

For geometric instances, this restriction can be implemented using K - d trees and a fixed-radius near neighbor search, as described in [103]. For non-geometric instances, one could simply precompute for each city an ordered list of the other cities by increasing distance. However, a quicker option is the following.

Bounded Neighbor Lists. Instead of creating for each city an ordered list of *all* the other cities, create a truncated list of the nearest k cities, ordered by increasing distance, on the assumption that more distant cities are unlikely to yield improving moves. In general, such lists can be computed in overall time $O(N^2 \log k)$. For geometric instances

this can be reduced to something more like $O(N \log N)$ using K - d trees. A possibly more robust version of this approach is to include for each city c the $\lfloor k/4 \rfloor$ cities closest to c in each of the four quadrants of the coordinate system with c at $(0,0)$. If these total fewer than k cities, we augment the set by the nearest remaining cities overall to bring the total up to k . This will be referred to in what follows as a *quad-neighbor* list. Another possibility for geometric instances, suggested by Reinelt [710, 711], is to construct a neighbor list from the cities closest to c in the Delaunay triangulation of the city set.

Don't-Look Bits. This idea was introduced by Bentley in [103] to help avoid the repetition of fruitless searches. Suppose our search for improving moves is as described above, with an outer loop that considers all N possible choices for t_1 . Suppose we are considering the case where $t_1 = a$ and that (i) the last time we searched with $t_1 = a$, we didn't find an improving move and (ii) a has the same tour neighbors as it had that last time. Then it might seem unlikely that we will find an improving move this time either. Bentley proposed not searching in this case, being willing to risk the possibility that occasionally an improving move might be missed. In order to keep track of the cities for which searches could be skipped, he suggested maintaining an array of *Don't-Look* bits. Initially, the bits are all set to 0. Thereafter, the bit for a is set to 1 whenever a search with $t_1 = a$ is unsuccessful. Conversely, if an edge of the tour is deleted when an improving move is made, both its endpoints get their don't-look bits set back to 0. Note that as the local search procedure continues, the number of bits that are set to 0 will decline, so it may make sense to simply keep the cities with 0-bits in a queue, ordered by the length of time since they were last examined, rather than keeping an explicit array of don't-look bits. In this way we not only avoid searches, but spend no time at all considering cities that are to be skipped.

Tree-Based Tour Representation. Empirical measurements reported in [103, 322] suggest that for Uniform instances both 2-Opt and 3-Opt typically make $\Theta(N)$ improving moves. Bentley in [103] observed that as N increases, the time spent performing these moves came to dominate the overall time for his implementations. This was because of the way he represented the tour. A 2-Opt move basically involves cutting the tour in two places and reversing the order of one of the two resulting segments before putting them back together. If the tour is stored in a straightforward way (either as an array or a doubly linked list), this means that the time for performing the 2-Opt move must be at least proportional to the length of the shorter segment. Bentley's empirical data suggested that for Uniform instances the average length of this shorter segment was growing roughly as $N^{0.7}$, as was the average

work for performing each move. If we consider alternative tree representations, this can be reduced to \sqrt{N} using the 2-level trees of [322] or to $\log N$ using the splay tree data structure of [764]. For a study of the tradeoffs involved and the crossover points between various tour representations, see [322].

Results for 2-Opt, 2.5-Opt, and 3-Opt. Implementation choices can make a difference, both in tour quality and running time. We consider three sets of implementations.

- 2-Opt and 3-Opt implementations by Johnson and McGeoch (-JM).
- 2-Opt, 3-Opt and “2.5-Opt” implementations by Bentley (-B). The third heuristic is a restricted version of 3-Opt in which the 2-Opt neighborhood is augmented only by those 3-Opt moves that delete a single city from the tour and reinserted it elsewhere.
- 2-, 2.5-, and 3-Opt implementations by Applegate, Bixby, Chvátal, and Cook (-ABCC). These are included as options in the `edgegen` program of the `Concorde` software release.

The three sets of implementations are similar in that all exploit don't-look bits, but differ in many other respects. `Concorde`'s implementations use Nearest Neighbor to generate starting tours, whereas the Bentley and Johnson-McGeoch implementations both use the Greedy heuristic (although Johnson and McGeoch use a randomized variant that picks the shortest edge with probability 2/3 and the second shortest with probability 1/3). For all three heuristics `Concorde` considers only one of the two neighbors of t_1 as a choice for t_2 whereas the Bentley and Johnson-McGeoch implementations consider both. Another difference has to do with move selection. For each choice of t_1 , the Johnson-McGeoch and `Concorde` implementations apply the first improving move found (except that in the JM implementations an improving 2-Opt move is not performed in 3-Opt unless no way is found to extend it to an even better 3-Opt move). In contrast, for each choice of t_1 , the Bentley implementations keep looking for improving moves until it has seen 8 (or run out of possibilities) and then performs the best of these. The Bentley and `Concorde` implementations also have more chance of finding improving moves, since they use fixed-radius near-neighbor searches to find all possible candidates for t_3 (and t_5 in the case of 3-Opt), whereas the JM implementations restrict the choices to quad-neighbor lists of length 20. On the other hand, in the case of 3-Opt, Bentley's implementation examines fewer classes of potential 3-Opt moves, omitting for example those 3-Opt moves that permute but do not reverse any of the three

Algorithm	Percent Excess			Time (Seconds)		
	U	C	T	U	C	T
Christo-G	9.9	9.6	9.5	21.3	37.8	29.5
Christo-HK	6.9	8.4	7.4	247.2	197.0	177.9
2opt-B	5.7	9.6	5.8	8.8	9.7	5.6
2opt-JM	4.8	10.7	6.0	10.7	12.4	6.5
2opt-ABCC	14.4	19.6	14.7	3.7	2.9	1.9
2.5opt-B	4.7	8.2	4.8	10.2	12.0	7.5
2.5opt-ABCC	12.6	17.3	13.0	4.3	3.2	2.4
3opt-B	3.6	5.5	3.8	15.5	176.8	17.8
3opt-JM	3.0	6.9	4.2	12.3	14.9	6.9
3opt-ABCC	8.4	11.2	9.2	6.1	12.5	3.7

Table 9.10. Average percent excess over the HK bound and normalized running times for 100,000-city Uniform and Clustered instances and for TSPLIB instance p1a85900. All codes were run on the same machine.

segments created when 3 tour edges are deleted. A final difference is that the Bentley and Concorde implementations represent the tour with an array whereas Johnson and McGeoch use the 2-level tree of [322].

Table 9.10 summarizes average heuristic performance of these implementations on the instances of approximately 100,000 cities in our three geometric classes. A first observation is that the Concorde (-ABCC) implementations are much faster and produce much worse tours than their -B and -JM counterparts. This is a reasonable tradeoff in the context of the intended use for the Concorde implementations, which is to quickly generate sets of good edges for use with other Concorde components. It does, however, illustrate the danger of Concorde’s restriction to one choice for city t_2 , which is the primary cause for this tradeoff: Applying the same restriction to the JM implementations yields similar improvements in running time and degradations in tour quality.

As to comparisons between the Bentley and JM implementations, the latter produce better results for Uniform instances but are worse for Clustered instances. For these, Bentley’s more complete examination of candidates for t_3 may be paying off, although there is a substantial running time penalty in the case of 3-Opt. For TSPLIB instances, the results are mixed, with the JM implementations more often producing better tours, although not for p1a85900 as shown in the table. Also not evident in the table is the running time penalty that the Bentley and Concorde implementations experience once $N > 100,000$, due to their use of the array representation for tours. Their observed running times have $\Omega(N^{1.5})$ growth rates, whereas the JM implementations, with their tree-based tour representations, have observed running times that appear to be $O(N \log^2 N)$. As a consequence, 2.5opt-B is 5 times slower than the JM implementation of full 3-Opt when $N = 3,162,278$ and

Algorithm	Percent Excess			Time (Seconds)		
	1,000	3,162	10,000	1,000	3,162	10,000
NN-ABCC	224	321	337	0.8	9.7	112
Greedy-JM	163	198	250	0.7	8.8	107
2opt-JM	66	92	114	1.0	12.2	157
3opt-JM	31	43	63	1.1	12.3	150

Table 9.11. Average percent excesses over the HK bound and normalized running times for Random Matrix instances of sizes from 1,000 to 10,000 cities. All codes were run on the same machine.

2opt-ABCC and 3opt-ABCC are both 5-10 times slower than the corresponding JM implementations once $N = 1,000,000$.

Table 9.10 also addresses the question of how these simple local search heuristics compare to the best of the tour construction heuristics in our study: greedy-shortcut Christofides (Christo-G) and its Held-Karp-based variant Christo-HK. Based on the results in the table, it would appear that both are dominated by the 2.5opt-B and 3opt-JM, and the first is also dominated by 2opt-B. The situation is a bit more complicated, however, if one looks at the Challenge testbeds as a whole. Christo-G tends to produce better tours than 2opt-B for many Clustered instances and to be faster than all the Bentley and JM implementations for Uniform and TSPLIB instances with 10,000 or fewer cities. 2.5opt-B and 3opt-JM, however, produce better tours than Christo-G and Christo-HK for almost all the instances in the Challenge testbeds on which the latter two could be run. (3opt-JM loses only on two instances.) Since the running time advantage for Christo-G is never more than a factor of 3 on the smaller instances, all of which can be handled by the 3opt-JM in normalized time of less than 10 seconds (usually less than 2), and since the running time for Christo-HK is substantially worse than that for 3opt-JM across the board, there is probably no real reason to use either Christofides variant in practice, assuming one has a good implementation of 3-Opt.

The JM implementations of 2-Opt and 3-Opt can handle non-geometric instances, and so we also ran them on our Random Matrix testbed. The results are summarized in Table 9.11, which for comparison purposes also includes results for two tour construction implementations that can handle such instances: the benchmark Greedy code, which provides a good estimate for the lengths of the starting tours used by 2opt-JM and 3opt-JM, and Concorde's implementation of NN.

Observe that all the heuristics produce much poorer tours for Random Matrix instances than they do for geometric instances. Even the best of them, 3-Opt, has percentage excesses that are worse by a factor of 10 or more. Also note that tour quality declines substantially as N

increases. The results for all four heuristics are consistent with the conjecture that the average percentage excess grows as $\log N$, whereas for the geometric instances in our testbeds, all of our heuristics seem to have average percentage excesses that are bounded, independent of N . The running time for Random Matrices is also much worse, growing somewhat more rapidly than N^2 , the nominal growth rate for simply reading the instance. An interesting side effect of this is that the time for performing local search becomes a much less significant component of the overall running time. `3opt-JM` takes only about 50% more time than `NN-ABCC` and the running time difference between `2opt-JM` and `3opt-JM` is inconsequential. (This is because most of the local search speedup tricks mentioned above continue to be applicable, so that the local search phase does not take much longer for Random Matrices than it did for geometric instances.) Given how much better `3opt-JM`'s tours are than those of the other heuristics, it is the obvious choice among the four, should one want to solve this kind of instance.

More on Starting Tours and Neighbor Lists. As noted above, all the Bentley and the JM implementations used the Greedy heuristic to generate starting tours. This decision was based on the extensive experiments reported in [103, 461], which showed that **Greedy** tours tended to yield the best results, both in comparison to worse tour construction heuristics such as **NN** or the infamous “generate a random tour” heuristic, and to better ones such as **FI** or **Savings**. It appears that the starting tour needs to have some obvious defects if a simple local search heuristic is to find a way to make major improvements, but it can't be too bad or else the heuristic will not be able to make up the full difference. Random starting tours have the additional disadvantage that they lead to increased running times because more moves need to be made to reach local optimality.

The JM implementations for which results were reported above used neighbor lists of length 20. Many authors have suggested using substantially shorter lists, but at least in the context of these implementations, 20 seems a reasonable compromise. Using lists of length 10 saves only 10-20% in running time but on average causes tour lengths to increase by 1% or more for both `2opt-JM` and `3opt-JM` and all four instance classes. Increasing the list length to 40 increases running time by 40-50% and for `3opt-JM` on average improves tour length by less than 0.3% on all but the Clustered instances. The average improvements for Clustered instances are more variable, but appear to average roughly 0.6% overall. For `2opt-JM` the tour length improvements due to increasing the neighbor list length to 40 are slightly larger, but the running time becomes

greater than that for 3opt-JM with 20 neighbors, and the latter finds much better tours. For full details, see the Challenge website.

Other Simple Local Search Heuristics. 2.5-Opt is not the only restricted version of a k -Opt heuristic that has been seriously studied. Much early attention was devoted to the Or-Opt heuristic of [635], which uses a neighborhood intermediate between that of 2.5-Opt and full 3-Opt. In 2.5-Opt, we consider those 3-Opt moves in which one of the three segments into which the tour is initially broken contains just one city; Or-Opt expands this to segments of as many as 3 cities. It was originally proposed as a way of reducing the running time overhead of the naive $\Omega(N^3)$ implementation of 3-Opt, before the existence of the speedup tricks mentioned above was widely known. Although these tricks should be adaptable to Or-Opt as well, it seems unlikely that the latter would retain much speed advantage over a more complete 3-Opt implementation. Thus the probable tour degradation due to the much smaller Or-Opt neighborhood is not likely to be justified, and Or-Opt no longer appears to be a serious competitor. No implementations were submitted to the Challenge.

Researchers have recently also considered putting restrictions on the k -Opt neighborhood when $k \geq 4$, with the intent of getting better tours than 3-Opt without paying the full running time penalty for k -Opt itself. Two families of heuristics of this type were submitted to the Challenge: the GENI/GENIUS heuristics of Gendreau, Hertz, and Laporte [351] and the HyperOpt heuristics of Burke, Cowling, and Keuthen [149].

From one point of view GENI can be viewed as a tour construction heuristic, in that the tour is augmented one city at a time. However, each augmentation is equivalent to a simple insertion followed by a 4- or 5-Opt move, so we have chosen to consider it here in the section on local search. GENIUS uses GENI to construct its starting tour, and then attempts to improve it by a “stringing-unstringing” procedure that technically is a restricted version of 8-, 9-, or 10-Opt. In addition, the heuristics use truncated nearest neighbor lists to restrict their choices, with the heuristic being parameterized by the length p of these lists.

We tested implementations of GENI and GENIUS provided to us by Gendreau et al., which we fine-tuned by improving their handling of memory allocation and by removing some redundant operations. This fine-tuning did not change the output tours but does result in substantial running-time improvements. The implementations still do not, however, exploit the full set of speedup tricks listed above, and with $p = 10$ GENI is 100 times slower for 10,000-city instances than 3opt-JM (on the same machine), and GENIUS is over 300 times slower. Moreover, although both GENI and GENIUS find better tours than 3-Opt for Clustered instances,

they are worse for Uniform instances and most TSPLIB instances. Increasing p to 20 does not yield a significant improvement in the tours but causes substantial increases in running time. It is possible that a reimplementing of the heuristics to take advantage of more speedup tricks might make them more competitive timewise. It seems unlikely, however, that we could speed up the heuristics sufficiently to make them competitive with implementations of Lin-Kernighan to be described in the next section, and those find better tours across the board. (Based on comparisons to GENIUS in [713], the I^3 heuristic of Renaud, Boctor, and Laporte [713], which uses a restricted 4-Opt neighborhood, would not seem to be competitive either.)

The HyperOpt heuristics of [149] also come in parameterized form, with the neighborhood structure for k -HyperOpt being a restricted version of the $2k$ -Opt neighborhood. To construct a neighboring tour, one first deletes two disjoint sets of k consecutive tour edges. This breaks the tour into two subtours and $2(k - 1)$ isolated cities. These are then recombined optimally into a tour using dynamic programming. The implementations of Burke et al. are substantially faster than those for GENI/GENIUS, even taking into account possible normalization errors. The normalized times for 2-HyperOpt are comparable to those for the 3opt-JM until they start to degrade around $N = 100,000$. Unfortunately, 2-HyperOpt's average tour quality is worse than that of 3opt-JM for all three geometric instance classes. 4-HyperOpt might be slightly better than 3opt-JM on Clustered instances, but it is a close call, and 3opt-JM is 50 times faster.

Thus as of now it does not appear that restricted k -Opt heuristics, $k \geq 4$, offer a promising avenue to cost-effective improvements on 3-Opt. As was first shown in 1973 by Lin and Kernighan [563], a much better generalization is the concept of *variable-depth* search. We cover heuristics based on this concept in the next section.

3.5. Lin-Kernighan and Variants

The Lin-Kernighan heuristic of [563] does not limit its search to moves that change only a bounded number of edges. In principle it can change almost all the edges in the tour in a single move. However, the moves have a specific structure: each can be viewed as a 3-Opt move followed by a sequence of 2-opt moves, although only the full final move need actually improve the tour. The heuristic keeps running time under control by restricting the "LK-Search" to moves that are grown one 2-Opt move at a time, without backtracking beyond a fixed level. In addition, it uses neighbor lists to restrict the number of growth alternatives it considers,

locks edges that have already been changed by the move so that the changes won't subsequently be undone, and aborts the search if a strict gain criterion based on the best tour seen so far is not met. More details can be found in Chapter 8.

In implementing Lin-Kernighan, one has far more choices to make than for simple heuristics like 2-Opt and 3-Opt, and the literature contains reports on many implementations of Lin-Kernighan with widely varying behavior. In addition to such standard choices as (a) what tour construction heuristic to use, (b) when to continue looking for better improving moves after one has been found, (c) how long neighbor lists should be and how they should be constituted, (d) what tour representation should be used, and (e) whether to use don't look-bits, here are some of the new choices:

- How broad should the search be? The original Lin-Kernighan implementation restricted attention to neighbor lists of length 5 at all levels of the search.
- How much backtracking is allowed? The original Lin-Kernighan implementation considered all 3-Opt moves that met the gain criterion (together with a restricted class of 4-Opt moves) as starting points for the LK-search.
- Should one consider just one or the standard two choices for t_2 ?
- Does one lock both deleted and added edges, or only one of the two classes? (The original LK implementation locked both, but locking either of the two classes separately is enough to insure that the search runs in polynomial time.)
- Does one tighten the gain criterion in the middle of the LK-search if a better tour is discovered along the way?
- Should one extend the partial move that yields the best tour, or the one that has the best gain criterion?
- Should one impose a constant bound on the depth of the LK-search?
- Should one augment the moves constructed by LK-search with non-sequential "double-bridge" 4-Opt moves, as proposed by Lin and Kernighan (but not used in their own implementation)?

Moreover, one can consider topological variants on the way that moves are grown and the reference structure maintained at each level of the search. Lin and Kernighan's reference structure is a path with one end

fixed, and the changes considered in growing the move all consist of adding an edge from the unfixed end of the path to a city on its neighbor list, and then deleting the one edge that will take us back to a path. Other possible reference structures have been implemented, such as the unanchored path of Mak and Morton [576] or the “stem-and-cycle” of Glover [373, 374]. In addition, alternative ways of extending an LK-search (besides the standard 2-Opt move) have been considered, as in the variant due to Helsgaun [446] that augments via 5-Opt moves.

We do not have space to go into full detail on how all the tested implementations differ. Indeed, many implementers’ written descriptions of their implementations do not provide all the answers. So we concentrate in what follows on key differences (and similarities) and our conclusions about the effects of various choices are necessarily tentative.

Basic Lin-Kernighan. We start with implementations that do not depart in major ways from the original heuristic, i.e., implementations that use a path as reference structure, use 2-Opt moves as the augmentation method in LK-search, and do not perform double-bridge moves. Four such implementations were submitted to the Challenge:

1. LK-JM (Johnson and McGeoch [461, 463]). The main results reported here for this implementation use **Greedy** starting tours, length-20 quad-neighbor lists for all levels of the search, don’t-look bits, and the 2-Level Tree tour representation [322]. In the LK-search, this C implementation uses an anchored path as its reference structure, locks only the added edges, updates the gain criterion when a better tour is found in mid-search, does not bound the depth of the searches, and otherwise follows the original Lin-Kernighan **Fortran** code, from which it is derived.
2. LK-Neto (Neto [626]). This implementation is based on the original Lin-Kernighan paper [563] and on the Johnson-McGeoch chapter [463]. It differs from the Johnson-McGeoch implementation in that neighbor lists consist of 20 quadrant neighbors *unioned* with the 20 nearest neighbors, LK-searches are bounded at 50 moves, and special *cluster compensation* routines are used with the hopes of improving performance for instances in which the cities are grouped in widely separated clusters, presumably as in our Clustered instances. Source code for this implementation (in CWEB) is available from <http://www.cs.toronto.edu/~neto/research/lk/>.
3. LK-ABCC (Applegate, Bixby, Chvátal, and Cook [27]). This is the default Lin-Kernighan implementation in **Concorde**. Based on remarks in [27], it would appear that this implementation differs

from the Johnson-McGeoch implementation mainly as follows: It uses Q-Boruvka starting tours, length-12 quad-neighbor lists, an unanchored path as reference structure, a narrower (but slightly deeper) backtracking strategy with just one choice for t_2 , and LK-searches bounded at 50 moves. Source code is available from the *Concorde* website.

4. LK-ACR (Applegate, Cook, and Rohe [32]). Based on remarks in [27, 32], it would appear that this implementation differs from *Concorde*'s in that it uses a slightly broader and deeper backtracking strategy and bounds the depth of the LK-search at 25 moves.

Both of the latter two implementations were optimized for use in the context of Chained Lin-Kernighan, where the speed of a single invocation of LK may be more important than saving the last few fractions of a percent in tour length. Results confirm this. See Table 9.12 which presents average tour qualities and running times for the four implementations. For comparison purposes, the corresponding results for 3opt-JM are also included.

Note that LK-JM and LK-Neto provide roughly equivalent tours except for the Clustered instances. These tours are typically substantially better than those for LK-ABCC and LK-ACR, although LK-ABCC and LK-ACR are significantly faster for Clustered and TSPLIB instances and for Uniform instances when $N \leq 10,000$. As in the case of the ABCC implementations of 2-, 2.5-, and 3-Opt, a major reason for this disparity is probably the restriction in LK-ABCC and LK-ACR to a single choice for t_2 . A second but lesser reason is the fact that the latter two heuristics construct shorter neighbor lists and do so more quickly. The fact that they bound the depth of the LK-search would not seem to be a major factor, however, since LK-Neto also bounds the depth. Moreover, if one imposes a depth bound of 50 on LK-JM, neither tour quality nor running time is typically affected significantly. As to asymptotic growth rates, the observed running times for LK-JM and LK-ACR appear to be $O(N^{1.25})$, while those for LK-Neto and LK-ABCC may be somewhat worse.

Between LK-ABCC and LK-ACR, the former tends to yield slightly better tours and take slightly longer, but the results for both are closer to those for 3opt-JM than to those for LK-JM. Indeed, for the three smallest TSPLIB sizes, 3opt-JM on average finds better tours than LK-ACR. This becomes less surprising when we look at running times, since LK-ACR actually appears (subject to normalizing errors) to use less time than 3opt-JM for these instances. LK-ABCC is almost as fast.

Average Percent Excess over the HK Bound

N =		1000	3162	10K	31K	100K	316K	1M	3M	10M
Random Uniform Euclidean Instances										
LK:	JM	1.92	1.99	2.02	2.02	1.97	1.96	1.96	1.92	–
	Neto	1.91	1.97	1.99	1.89	1.95	1.97	1.92	1.88	–
	ABCC	2.22	2.43	2.60	2.48	2.54	2.67	2.68	2.55	2.54
	ACR	2.36	2.90	2.72	2.73	2.74	2.75	2.77	2.67	2.49
3opt:	JM	2.96	2.84	3.06	3.02	2.97	2.93	2.96	2.88	–
Random Clustered Euclidean Instances										
LK:	JM	1.75	2.95	3.41	3.71	3.63	3.67			
	Neto	2.52	4.19	4.76	4.42	4.78	–			
	ABCC	3.77	6.23	5.70	6.38	5.31	5.45			
	ACR	3.89	6.13	5.93	6.28	5.54	5.54			
3opt:	JM	4.08	6.06	6.89	7.48	6.88	7.08			
TSPLIB Instances										
LK:	JM	2.38	2.16	1.92	1.73	1.61				
	Neto	2.40	2.32	1.88	2.00	–				
	ABCC	3.54	3.29	2.39	2.16	1.60				
	ACR	4.48	3.48	3.72	2.91	2.40				
3opt:	JM	3.93	3.67	3.17	3.99	4.20				

Average Normalized Running Time in Seconds

Random Uniform Euclidean Instances										
LK:	JM	0.20	0.69	2.32	7.2	22.8	61	323	1255	–
	Neto	0.19	0.87	3.35	14.4	89.6	574	3578	17660	–
	ABCC	0.09	0.34	1.49	6.0	21.4	61	307	1330	6980
	ACR	0.07	0.29	0.93	3.0	16.4	76	318	1290	5760
3opt:	JM	0.13	0.45	1.44	4.2	12.3	33	162	600	–
Random Clustered Euclidean Instances										
LK:	JM	1.66	4.97	15.37	59.3	173.1	495			
	Neto	4.35	15.04	51.17	138.6	558.1	–			
	ABCC	0.19	0.72	2.55	11.0	37.9	108			
	ACR	0.10	0.45	1.40	4.5	25.0	114			
3opt:	JM	0.15	0.54	1.77	5.0	14.9	38			
TSPLIB Instances										
LK:	JM	0.34	0.64	4.29	13.0	24.3				
	Neto	0.40	1.08	10.26	47.1	–				
	ABCC	0.10	0.29	1.21	3.5	8.8				
	ACR	0.08	0.23	0.74	1.7	5.4				
3opt:	JM	0.14	0.38	1.42	3.4	6.9				

Table 9.12. Results for 3-Opt and four implementations of Lin-Kernighan. Averages for TSPLIB are taken over the same instances as in Figure 9.3.

Turning now to the relationship between LK-JM and LK-Neto, a first observation is that the algorithmic descriptions in [563, 463] (on which the LK-Neto implementation was based) seem to be adequate to define a reproducible Lin-Kernighan implementation, at least as far as tour quality on Uniform and most TSPLIB instances is concerned. The two implementations do differ significantly on Clustered instances, but this is presumably because Neto has added innovations designed to handle clustered instances more effectively. Unfortunately, for this particular class of Clustered instances, the innovations do not appear to be effective. LK-Neto provides significantly worse tours than LK-JM and the ratio of its normalized running time to that of LK-JM is higher for Clustered instances than it is for Uniform instances. There also appears to be some as-yet-unidentified difference in the two implementations that has an asymptotic effect on running times: For Uniform instances the normalized running times for LK-Neto start to diverge from those for LK-JM when N gets large.

All the above LK implementations use variants on the Greedy heuristic to generate starting tours and so do not provide much of an opportunity to analyze the effects of different starting heuristics on the final LK results. (Analyses of this sort can be found in [32, 461].) However, one question is hard to resist: What would happen if we combined Lin-Kernighan with the very best (and slowest) of our tour generation heuristics, *Christo*-HK? Andre Rohe submitted results for just such a combination, and although his Lin-Kernighan implementation was an early one that lacks the detailed tuning of LK-ABCC and LK-ACR, the results are intriguing. For $N > 1,000$ the total running time is dominated by that simply to generate the starting tour and for Uniform and TSPLIB instances averages from 3 to 10 times slower than that for LK-JM. However, the final tours are significantly better. For Uniform instances the limiting ratio to the HK bound appears to be about 1.5% versus the 1.9% for LK-JM and the average improvement over LK-JM on TSPLIB instances ranges from 0.2% to 0.5%. For the larger Clustered instances the improvement over LK-JM is roughly 1.3% and there is essentially *no* running time penalty. Although even bigger improvements are possible using the repeated local search heuristics of the next section, those heuristics typically take even more time. Thus combining Lin-Kernighan with *Christo*-HK could sometimes be an appealing option.

To complete the Lin-Kernighan picture, Table 9.13 covers Random Matrix instances. Since the results for LK-Neto do not cover the 10,000-city instance, we also include a version of LK-JM with LK-search depth bounded at 50 (LK-JM-BD). (Note that because the triangle inequality does not hold, *Christo*-HK starting tours are no longer relevant.)

Algorithm	Percent Excess			Time (Seconds)		
	1,000	3,162	10,000	1,000	3,162	10,000
LK: JM	3.5	4.4	5.9	1.2	12.8	161
JM-BD	3.5	4.8	6.2	1.1	12.4	154
Neto	3.0	4.1	–	1.2	16.9	–
ABCC	4.0	6.0	9.0	1.1	12.6	151
ACR	5.3	7.6	10.3	0.3	3.6	32
3opt: JM	31.2	42.6	62.7	1.1	12.3	150

Table 9.13. Average percent excesses over the HK bound and normalized running times for Random Matrix instances of sizes from 1,000 to 10,000 cities.

Once again LK-ACR is faster than 3opt-JM (in this case significantly so), but now it also finds much better tours. Its tours are however worse than those for LK-ABCC, and both produce significantly worse tours than LK-JM and LK-Neto. Once again it is not clear if any of this difference can be attributed to bounding the depth of the LK-search. Indeed, LK-Neto, with bounding, finds significantly better tours than the LK-JM, with or without bounding (for reasons unknown to us and in contrast to the results for the geometric case). Running times for all the implementations except LK-ACR are comparable, being dominated by the time to read and preprocess the instance. As to the difficulty of this instance class in general, note that for all the implementations, the percentage above the Held-Karp bound again appears to be growing with N (possibly at a $\log N$ rate), something that doesn't happen for the geometric classes.

We now turn to variants on basic Lin-Kernighan that involve more substantial changes to the basic design of the heuristic.

Stem-and-Cycle Variants. These variants differ primarily in their choice of reference structure for the LK-search. Here the structure consists of a cycle with a path connected to one of its vertices, as described in more detail in Chapter 8. The implementation for which we report results is due to Rego, Glover, and Gamboa and is based on the heuristic described in [704]. It makes use of the 2-Level Tree tour representation, but does not use don't-look bits. (As we shall see, this probably exacts a severe running time penalty.) Variants using random starting tours (SCLK-R) or Boruvka starting tours (SCLK-B) yield roughly comparable tours, although the latter is usually at least twice as fast.

Figure 9.10 shows the relative tour-quality performance on our geometric testbed for SCLK-B and LK-JM. Note that the two are consistently close, usually well within 1% of each other. Neither implementation has tours that are consistently better than the other's, although LK-JM wins more often than it loses. The normalized running time for SCLK-B ranges from 4 times slower than LK-JM on 1,000-city Clustered instances

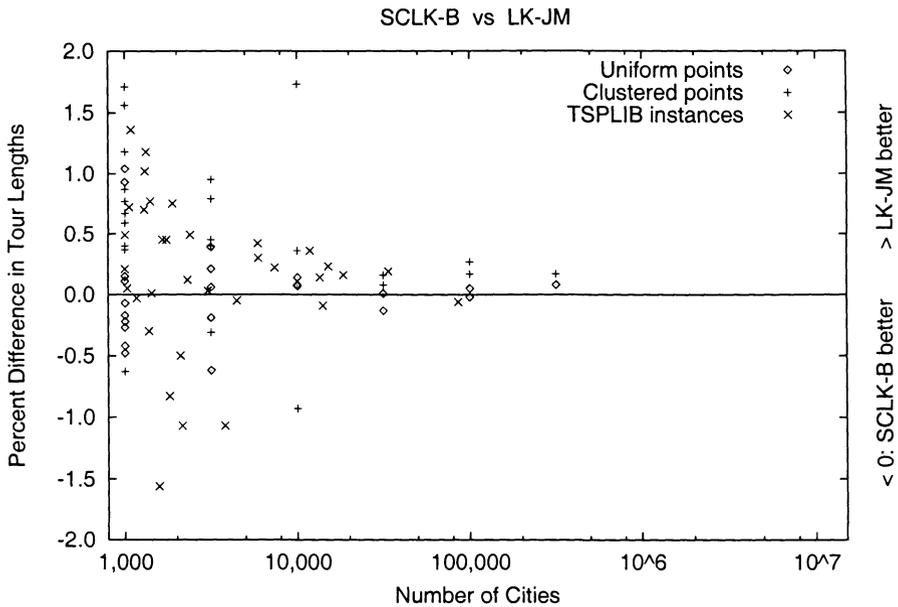


Figure 9.10. Tour quality comparisons between the Rego-Glover-Gamboia implementation of the Stem-and-Cycle variant of Lin-Kernighan with Boruvka starting tours and the Johnson-McGeoch implementation of basic Lin-Kernighan.

to 1300 times slower on the 316,228-city Uniform instance (the largest on which SCLK-B was run). It is possible that incorporating don't-look bits into the implementation will make up much of this gap, but for now there does not seem to be any significant advantage to this approach over basic Lin-Kernighan.

The Helsgaun Variant (Helsgaun). This variant, described in [446], offers several major innovations. First, the augmentation step in the LK-search is not a 2-Opt move but a sequential 5-Opt move. To keep running time under control, search at all levels is limited to length-5 neighbor lists. These are constructed in an innovative fashion.

We begin as if we were going to compute an estimate of the Held-Karp bound using the Lagrangean relaxation approach of [444, 445] augmented with techniques from [230, 442]. This yields a vector of “ π -values” (π_1, \dots, π_N) such that the minimum one-tree (spanning tree plus an edge) under the distance function $d_\pi(c_i, c_j) = d(c_i, c_j) + \pi_i + \pi_j$ is a close lower bound on the Held-Karp bound. Based on this new distance function, the “ α -value” $\alpha(i, j)$ for each edge $\{c_i, c_j\}$ is defined to be the difference between the length of the minimum one-tree (under d_π) that is required to contain $\{c_i, c_j\}$ and the length of the minimum unconstrained one tree (under d_π). Given the vector of π -values, the

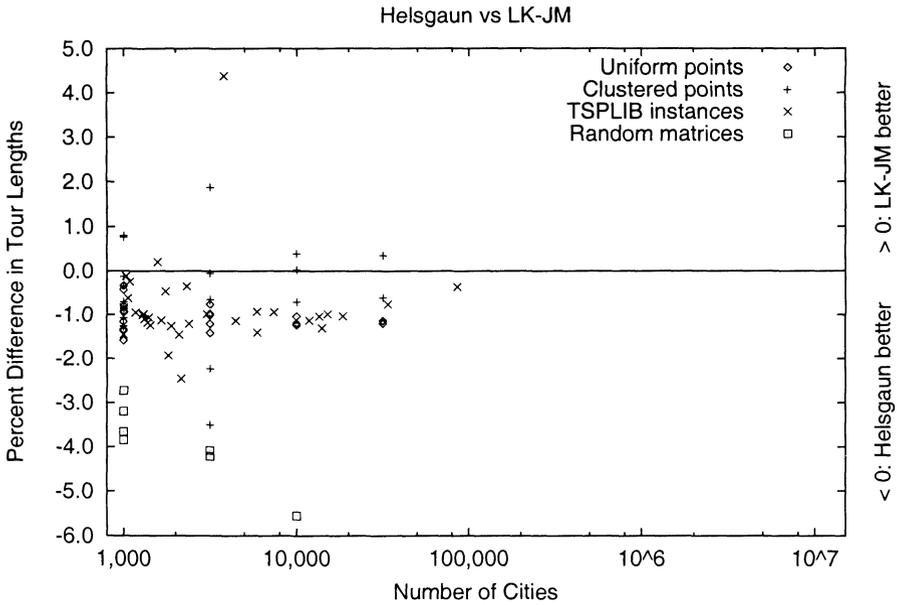


Figure 9.11. Tour quality comparisons between the Helsgaun variant on Lin-Kernighan and the Johnson-McGeoch implementation of basic Lin-Kernighan.

α -values for all edges can be computed in linear space and $O(N^2)$ time [446]. The neighbor list for c_i then consists of the 5 cities c_j with smallest values of $\alpha(i, j)$. These are initially ordered by increasing α -value, but are subsequently dynamically reordered to give priority to edges shared by the current best tour and its predecessor.

In addition, the implementation alternates between searching its “LK-search-with-sequential-5-Opt-augmentations” neighborhood and searching a second neighborhood defined by a set of non-sequential moves that includes not only double-bridge 4-Opt moves but also some specially structured 5-Opt moves as well. Given the power of its search strategy, the implementation needs only to backtrack at the first (t_1) level. For its starting tours it uses a special heuristic that exploits the α -values. The reader interested in this and the other details of the heuristic is referred to Helsgaun’s paper [446]. The C source code for **Helsgaun** is currently available from <http://www.dat.ruc.dk/~keld/>.

The tour-quality results for this variant are impressive. See Figure 9.11, which compares **Helsgaun** to LK-JM. Note that **Helsgaun** finds better tours for a large majority of the instances in our testbed, usually better by 1% or more for Uniform and TSPLIB instances, a large amount in the context of the differences between LK-variants. The difference is even greater for our Random Matrix instances. The detailed

Average Percent Excess over the HK Bound

N =		1000	3162	10K	31K	100K	316K	1M	3M	10M
U	LK-JM	1.92	1.99	2.02	2.02	1.97	1.96	1.96	1.92	-
	Helsgaun	0.90	0.89	0.83	0.83	-	-	-	-	-
C	LK-JM	1.75	2.95	3.41	3.71	3.63	3.67			
	Helsgaun	1.25	2.00	3.32	3.58	-	-			
T	LK-JM	2.38	2.16	1.92	1.73	1.61				
	Helsgaun	1.20	1.01	0.77	0.96	1.25				
M	LK-JM	3.52	4.35	5.91						
	Helsgaun	0.04	0.02	0.03						

Average Normalized Running Time in Seconds

U	LK-JM	0.20	0.7	2	7	23	61	323	1255	-
	Helsgaun	5.64	71.5	862	7820	-	-	-	-	-
C	LK-JM	1.66	5.0	15	59	173	495			
	Helsgaun	7.02	70.3	768	12800	-	-			
T	LK-JM	0.34	0.6	4	13	24				
	Helsgaun	7.82	73.3	1060	7980	48200				
M	LK-JM	1.15	12.8	161						
	Helsgaun	7.78	100.6	1270						

Table 9.14. Results for the Helsgaun variant on Lin-Kernighan as compared to those for the Johnson-McGeoch implementation of basic Lin-Kernighan.

averages are shown in Table 9.14, which indicates that for Random Matrices, Helsgaun finds tours that are essentially optimal. The table also reports on average running times, which show that Helsgaun pays a significant running time penalty for its improved tour quality. In some cases Helsgaun takes as much as 1,000 times longer than LK-JM, and its running time growth rate appears to be $\Omega(N^2)$ versus the observed $O(N^{1.25})$ rate for LK-JM. In certain applications such a price may be worth paying of course. The most active and interesting research on STSP heuristics today concerns how best to use a large amount of computation time to help narrow the small gap above optimal tour length left by Lin-Kernighan. In the next section we consider other ways of buying better tours by spending more time.

3.6. Repeated Local Search Heuristics

One way to invest extra computation time is to exploit the fact that many local improvement heuristics have random components, even if only in their initial tour construction. Thus if one runs the heuristic multiple times one will get different results and can take the best. Unfortunately, as noted by many authors and aptly illustrated in [463],

the value of the straightforward repeated-run approach diminishes as instance size increases.

A much more effective way to use repeated runs has received wide attention in the last decade. The idea is basically this: Instead of using independently generated starting tours, derive the starting tours by perturbations of the output tours of previous runs. This idea was suggested by Baum in 1986 [93] and is in a sense implicit in the operation of the Tabu Search approach of Glover [369, 370]. However, its first effective realization in the context of the TSP is due to Martin, Otto, and Felten, who in [587, 588] proposed generating a new starting tour by perturbing the current best tour with a “kick” consisting of a random double-bridge 4-Opt move. Their original implementation used 3-Opt as the local search engine, but this was quickly upgraded to Lin-Kernighan in their own work and that of Johnson [460].

If the underlying Lin-Kernighan (or 3-Opt) variant uses don’t-look bits, this approach has an additional advantage. At the end of a run of Lin-Kernighan, all the don’t-look bits will typically be on since no improving moves have been found for any choice of t_1 . Performing one 4-Opt move to get a new starting tour only changes the tour neighbors for at most 8 cities. This suggests that, instead of starting with all the don’t-look bits off, as in a stand-alone run, we might want to start with the don’t-look bits on for all but these 8 or fewer cities, i.e., we place only these cities in the initial priority queue of candidates for t_1 . In practice, this can lead to sublinear time per iteration if other data structures are handled appropriately, which more than makes up for any loss in the effectiveness of individual iterations.

Martin, Otto, and Felten referred to their approach as “chained local optimization” since one could view the underlying process as a Markov chain. They also attempted to bias their choice of 4-Opt move toward better than average moves, and incorporated a fixed-temperature simulated annealing component into the heuristic. Results for five descendants of this approach were submitted to the Challenge, differing both in the underlying local search heuristic used and in the methods used for generating the double-bridge kicks. (Variants using more elaborate kicks have been studied [205, 452], but none have yet proved competitive on large instances with the best of the heuristics presented here.) Together, these five include all the current top performers known to the authors for this region of the time/quality tradeoff space. None of the implementations use simulated annealing, and the first three call themselves “Iterated” 3-Opt/Lin-Kernighan to mark this fact.

1. **Iterated 3-Opt (Johnson-McGeoch) (I3opt)** [463]. This uses 3opt-JM and random double-bridge kicks.

2. **Iterated Lin-Kernighan (Johnson-McGeoch) (ILK-JM) [463].** This uses LK-JM-BD and random double-bridge kicks.
3. **Iterated Lin-Kernighan (Neto) (ILK-Neto) [626].** This uses LK-Neto with random double-bridge kicks.
4. **Chained Lin-Kernighan (Applegate-Bixby-Chvátal-Cook) (CLK-ABCC).** This uses LK-ABCC and the sophisticated method for generating promising kicks described in [27].
5. **Chained Lin-Kernighan (Applegate-Cook-Rohe) (CLK-ACR) [32].** This uses LK-ACR together with a new method of generated biased kicks based on random walks, with the number of steps in the random walk doubling after the N th iteration.

Given the huge running times for Helsgaun's variant on Lin-Kernighan, the above variants on Chained Lin-Kernighan can all perform many iterations and still not take as much time as Helsgaun does. However, if one has time to spare, one can also perform the latter repeatedly. For this, Helsgaun [446] has devised a different way of exploiting information about previous runs. Instead of using a kick to perturb his current champion into a new starting tour, he uses his standard tour construction heuristic to generate the new starting tour, but biases its choices based on the edges present in the current champion tour. He gets a per-iteration speedup because he doesn't have to recompute the π -vector and the α -values after the first iteration, although his running times remain substantial. We shall refer to this heuristic as *Helsgaun- k* , where k is the number of iterations.

Tables 9.15 and 9.16 summarize results for the various repeated-run heuristics described above. Results for ILK-Neto are omitted because of its similarity to the ILK-JM implementation and the fact that the results reported for it were less complete. As with the corresponding base implementations, ILK-JM and ILK-Neto seem to produce similar results, except that ILK-Neto is somewhat slower and does better on Random Matrices. The tables are divided into four sections, one for each class of instances in the Challenge testbed. Within each section, heuristics are ordered and grouped together according to the tour quality they provide. The grouping is somewhat subjective, but attempts to reflect performance over all instance sizes within a class. Thus for example for Clustered instances (C) we group CLK-ABCC- N and I3opt-10 N together even though the latter is much better for small N , because it is worse for large N . These groupings are carried over from the tour quality table to the running time table, so that the most cost-effective heuristic in each group can be identified.

The performance of a repeated-run heuristic naturally depends on how many iterations are performed. In practice one may simply run for as many iterations as can be handled within some fixed time limit or until a satisfactory solution is reached, and several of these implementations were designed with this usage in mind. For a scientific study, however, one needs results that are more readily reproducible, and hence a combinatorial stopping criterion is to be preferred. Here we bound the number of iterations by a function of the number N of cities. Typical bounds are $N/10$, N , or $10N$, as indicated by the suffix on the heuristic's name in the tables. For comparison purposes, the tables also include the results for both LK-JM and Helsgaun.

A first observation is that none of the heuristics in the table is consistently dominated by any other in both tour quality and running time. However, if running time is no object, then the iterated versions of Helsgaun's heuristic appear to be the way to go. Although Table 9.15 does not include rows for the optimal solution quality (which itself has a gap above the Held-Karp bound), the row for Helsgaun- N serves that purpose fairly well. The average excess for the optimal solution is known for all testbed instances with 3,162 or fewer cities, and all TSPLIB and Random Matrix instances with 10,000 cities or fewer. Helsgaun- N 's average excess is within 0.01% of this value in all these cases except for the "1,000-city" TSPLIB instances, for which it is 0.04% above the optimal excess. Moreover, for no instance in the Challenge testbed with a known optimal solution is the Helsgaun- N tour more than 0.18% longer than optimum. Much of this quality is retained by the $N/10$ -iteration version, which uses substantially less time and can feasibly be applied to larger instances (although for instances with 10,000 or fewer cities even Helsgaun- N never takes more than four hours of normalized running time, which should be feasible in many applications). Both variants have running time growth rates that appear to be $\Omega(N^{2.5})$ or worse, however, and so computers will have to be a lot faster before they can be applied to million-city instances.

The basic Helsgaun heuristic is itself a strong performer in three of the four classes, although it does fall down seriously on Clustered instances and is probably not cost-effective compared to CLK-ACR- N on Uniform and TSPLIB instances. For Random Matrix instances, none of the other heuristics come close to Helsgaun and its iterated versions, and their running times are moreover quite reasonable. If there were ever a reason to solve instances like these in practice, however, optimization should be considered an option. Concorde was able to solve all the Random Matrix instances in our testbed to optimality using its default settings. The average normalized running times were also quite reasonable: For

Average Percent Excess over the HK Bound

		N =	1000	3162	10K	31K	100K	316K	1M	3M
U	LK-JM	1.92	1.99	2.02	2.02	1.97	1.96	1.96	1.92	
	I3opt-10N	1.16	1.21	1.29	1.37	1.35	1.37	1.37		-
	ILK-JM-.1N	1.25	1.19	1.24	1.36	1.29	1.31	1.32		-
	CLK-ABCC-N	1.02	0.98	0.90	0.89	0.92	0.95	0.91		-
	CLK-ACR-N	0.84	0.93	0.92	0.91	0.92	0.95	0.90	0.86	
	ILK-JM-N	0.90	0.87	0.89	1.01	0.94	0.94			-
	CLK-ABCC-10N	0.91	0.82	0.83	0.79	0.81	0.84			-
	Helsgaun	0.90	0.89	0.83	0.83					-
	Helsgaun-.1N	0.75	0.72	0.69	0.68					-
Helsgaun-N	0.74	0.71	0.68	0.67					-	
C	LK-JM	1.75	2.95	3.41	3.71	3.63	3.67			
	Helsgaun	1.25	2.00	3.32	3.58					
	CLK-ABCC-N	1.63	2.83	2.09	2.37	1.93	1.96			
	I3opt-10N	0.82	1.43	1.62	2.19	2.02	2.29			
	CLK-ABCC-10N	1.21	1.93	1.83	1.75	1.68				
	CLK-ACR-N	0.62	1.27	1.82	1.73	1.78	1.70			
	ILK-JM-.1N	0.87	1.33	1.19	1.91	1.69	1.79			
	ILK-JM-N	0.60	0.78	0.91	1.33	1.16				
	Helsgaun-.1N	0.57	0.65	1.05	0.54					
Helsgaun-N	0.54	0.62	0.83	0.53						
T	LK-JM	2.38	2.16	1.92	1.73	1.61				
	Iopt-10N	1.29	1.53	1.04	1.12	1.18				
	ILK-JM-.1N	1.62	1.33	1.05	1.07	0.96				
	CLK-ABCC-N	1.40	1.31	1.01	0.92	0.69				
	CLK-ACR-N	1.12	1.21	0.76	0.88	0.59				
	ILK-JM-N	1.20	1.06	0.73	0.76	0.66				
	CLK-ABCC-10N	1.47	1.12	0.71	0.89	0.52				
	Helsgaun	1.20	1.01	0.77	0.96	1.25				
	Helsgaun-.1N	1.05	0.87	0.55	0.60	0.57				
Helsgaun-N	1.00	0.86	0.53							
M	I3opt-10N	6.25	11.85	16.81						
	LK-JM	3.52	4.35	5.91						
	ILK-JM-.1N	2.12	2.88	4.62						
	CLK-ABCC-N	1.82	3.26	4.89						
	CLK-ACR-N	1.50	3.04	3.92						
	ILK-JM-N	1.11	2.30	3.60						
	CLK-ABCC-10N	1.18	2.39	3.74						
	Helsgaun	0.04	0.02	0.03						
Helsgaun-.1N	0.02	0.01	0.01							
Helsgaun-N	0.02	0.01	0.01							

Table 9.15. Tour quality results for repeated local search heuristics, with Helsgaun and LK-JM included for comparison purposes.

Average Normalized Running Time in Seconds

N =		1000	3162	10K	31K	100K	316K	1M
U	LK-JM	0.20	0.7	2	7	23	61	323
	I3opt-10N	4.41	17.3	73	255	868	2660	16200
	ILK-JM-.1N	0.88	4.2	22	117	553	2240	18600
	CLK-ABCC-N	2.22	11.1	64	267	912	2590	16000
	CLK-ACR-N	3.19	14.6	49	164	933	4230	19200
	ILK-JM-N	6.16	32.4	169	920	4530	18500	-
	CLK-ABCC-10N	20.09	98.4	597	2480	8500	24300	-
	Helsgaun	5.64	71.5	862	7820	-	-	-
	Helsgaun-.1N	6.89	113.7	1830	35400	-	-	-
Helsgaun-N	18.00	468.0	9390	229000	-	-	-	
C	LK-JM	1.66	5.0	15	59	173	495	
	Helsgaun	7.02	70.3	768	12800	-	-	
	CLK-ABCC-N	9.32	42.1	206	899	3110	9230	
	I3opt-10N	7.32	27.1	102	327	948	2380	
	CLK-ABCC-10N	90.31	387.0	2010	8810	29900	-	
	CLK-ACR-N	9.99	45.0	142	490	2830	13000	
	ILK-JM-.1N	17.03	74.7	258	1260	4160	14100	
	ILK-JM-N	167.93	698.4	2320	9600	33500	-	
	Helsgaun-.1N	12.0	119.0	1780	26700	-	-	
Helsgaun-N	67.47	727.0	11900	89500	-	-		
T	LK-JM	0.34	0.6	4	13	24		
	I3opt-10N	5.52	16.5	74	216	582		
	ILK-JM-.1N	1.70	3.7	50	358	663		
	CLK-ABCC-N	2.39	8.3	53	173	442		
	CLK-ACR-N	3.18	9.3	35	73	222		
	ILK-JM-N	11.43	31.5	360	2590	6130		
	CLK-ABCC-10N	22.12	73.9	489	1740	4180		
	Helsgaun	7.82	73.3	1060	7980	48200		
	Helsgaun-.1N	9.61	104.9	4900	120000	303000		
Helsgaun-N	21.57	404.0	17600	-	-			
M	I3opt-10N	8.12	57.0	394				
	LK-JM	1.15	12.8	161				
	ILK-JM-.1N	3.30	32.0	354				
	CLK-ABCC-N	33.90	377.0	3820				
	CLK-ACR-N	140.29	1410.0	9650				
	ILK-JM-N	13.14	119.2	1250				
	CLK-ABCC-10N	303.00	3040.0	29800				
	Helsgaun	7.78	100.6	1270				
	Helsgaun-.1N	8.11	101.3	1540				
Helsgaun-N	13.67	237.3	4580					

Table 9.16. Normalized running times for repeated local search heuristics. The “3M” column was omitted so the table would fit on a page. The missing entries are 1255 seconds for LK-JM and 94700 seconds for CLK-ACR-N.

10,000 cities the normalized optimization time was 1,380 seconds, only marginally more than that for *Helsgaun*.

At the other end of the spectrum, consider *I3opt-10N*. Although it does very poorly on Random Matrix instances, it finds significantly better tours than *LK-JM* for the other three classes, with its much greater running time balanced by the fact that it should be much easier to implement than any of the other heuristics in the table. Moreover, it substantially outperforms *Helsgaun* on large Clustered instances, beats *CLK-ABCC-N* for the smaller ones, and is fairly close to *ILK-JM-.1N* for Uniform and *TSPLIB* instances, with similar running times when N is large. However, if one looks at the details of its runs, it would appear that not much would be gained by providing *I3opt* more iterations. On average there were no further improvements in the last 15% of the $10N$ iterations. Thus it would appear that the tour quality achieved by *ILK-JM-N*, *CLK-ACR-N* and *CLK-ABCC-10N* is beyond the capabilities of *I3opt*. Note that the running time growth rate for *I3opt-10N* appears to be $O(N^{1.25})$, indicating that the use of don't-look bits is paying off asymptotically. A similar effect is observed for Lin-Kernighan based variants, and so faster computers will extend the range of these heuristics more readily than they will *Helsgaun* and its repeated-run variants.

Turning to the Lin-Kernighan-based variants, we see the effect of the fact that *ILK-JM* is based on a more powerful but slower Lin-Kernighan engine than *CLK-ABCC* and *CLK-ACR*. Even though *ILK-JM-N* performs only one tenth as many iterations as *CLK-ABCC-10N*, the running times for the two are (roughly) comparable. So are the tour lengths, with the exception of the Clustered instances. Here *ILK-JM-N* finds distinctly better tours, possibly because of its use of longer neighbor lists. As to *CLK-ACR*, note that *CLK-ACR-N* produces distinctly better tours than *CLK-ABCC-N* in all but the Uniform class (where they are comparable), so it is possible that results for *CLK-ACR-10N*, if we had them, would show better tours than *ILK-JM-N*, again in comparable time. Results for *ILK-JM-10N* are not included in the table for space reasons, but tend to yield an average tour-length improvement of 0.1% over *ILK-JM-N* for the three geometric classes (at the price of taking 10 times as long). For Random Matrix instances the improvement is closer to 1%.

At some point, however, simply running for more iterations may not be the best use of computation time. In our next section we consider other possibilities for getting the last ϵ improvement in tour quality: heuristics that use Chained Lin-Kernighan as a subroutine. But first, a brief digression to follow up our earlier remark about the relation between Tabu Search and Chained Lin-Kernighan.

Tabu Search. In its simplest form, a Tabu Search heuristic operates as follows. As with other local search variants, it assumes a neighborhood structure on solutions. Given the current solution, we find the the best neighbor (or the best of a random sample of neighbors) subject to certain “Tabu” restrictions needed to avoid cycling. One then goes to that neighbor, whether it is an improving move or not. Note that in effect this breaks the search into alternating phases. First we make improving moves until a local optimum (or at least an apparent local optimum) is found, then we perform a “kick” consisting of one or more uphill moves until we reach a solution from which we can once again begin a descent. A full Tabu Search heuristic is usually much more complicated than this, using various strategies for “diversification” and “intensification” of the search, as well as parameters governing lengths of Tabu lists and “aspiration levels,” etc., for which see [369, 370]. However, the underlying similarity to chained local optimization remains.

And note that, in the case of the STSP at least, Tabu Search comes with extra overhead: Whereas in most of the local search implementations studied above, one performs one of the first improving moves seen, in a Tabu Search heuristic one typically must generate a sizable collection of moves from which the best is to be picked. This perhaps partially explains the results observed for the one collection of Tabu Search variants submitted to the challenge, implemented by Dam and Zachariasen [238]. Their implementations allow for the possibility of using different neighborhood structures for the downhill and uphill phases with the choices being the 2-Opt neighborhood, the double bridge neighborhood (DB), the standard LK-search neighborhood (LK), and the Stem-and-Cycle variant on it, called “flower” in [238] (SC). Interestingly, the best tours are most often found by the variants closest to Chained Lin-Kernighan: those that use standard LK-search or the Stem-and-Cycle variant for the downhill phase and double-bridge moves for the uphill phase, with the LK-search variant being substantially faster.

Unfortunately the running times even for this version are sufficiently slow that it is almost totally dominated by ILK-JM-N and CLK-ACR-N. The latter almost always finds better tours and averages between 35 and 200+ times faster (normalized running time) depending on instance class and size. Details can be viewed at the Challenge website. Although the Tabu Search implementations did not use all the available speedup tricks and is not as highly optimized as ILK-JM-N and CLK-ACR-N, it seems unlikely that more programming effort would bridge this gap.

3.7. Using Chained LK as a Subroutine

Just as we were able to get improved results out of Lin-Kernighan by using it as a subroutine in Chained Lin-Kernighan, one might consider improving on Chained LK by using *it* in more complicated procedures. One simple possibility would be to use Chained LK to generate starting tours for a second local search heuristic. Given how effective Chained LK already is, the second heuristic would probably need a neighborhood structure that is quite different from that used by Chained LK. Balas and Simonetti have proposed a likely candidate in [80].

Balas and Simonetti Dynamic Programming. This approach starts by identifying a *home* city c_1 and without loss of generality represents all tours as ordered sequences of the cities starting with c_1 . For any fixed k , we say that such a tour T' is a *k-bounded neighbor* of a tour $T = c_{\pi[1]}, c_{\pi[2]}, \dots, c_{\pi[N]}$ if for no i, j with $i \geq j + k$ does city $c_{\pi[j]}$ occur after $c_{\pi[i]}$ in T' . Note that in this neighborhood structure, two tours can be neighbors even if they have no edges in common. This is in sharp contrast to the k -Opt neighborhood structure, in which neighboring tours differ in at most k edges. Moreover, the new neighborhood can be searched much more effectively. For each k there is a linear-time dynamic programming algorithm that for any tour T finds its *best k-bounded neighbor* [80]. The running time is exponential in k , but this is not a major problem for small k . One thus can use this algorithm to perform steepest-descent local search under the k -bounded neighborhood structure.

Simonetti submitted results to the Challenge using this approach with $k \in \{6, 8, 10\}$ and with CLK-ABCC-N used to generate starting tours. The combination was run on all benchmark instances with one million or fewer cities except the 10,000-city Random Matrix instance. Improvements over the starting tour were found on 20 of these 87 instances when $k = 6$, on 22 when $k = 8$ and on 25 when $k = 10$. On larger instances, improvements were found at a significantly higher rate. For problems over 30,000 nodes, improvements over the starting tour were found on 11 of 13 instances when using $k = 6$, and on 12 of 13 instances with $k = 8$ and 10. Improvements were small, however. Even for $k = 10$ there were only six improvements larger than 0.01% and only one larger than 0.02%. This was an improvement of 0.07% for r11323 and was already found with $k = 6$. In a sense this last improvement is quite substantial, since it reduced the tourlength from 0.60% above optimum to 0.53%. Moreover, even tiny improvements can be worthwhile if they can be obtained relatively inexpensively, which in this case they can. For $k = 8$ the added overhead for applying this algorithm to the tour output by CLK-ABCC-N

is minimal, ranging from 5 to 20%, and even for $k = 10$ the total running time was increased by at most a factor of 2.4. Thus this approach might make a worthwhile low-cost general post-processor to any heuristic, especially for the larger instances where it appears to be more successful. The code is currently available from Simonetti's TSP webpage, <http://www.contrib.andrew.cmu.edu/~neils/tsp/index.html>.

The second heuristic we cover also uses Chained LK to generate starting tours, but in a more sophisticated framework.

A Multi-Level Approach (Walshaw [818]). The idea is to recursively apply Chained LK (or any other local search heuristic) to a smaller "coalesced" instance in order to generate a starting tour for the full instance. A coalesced instance is created by matching nearby cities and requiring that the edge between them be included in all generated tours (*fixing* the edge). If the instance to be coalesced already has fixed edges, then the only cities that can be matched must have degree 0 or 1 in the graph of fixed edges and can't be endpoints of the same path in that graph. Note that a coalesced instance containing a fixed path can be modeled by a smaller instance in which that path is replaced by a single fixed edge between its endpoints and the internal cities are deleted. Starting with the original instance, we thus can create a sequence of smaller and smaller instances by repeatedly applying this approach until we reach an instance with four or fewer cities.

Having constructed this hierarchical decomposition, we then proceed as follows, assuming our base heuristic has been modified so that it always outputs a tour containing all fixed edges. We start by applying the heuristic to the last (smallest) instance created, using the heuristic's native starting tour generator. Thereafter, as we progress back up the hierarchy, we use the result of running the heuristic on the previous instance as the starting tour for the current instance. We end up running the base heuristic $\Theta(\log N)$ times, but most of the runs are on small instances, and overall running time is no more than 2 to 3 times that for running the base heuristic once on the full instance. See [818] for more details, including the geometry-based method for matching cities during the coalescing phase. Walshaw submitted results for two instantiations of the Multi-Level approach. The first (MLLK) used LK-ABCC as its base heuristic and the second (MLCLK-N) used CLK-ABCC-N. The base heuristics were forced to obey fixed-edge constraints by setting the costs of required edges to a large negative value.

MLLK could well have been discussed in the previous section. Although it can't compete with CLK-ABCC-N on average tour quality, for large instances it is 15 to 35 times faster, and it did find better tours for a significant number of Clustered instances. When compared to

LK-ABCC, it found better tours on average for all three geometric classes, with the advantage averaging 3% for Clustered instances. MLCLK-N did not obtain significant tour-length improvements over its base heuristic (CLK-ABCC-N) for Uniform instances, but it did find better tours for over half the TSPLIB instances and almost all the Clustered instances. Its tours averaged 1% better for this latter class.

The two approaches considered so far in this section needed little more than twice the time for running Chained LK. In the remainder of the section, we consider what one might do if one wants even better tours and is willing to spend substantially more computational resources to get them. Currently the people willing to pay this price are mainly researchers interested in testing the ultimate limits of heuristic techniques and in generating better upper bounds on unsolved testbed instances, but this is an active community. Work in this area has also followed the paradigm of using Chained LK as a subroutine, but now the key factor being exploited is the randomization inherent in the heuristic.

Multiple Independent Runs. Chained Lin-Kernighan is a randomized heuristic, not only because of possible randomization in its starting tour generation, but also because the kicks are randomly generated. If one runs it several times with different seeds for the random number generator, one is likely to get different tours. This fact can be

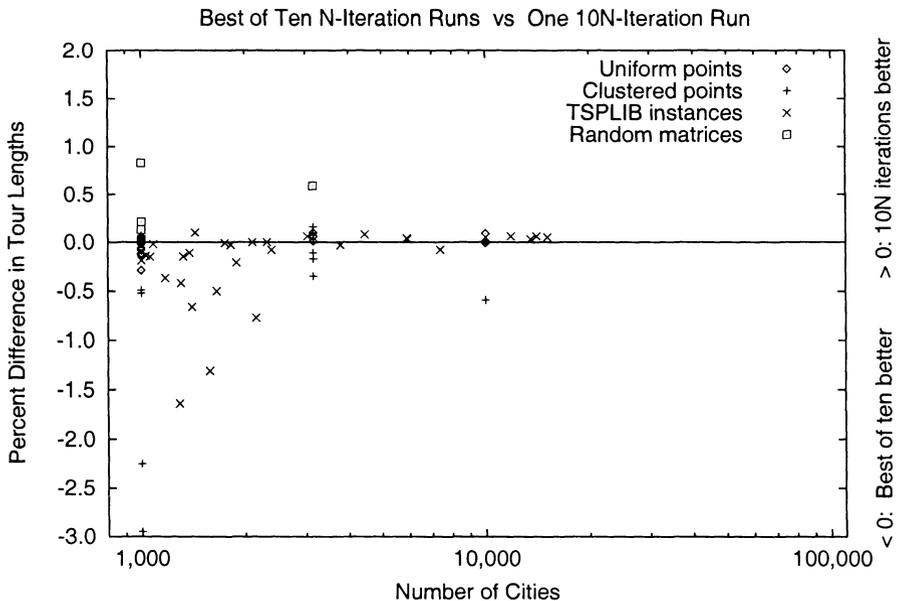


Figure 9.12. Tour quality comparisons between taking the best of ten runs of CLK-ABCC-N and taking the result of a single run of CLK-ABCC-10N.

exploited once one reaches the point where added iterations no longer seem to be helping, that is, it might be more effective to take the best of ten N -iteration runs instead of just performing one $10N$ -iteration run, which would take roughly the same time. See Figure 9.12, which for all the instances in our testbeds with 100,000 or fewer cities compares the best tour length over ten runs of CLK-ABCC- N (a composite heuristic we shall denote by CLK-ABCC- N -b10) with the tour length for one run of CLK-ABCC- $10N$. Although the “best of ten” strategy falls down for Random Matrix instances, it never loses by much on any instance from the other three classes, and it wins by significant amounts on many of the smaller Clustered and TSPLIB instances.

Note that taking the best of ten runs of a heuristic may not be enough to make up the gap between that base heuristic and a better heuristic. For example, Walshaw’s Multi-Level Approach using CLK-ABCC- N does even better on Clustered instances than does CLK-ABCC- N -b10 and takes only 1/5 as much time. Furthermore, Helsgaun’s variant on Chained LK is so good that, even with just $0.1N$ iterations it consistently outperforms CLK-ABCC- N -b10, as shown in Figure 9.13. Moreover, Helsgaun-.1N is typically faster for the smaller instances, although its running time grows much more rapidly with N , so that by the time one reaches 30,000 cities it is more than 10 times slower than the best-of-ten approach.

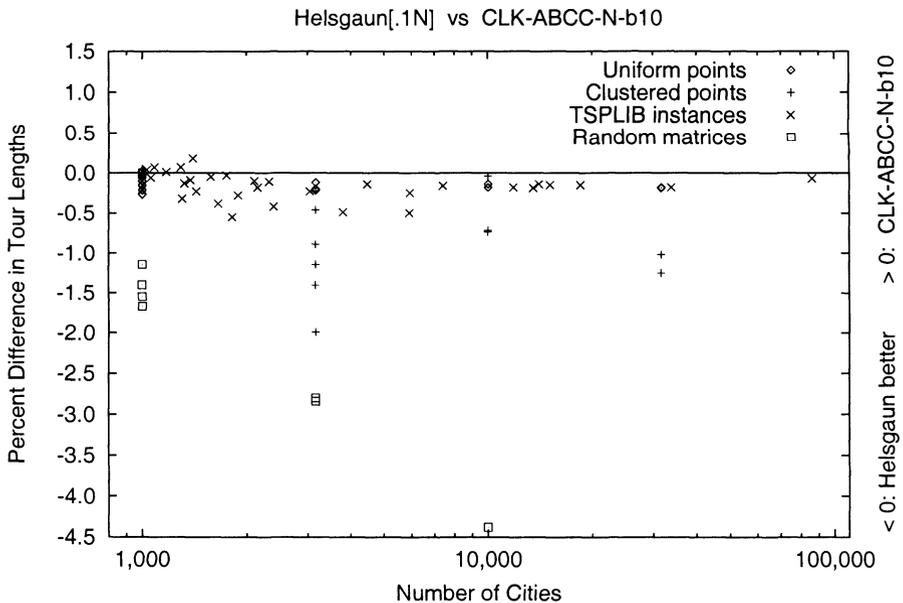


Figure 9.13. Tour quality comparisons between a $0.1N$ iteration run of Helsgaun’s LK variant and taking the best of ten runs of CLK-ABCC- N .

This doesn't, however, totally rule out the value of the simple best-of-ten approach. The best of ten runs of `Helsgaun-.1N` might be an interesting competitor for `Helsgaun-N`, assuming the running time is available. There are, however, more creative approaches to exploit the randomization involved in our heuristics.

Tour Merging (Applegate, Bixby, Chvátal, and Cook [27]). This approach is based on the observation that tours that are very close to optimal share many common edges. Moreover, if one takes the union of all the edges in a small set of such tours, one typically obtains a graph with low "branch-width," a graph parameter introduced by Robertson and Seymour [724] that in a sense quantifies how "treelike" a graph is.

This can be exploited algorithmically as follows. First, although it is NP-hard to determine the precise branch-width of a graph, there are effective heuristics for finding near-optimal branch-width decompositions of low branch-width graphs. Second, there is a dynamic programming algorithm due to Cook and Seymour [208] that, given an edge-weighted Hamiltonian graph G and a branch decomposition for it with branch-width k , finds the shortest Hamiltonian cycle in G in time linear in N . The running time is exponential in k but is quite feasible for small values. Thus one can typically take the results of several runs of Chained LK and find the optimal tour contained in the union of their edges. During the last few years, this technique has helped Applegate, Bixby, Chvátal, and Cook solve many previously unsolved TSPLIB instances. Note that in this approach one needn't restrict oneself to Chained LK tours. Applegate et al. typically performed many tour-merging runs, often using the result of one run as one of the tours to be merged in a subsequent run. Moreover, the recent solution of `d15112` was aided by improved upper bounds that were obtained by applying tour-merging to Iterated Helsgaun tours in addition to Chained LK tours.

Although tour-merging has most typically been used in a hands-on fashion, with manual choice of tours to be merged, etc., it is possible to run it in a stand-alone manner. Cook submitted results to the Challenge for a heuristic of this sort (`Tourmerge`), in which the tour-merging process is applied to the results of ten runs of `CLK-ABCC-N`. Five runs of `Tourmerge` were performed on the set of all testbed instance with fewer than 10,000 cities. The heuristic was not always successful. For none of the runs on Random Matrix instances did the combined graph for the 10 tours have low enough branch-width for tour-merging to be feasible. Similar failures occurred in all runs on the TSPLIB instances `u2319`, `fn14461`, and `pla7397` and in one or two runs on each of the 3162-city Uniform instances. Nevertheless, results *were* generated for most of the instances attempted and were substantially better than those for sim-

ply taking the best of the ten runs of CLK-ABCC-N. Moreover, if we take the best of the five tour-merging runs (a composite heuristic we shall denote by *Tourmerge-b5*), we now become competitive with *Helsgaun's* heuristic, even if the latter is allowed N rather than $0.1N$ iterations.

Figure 9.14 compares the results of *Tourmerge-b5* and *Helsgaun-N* on all instances for which *Tourmerge-b5* generated tours. Note first the much narrower range in differences. *Tourmerge-b5* is never more than 0.05% worse (when it actually produces a solution) and is never more than 0.20% better. This is in comparison to the -4.5% to $+1.5\%$ range in Figure 9.13. The biggest variation is on TSPLIB instances, where *Tourmerge-b5* does better more often than it does worse.

The normalized running time for performing all five tour-merging runs can however be substantially worse than that for one run of *Helsgaun-N*, typically some 5 times slower but occasionally as much as 100. (For the instances where some of the runs had to be aborted because of high branch-width, no times were reported, so we estimated the running time for the failed run as the average of those for the successful runs.) Moreover, for 14 of the 54 instances on which *Tourmerge-b5* was successfully run, its normalized time was greater than that for finding the *optimal* solution and proving optimality using *Concorde's* default settings. In five cases *Concorde* was faster by a factor of 4 or more. (*Helsgaun-N* was

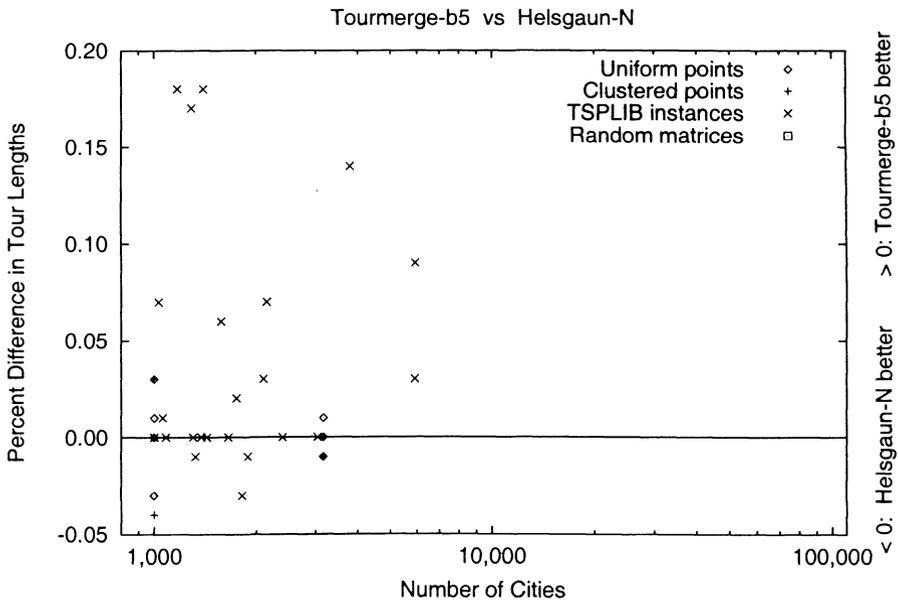


Figure 9.14. Tour quality comparisons between an N -iteration run of *Helsgaun's* LK variant and taking the best of five tour-merges, each involving ten CLK-ABCC-N tours.

slower than *Concorde*-optimization on only two instances.) Nevertheless, the time for *Tourmerge-b5* would probably be manageable in many applications. With the exception of the 3,162-city Uniform instances, no instance with less than 10,000 cities required more than an hour of normalized time, and even the slowest 3,162-city instance took just 16 hours (and for this instance *Tourmerge-b5* found an optimal solution, whereas full optimization via *Concorde* under default settings was still far from complete when terminated after several days).

The fact that neither *Tourmerge-b5* nor *Helsgaun-N* beats the other on all instances suggests a final “heuristic” for getting improved results at the cost of increased running times: Run both and take the best tour found. Better yet, run both plus the several other effective heuristics from this and the previous section and take the best tour found. Table 9.17 summarizes how that approach would work on the subset of the 71 Challenge testbed instances for which optimal solution values are known, using the four heuristics *Tourmerge-b5*, *Helsgaun-N*, *ILK-JM-10N*, and *ILK-JM-N-b10* (the best of ten runs of *ILK-JM-N*).

Running times for large instances would of course be huge, but this approach might well be the method of choice when *Concorde* is unsuccessful in optimizing within the large number of hours or days one is willing to spend. Recall from Table 9.1 that for a significant subset of the instances with fewer than 10,000 cities *Concorde* under its default settings took more than 100 (normalized) hours if it succeeded at all, and it failed for all our geometric instances with more than 4,000 cities.

4. Conclusions and Further Research

In this chapter we have discussed a wide variety of heuristics occupying many positions on the tradeoff curve for tour quality and running time. Moreover, we have compared their performance on a wider range of instance types and sizes than in previous studies. In order to get such broad coverage, however, we have had to accept some compromises, and it is appropriate to remind the reader about these.

First, for many of the codes we cover, the results we include come from unverified reports submitted by the implementers, based on runs on a variety of machines. As discussed in Section 2.2, our methodology for normalizing reported running times is necessarily inexact. Second, as seen in many places in this chapter, two implementations of the same heuristic can have markedly different running times, even on the same machine, depending on the data structures and coding expertise used. For local search heuristics, even the tour lengths can differ depending on implementation choices. Thus the conclusions we draw are often as

Instance	Percent Above		Instance	Percent Above	
	Optimal	Heuristic		Optimal	Heuristic
E1k.0	-	h	M3k.0	-	h
E1k.1	-	h,m	M3k.1	-	h
E1k.2	-	h,m	M10k.0	0.0033	h
E1k.3	-	m,i10	dsj1000	-	h
E1k.4	-	h,m	pr1002	-	h,m,i10
E1k.5	-	h,m,ib	si1032	-	m,ib
E1k.6	-	h	u1060	-	m,i10
E1k.7	-	m	vm1084	-	h,m
E1k.8	0.0012	m	pcb1173	-	m
E1k.9	-	h,m,ib	d1291	-	ib
E3k.0	0.0034	h	rl1304	-	h,ib,i10
E3k.1	-	h	rl1323	-	h,ib
E3k.2	-	h	nrv1379	-	h,m
E3k.3	0.0174	m	fl1400	-	m,ib,i10
E3k.4	-	m	u1432	-	h,m
C1k.0	-	h,m	fl1577	-	m,ib
C1k.1	-	h,m,ib,i10	d1655	-	h,m
C1k.2	-	h,ib	vm1748	-	m,ib,i10
C1k.3	-	h,m,ib,i10	u1817	-	h
C1k.4	-	i10	rl1889	0.0013	h
C1k.5	-	ib,i10	d2103	-	ib,i10
C1k.6	-	h	u2152	-	h
C1k.7	-	h,ib	u2319	-	m
C1k.8	-	h,m,ib,i10	pr2392	-	h,m
C1k.9	-	m,ib,i10	pcb3038	-	h
C3k.0	-	h,i10	fl3795	-	m,i10
C3k.1	-	h	fnl4461	0.0027	h
C3k.2	-	i10	rl5915	0.0057	m
C3k.3	-	m	rl5934	0.0023	m
C3k.4	-	h,m	pla7397	-	h
M1k.0	-	h	rl11849	0.0610	h
M1k.1	-	h	usa13509	0.0065	h
M1k.2	-	h	d15112	0.0186	h
M1k.3	-	h			

Table 9.17. Best results obtained for all testbed instances whose optimal solutions are known. Four heuristics sufficed to generate these results: **Helsgaun-N** (abbreviated in the table as h), **Tourmerge-b5** (m), **ILK-JM-10N** (i10), and best of ten runs of **ILK-JM-N** (ib). Where more than one heuristic found the best solution, we list all that did.

much about the implementation as about the heuristics implemented. Finally, most of our results only cover one run of the given heuristic on each instance, which for heuristics that incorporate randomization can lead to very noisy data. For this reason we have tried to concentrate on averages over similarly sized instances and on observable patterns in the data, rather than results for particular instances.

Time constraints have also had their effect, forcing us to concentrate on just four instance classes with an emphasis on 2-dimensional geometric instances, the type that occur most often in practice. We also have not considered in detail the dependence of heuristic performance on parameter settings (for example the lengths of neighbor lists, the choice of starting tour, or the nature of the “kick” in Chained Lin-Kernighan). Such questions should more naturally fall in the domain of individual papers about the heuristics in question, and we hope the Challenge testbeds and benchmarks will facilitate such work in the future. We intend to provide just such detailed experimental analysis for the Bentley and Johnson-McGeoch implementations discussed here in the forthcoming monograph [461]. As mentioned earlier, we also plan to maintain the Challenge website indefinitely as a resource and standard of comparison for future researchers.

If the reader is to take one lesson away from this chapter, it should be the high level of performance that can be attained using today’s existing heuristics, many of them with publicly available implementations. A second major lesson concerns the large extent to which a heuristic’s performance (both running time and tour quality) can depend on implementation details, as we have seen many times in this chapter.

As a final lesson, let us review once more the wide range of trade-offs to which we referred above. Recall that we have already provided one illustration of this in Section 3. In that section, Figure 9.3 and Table 9.2 showed the range of performance possibilities for Random Uniform Geometric instances. For such instances and most heuristics, the percent excess over the Held-Karp bound appears to approach a rough limiting bound, which typically has almost been reached by the time $N = 10,000$. The table and figure presented average results for selected heuristics on Uniform instances of that size, covering a wide range of behavior.

Results can be more instance-dependent for structured instances such as those in TSPLIB. To put the earlier table in perspective (while partially ignoring the above-mentioned proviso about drawing detailed conclusions from single runs on individual instances), we conclude our discussion with Table 9.18, which considers the currently largest solved TSPLIB instance d15112, and gives both the percentage excess above optimum and the normalized running time obtained for it by all the key

implementations covered in the chapter, as well as a few related ones. The abbreviated heuristic names used in the table and elsewhere in this chapter are explained in Table 9.19. Analogous tables for other instances can be generated from the Comparisons page at the Challenge website.

Acknowledgment. The authors thank Jon Bentley, Gregory Gutin, and César Rego for helpful comments on drafts of this chapter and all the Challenge participants for their participation and feedback.

Percent Above Optimal	Running Time (Seconds)	Heuristic	Percent Above Optimal	Running Time (Seconds)	Heuristic
-0.5215	90.13	HeldKarp	2.8124	3.27	3opt-B
0.0000	-	Optval	3.0729	16.12	Hyper-3
0.0186	26322.93	Helsgaun-N	3.1389	2773.49	GENIUS-10
0.0236	7896.88	Helsgaun-.1N	3.7261	1.94	2.5opt-B
0.1051	1515.99	Helsgaun	4.0234	2.53	2opt-JM-40
0.1266	980.66	CLK-ABCC-10N	4.2518	1.73	2opt-JM-20
0.1358	8738.10	ILK-JM-10N	4.2704	1.81	2opt-JM-20a
0.1744	1046.75	CLK-ABCC-N-b10	4.3852	2.53	2opt-JM-40a
0.1810	102.55	CLK-ABCC-N	4.5620	2.47	Hyper-2
0.1952	154.79	MLCLK-N	4.7515	325.66	GENI-10
0.1988	3408.07	ILK-JM-N-b10	5.1630	1.56	2opt-JM-10
0.2013	65.36	CLK-ACR-N	5.2668	1.81	2opt-B
0.2114	89.59	MLCLK-.5N	6.2298	14.14	Christo-HK
0.2259	542.61	ILK-Neto-N	8.9277	1.65	Christo-G
0.2447	332.16	ILK-JM-N	10.4116	0.73	AppChristo
0.3359	108.29	ILK-JM-.3N	10.7438	2578.46	CCA
0.4451	21.53	MLCLK-.1N	11.0510	0.41	Savings
0.5232	42.07	ILK-JM-.1N	11.7303	4.67	FI
0.6138	6.55	CLK-ACR-1000	11.8092	2.33	FA ⁺
0.6282	69.52	ILK-Neto-.1N	12.9387	3.38	RI
0.6464	112.96	I3opt-10N	13.6291	1.65	Christo-S
0.6747	15.64	LK-HK	13.6803	1.13	RA ⁺
0.7579	38531.28	Tabu-SC-SC	13.6803	0.17	Q-Boruvka
0.9739	2203.88	Tabu-LK-DB	14.5281	0.26	Boruvka
1.0929	4.96	LK-JM-40	14.7968	0.44	Greedy-ABCC
1.1038	3.03	LK-JM-20	15.9946	31.09	Litke-15
1.1418	5.70	LK-Neto-20	17.1529	1.31	CI
1.1443	11.74	LK-Neto-40	17.3106	1.38	CHCI
1.1620	2.98	LK-JM-20a	22.3039	2.84	NI
1.1975	35660.20	Tabu-SC-DB	22.3094	2.01	NA ⁺
1.2170	2.61	LK-JM-10	23.1644	0.49	DENN
1.2549	4.23	LK-Neto-12	24.6647	0.14	NN-ABCC
1.2678	498.12	SCLK-R	28.5153	1.59	NA
1.3357	134.58	SCLK-B	31.0319	100.03	Karp-20
1.3422	2.01	CLK-ACR-100	32.0524	0.06	Spacefill
1.4427	4.97	MLLK	36.1238	0.55	DblMST
1.8178	2.38	LK-ABCC	37.0620	0.60	RA
1.8456	1.53	CLK-ACR-10	42.1799	1.77	FA
1.8761	1.46	LK-ACR	42.8104	0.05	Strip
2.1715	1.90	3opt-JM-20	42.8104	0.07	Strip2
2.3504	2.73	3opt-JM-40	49.2882	0.17	FRP
2.5974	136.37	Hyper-4	59.2344	0.26	Karp-15
2.7254	1.70	3opt-JM-10	-	0.03	Read

Table 9.18. Tour quality and normalized running times for TSPLIB instance d15112. General conclusions should not be drawn from small differences in quality or time.

Abbrev	Short Description of Heuristic
AppChristo	Approximate Christofides (JM)
{Q-}Boruvka	Concorde's implementation of (Quick) Boruvka
CCA	The Golden-Stewart CCA heuristic (JM)
{CH}CI	The JM implementations of (Convex Hull) Cheapest Insertion
CLK- X - k	The X version of Chained LK with k iterations
Christo-{S,G}	The Christofides heuristic with {standard,greedy} shortcuts (JM)
Christo-HK	Christofides using Held-Karp one-trees instead of MST's (Rohe)
Concorde	Concorde used for optimization with default settings
Db1MST	The Double Minimum Spanning Tree heuristic (JM)
F{I,A,A ⁺ }	Bentley's farthest {insertion,addition,augmented addition}
FRP	Bentley's implementation of the Fast Recursive Partitioning heuristic
GENI{US}- p	The Gendreau-Hertz-Laporte GENI(US) heuristic with p neighbors
Greedy{- X }	The X implementation of Greedy (Default: JM implementation)
HeldKarp	Held-Karp bound as computed by Concorde
Helsgaun{- k }	Helsgaun's heuristic (with k iterations)
Hyper- k	The Burke-Cowling-Keuthen implementation of k -Hyperopt
I3opt- k	The JM implementation of Iterated 3-Opt with k iterations
ILK- X - k	The X version of Iterated LK with k iterations
Karp- n	Karp's Partitioning heuristic for maximum subproblem size k (JM)
k opt- X	The X implementation of k -Opt, $k \in \{2, 2.5, 3\}$
k opt-JM{- p }	The JM implementation of k -Opt with p quad neighbors
Litke- k	Litke's Clustering heuristic for maximum subproblem size k (JM)
LK- X - p	The X version of basic Lin-Kernighan with p quad neighbors
LK- X {-BD}	The X version of basic Lin-Kernighan with default neighbor lists (and bounded depth LK-searches if that is not the default)
LK-HK	Lin-Kernighan using Christo-HK starts (Rohe)
MLCLK- k	Walshaw's implementation of Multi-Level k -iteration Chained LK
MLLK	Walshaw's implementation of Multi-Level Lin-Kernighan
N{I,A,A ⁺ }	Bentley's nearest {insertion,addition,augmented addition}
NN{- X }	The X implementation of Nearest Neighbor (Default: B)
Optval	Optimal solution lengths (from a variety of sources)
R{I,A,A ⁺ }	Bentley's random {insertion,addition,augmented addition}
Read	Time to simply read the instance using standard I/O routines
Savings	The JM implementation of the Clarke-Wright "Savings" heuristic
SCLK-{R,B}	The Glover-Rego implementation of a Stem-and-Cycle variant of Lin-Kernighan with {random,boruvka} starts
Spacefill	The Bartholdi-Platzmann implementation of Spacefilling Curve
Strip{2}	The JM implementation of the Strip (2-Way Strip) heuristic
Tabu- D - U	The Dam-Zachariasen implementation of Tabu Search using the D (U) neighborhood for downhill (uphill) moves
Tourmerge	The tour-merging heuristic of ABCC applied to 10 runs of CLK-ABCC-N

Table 9.19. Abbreviated names used in this chapter. The symbol X stands for an abbreviation of the implementers' names: "ABCC" for Applegate, Bixby, Chvátal, and Cook (Concorde), "ACR" for Applegate, Cook, and Rohe, "B" for Bentley, "JM" for Johnson-McGeoch, "Neto" for Neto, and "R" for Rohe. Adding the suffix "- bn " to any name means that one is taking the best of n runs.

Chapter 10

EXPERIMENTAL ANALYSIS OF HEURISTICS FOR THE ATSP

David S. Johnson

AT&T Labs – Research, Room C239, Florham Park, NJ 07932, USA
dsj@research.att.com

Gregory Gutin

Department of Computer Science, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK, G.Gutin@rhul.ac.uk

Lyle A. McGeoch

Department of Mathematics and Computer Science,
Amherst College, Amherst, MA 01002, USA, lam@cs.amherst.edu

Anders Yeo

Department of Computer Science, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK, anders@cs.rhul.ac.uk

Weixiong Zhang

Department of Computer Science, Washington University, Box 1045
One Brookings Drive, St. Louis, MO 63130, USA, zhang@cs.wustl.edu

Alexei Zverovitch

Department of Computer Science, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK, A.Zverovitch@rhul.ac.uk

1. Introduction

In this chapter, we consider what approaches one should take when confronting a real-world application of the asymmetric TSP, that is, the general TSP in which the distance $d(c, c')$ from city c to city c' need not equal the reverse distance $d(c', c)$. As in the previous chapter on heuristics for the symmetric TSP, we will discuss the performance of a variety of heuristics exhibiting a broad range of tradeoffs between running time and tour quality.

The situation for the ATSP is different from that for the STSP in several respects. First, no single type of instance dominates the applications of the ATSP in the way that two-dimensional geometric instances dominate the applications of the STSP. General-purpose ATSP heuristics must thus be prepared to cope with a much broader range of instance structures and are less likely to be robust across all of them. Moreover, the variety of possible instance structures rules out the possibility of a common linear-size instance representation. Thus general-purpose codes rely on instance representations that simply list the $N(N - 1)$ inter-city distances, where N is the number of cities. This makes it difficult to handle large instances: if each distance requires say four bytes of memory, a 10,000-city instance would require roughly 400 megabytes of memory. Thus the experimental study of asymptotic behavior is much less advanced than in the case of the STSP. Whereas TSPLIB¹ contains STSP instances with as many as 85,900 cities, the largest ATSP instance it contains has only 443 cities and over half of its ATSP instances have fewer than 100 cities.

A second significant difference between the STSP and the ATSP is that the latter appears to be a more difficult problem, both with respect to optimization and approximation. Whereas all the randomly generated and TSPLIB instances with 1000 cities or less covered in the chapter on the STSP could be optimally solved using the Concorde² optimization code under its default settings, many 316-city ATSP instances from [200] remain unsolved. Moreover, as we shall see, there are plausibly realistic instance classes where *none* of our heuristics get as close to optimal as several STSP heuristics did for all the geometric instances in the testbeds covered in the STSP chapter.

¹TSPLIB is a database of instances for the TSP and related problems created and maintained at <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/> by Gerd Reinelt and described in [709].

²Concorde is a publicly available software package written by Applegate, Bixby, Chvátal, and Cook, described in [29].

Finally, the experimental study of heuristics for the ATSP is much less advanced. Until recently there has been no broad-based study covering a full range of ATSP heuristics and what ATSP studies there have been, such as [377, 714, 837, 838], have concentrated mostly on the TSPLIB instances and randomly generated instances with no obvious connection to applications. Nor has there been a formal ATSP Challenge like the STSP Challenge that served as the main resource for Chapter 9. Fortunately, a first attempt at a more comprehensive study has recently appeared: a 2001 paper by Cirasella et al. [200]. This chapter will build on the results in [200] by looking at them in new ways and augmenting them with results for several additional heuristics.

The chapter is organized as follows. In Section 2 we describe the standards we use for measuring tour quality and comparing running times and the instance classes that make up our testbeds. Section 3 describes the various heuristics we cover, divided into groups based on approach taken (modified STSP tour construction heuristics, heuristics that solve assignment problems as a subroutine, and local search heuristics). Section 4 then describes the results for the various heuristics on our testbeds and how they compare, and attempts to derive meaningful insights based on the wealth of disparate data provided. We conclude in Section 5 with a summary of our major conclusions and suggestions for future research.

For a look at ATSP heuristics from a more theoretical point of view, see Chapter 6.

2. Methodology

In this chapter we evaluate heuristics on the basis of their performance on the collection of benchmark instances used in [200]. We begin in Section 2.1 by discussing the various measures of tour quality that we use in our comparisons (the optimal tour length, the Held-Karp lower bound, and the assignment problem lower bound) and the relations between them. In Section 2.2 we describe how running times were measured and the normalization process by which running times on different machines were compared (a variant on the approach taken in Chapter 9). In Section 2.3 we describe the benchmark instances and how they were obtained or generated.

2.1. Evaluating Tour Quality

The most natural standard for tour quality is the distance from the optimal solution, typically measured as the percentage by which the tour length exceeds the length of an optimal tour. In order to use this standard, one unfortunately must *know* the optimal solution value, and

as remarked above, optimization technology for the ATSP is even less advanced than that for the STSP. Whereas optimal tour lengths were known for all the STSP testbed instances with 3,162-city cities or less, there are quite a few 316-city instances in our ATSP testbeds for which we have been unable to determine the optimal tour length.

Thus, as was the case for the STSP, we must settle for some computable lower bound on the optimal tour length as our standard of comparison. We shall actually consider two such bounds. Until recently, the most commonly studied lower bound for the ATSP was the *Assignment Problem* (AP) lower bound: Treat the distance matrix as representing the edge weights for a complete (symmetric) bipartite graph on $2N$ vertices, and compute the minimum-cost perfect matching for this graph, a computation that can be done in worst-case $\Theta(N^3)$ time but usually can be performed much more quickly. It is not difficult to see that this matching corresponds to a minimum-cost cover of the cities by vertex-disjoint directed simple cycles, and hence is a lower bound on the optimal tour length, which is the minimum cost cover of the cities by a single directed simple cycle. As we shall see, the AP bound is quite close to the optimal tour length for many of our instance classes. However, this is not always true. In the worst case the percent by which the optimal tour length exceeds it is not bounded by any constant even for $N = 4$. For five of our random instance classes the average gap is well over 10%.

To obtain a better lower bound, [200] proposed using the generalization of the Held-Karp STSP lower bound (HK bound) [444, 445] to the asymmetric case. As in the symmetric case, the asymmetric HK bound is the solution to the linear programming relaxation of the standard integer programming formulation of the problem. For the ATSP, that formulation is presented in Chapter 4. As in the symmetric case, we can use *Concorde* to compute it, although to compute the bound for an ATSP instance I , we first need to transform the instance to an STSP instance I' to which *Concorde* can be applied. We use the standard transformation that replaces each city c_i by three cities c_i^1, c_i^2, c_i^3 with an (undirected) path of 0-cost edges linking them in the given order. The directed arc (c_i, c_j) then corresponds to the undirected edge $\{c_i^3, c_j^1\}$. All edges not specified in this transformation are given lengths longer than the tour computed by Zhang's heuristic for the ATSP instance. It is easy to see that the optimal undirected tour for I' has the same length as the optimal directed tour for I and that the HK bounds likewise coincide. The (normalized) time to compute the HK bound by this process (including running Zhang's heuristic and performing the transformation) ranges from seconds for 100-city instances to 35 minutes for 3,162-city instances. For full running time details, see the tables of Section 4.6.

This same approach can be used to attempt to find optimal tours, simply by setting *Concorde*'s flags to make it do optimization instead of computing the Held-Karp bound. For this a good upper bound is needed, for which we would now recommend running *Helsgaun*'s heuristic (different heuristics were used in [200]). When the optimum could be computed, the maximum average gap between it and the HK bound for any of our instance classes was 1.86%, with most averages being well under 1%. The maximum gap for any of the real-world instances in our testbed was 1.79%. Thus the HK bound appears to be a much more robust estimate of the optimal solution than the AP bound. Moreover, as we shall see, the gap between the AP and HK bounds has interesting correlations with algorithmic behavior.

2.2. Running Time Normalization

As in the chapter on STSP heuristics, this chapter summarizes results obtained on a variety of machines. Although we rely heavily on the results reported in [200], which were all obtained on the same machine (an SGI Power Challenge using 196 Mhz MIPS R10000 processors), we have augmented these with results from additional researchers, performed on their home machines. We thus again need a mechanism for comparing running times on different machines.

As in the STSP chapter, we base our comparisons on normalization factors derived from runs of a standard benchmark code. For the ATSP, our benchmark code is an implementation of the "Hungarian method" for solving the Assignment problem (available as part of the ATSP download from the TSP Challenge website [464]). It is intended to reflect ATSP computations more accurately than the Greedy geometric benchmark code used for the STSP. Contributors ran this code on 100-, 316-, 1,000-, and 3,162-city instances from the *rtilt* class described below on the same machine they used to test their own heuristics and reported the benchmark's running times along with the results for their own heuristics. The benchmark times were then used to construct size-dependent normalization factors by which their heuristics' running times could be multiplied to produce a rough estimate of what they would have been on our target machine. To facilitate comparisons with results from the STSP chapter, we use the same target machine: a Compaq ES40 with 500 Mhz Alpha processors and 2 gigabytes of main memory.

For more details on the conversion process (and its potential weaknesses), see Chapter 9. Based on limited tests, it appears that our normalization process is likely once again to introduce no more than a factor-of-two error in running time. Indeed, for the ATSP and its bench-

		N =	100	316	1000	3162
Class	Heuristic		Time in Seconds			
disk	Zhang	Normalized	.05	.56	6.4	105
		Actual	.09	.54	7.0	111
		Discrepancy	-44%	+4%	-9%	-5%
coin	Helsgaun	Normalized	1.40	17.45	205.4	4975
		Actual	.90	13.20	230.7	4522
		Discrepancy	+55%	+32%	-11%	+10%

Table 10.1. Comparisons between normalized 196 Mhz MIPS processor time and actual running time on the 500 Mhz Alpha target machine.

mark code, we may be slightly more accurate than that. See Table 10.1, where the predicted and actual running times on the target machine are summarized for two pairs of (heuristic,instance class). As in the symmetric case, we get both under- and over-estimates, but here the times are typically within 33% of each other except for the 100-city instances where running times are too small to estimate accurately.

2.3. Testbeds

Thirteen classes of instances were covered in [200]. Twelve of those classes were randomly generated. The generators for seven of the twelve classes were designed to produce instances with some of the structure that might arise in particular applications. The other five generators produced less-structured instances of types previously studied in the literature, three of which were actually symmetric instances viewed as asymmetric ones. For each class there were ten instances with 100 and 316 cities, three instances with 1,000 cities, and one instance with 3,162 cities. The thirteenth class consisted of “real-world” instances from TSPLIB and other sources, ranging in size from 43 to 1,000 cities.

For space reasons, in this chapter we consider a slightly restricted set of testbeds. First, we ignore all real-world instances with fewer than 100 cities, just as STSP instances with fewer than 1,000 cities were ignored in Chapter 9. (The lower threshold in the case of the ATSP reflects the fact that this problem appears to be harder in general.) All the real-world instances with 100 cities or less from [200] and all the 100-city randomly generated instances in our testbed can be solved to optimality in reasonable time using the technique described in Section 2.1. Moreover, the publicly available implementation of Helsgaun’s heuristic [446] should suffice for most practical situations when instances are this small. It gets within 0.3% of optimal within a (normalized) second for all the real-world instances with less than 100 cities from [200], and as we shall see it averages two seconds or less to get within 0.1% of optimal for the 100-city instances in each of our random classes.

One can conceive of applications with a greater need for speed, for example the phylogenetic inference problem discussed by Moret et al. in [606, 607], where millions of ATSP instances with 37 cities or less need to be solved as part of the overall process. However, in such cases it typically becomes necessary to use extensive problem-specific algorithm engineering, which is well beyond the scope of this chapter. (In fact, Moret et al. ended up using a highly tuned branch-and-bound optimization code for their application [606].) Thus our lower bound of 100 cities seems quite reasonable.

In addition, we restrict ourselves to just two of the three symmetric instance classes from [200]: the ones corresponding to the Random Uniform Euclidean instances and Random Matrix instances of Chapter 9. We consider these classes mainly to illustrate the loss in tour quality that ATSP heuristics experience in comparison to their STSP counterparts. The third class, consisting of the closure under shortest paths of Random Matrix instances, adds little additional insight to this question.

We now say a bit about each of the twelve classes we do cover. For more details the interested reader is referred to [200] or to the code for the generators themselves, which together with associated README files is available from the DIMACS Implementation Challenge website [464].

Random Symmetric Matrices (smat). These are the Random Matrix instances of Chapter 9 written in asymmetric format. For this class, $d(c_i, c_j)$ is an independent random integer x , $0 \leq x \leq 10^6$ for each pair $1 \leq i < j \leq N$, and $d(c_i, c_j)$ is set to $d(c_j, c_i)$ when $i > j$. (Here and in what follows, “random” actually means pseudorandom, using an implementation of the shift register random number generator of [508, pages 171–172].) Such instances have no plausible application, but have commonly been studied in this context and at least provide a ground for comparison to STSP heuristics.

Random 2-Dimensional Rectilinear Instances (rect). These correspond to the Random Uniform Euclidean instances of Chapter 9 written in asymmetric format, except that we use the Rectilinear rather than the Euclidean metric. That is, the cities correspond to random points uniformly distributed in a 10^6 by 10^6 square, and the distance between points (x_1, y_1) and (x_2, y_2) is $|x_2 - x_1| + |y_2 - y_1|$. For STSP heuristics, tour quality and running times for the Euclidean and Rectilinear metrics are roughly the same [461].

Random Asymmetric Matrices (amat). The random asymmetric distance matrix generator chooses each distance $d(c_i, c_j)$ as an independent random integer x , $0 \leq x \leq 10^6$. For these instances it is known that both the optimal tour length and the AP bound approach a con-

stant (the same constant) as $N \rightarrow \infty$. The rate of approach appears to be faster if the upper bound U on the distance range is smaller, or if the upper bound is set to the number of cities N , a common assumption in papers about optimization algorithms for the ATSP (e.g., see [164, 597]). Surprisingly large instances of this type can be solved to optimality, with [597] reporting the solution of a 500,000-city instance. (Interestingly, the same code was unable to solve a 35-city instance from a real-world manufacturing application.) Needless to say, there are no known practical applications of asymmetric random distances matrices. We include this class to provide a measure of comparability with past studies.

Shortest-Path Closure of amat (tmat). One of the reasons the previous class is uninteresting is the total lack of correlation between distances. Note that instances of this type are unlikely to obey the triangle inequality, i.e., there can be three cities c_1, c_2, c_3 such $d(c_1, c_3) > d(c_1, c_2) + d(c_2, c_3)$. Thus heuristics designed to exploit the triangle inequality, such as the Repeated Assignment heuristic of [324] may perform horribly on them. A somewhat more reasonable instance class can be obtained by taking Random Asymmetric Matrices and closing them under shortest path computation. That is, if $d(c_i, c_j) > d(c_i, c_k) + d(c_k, c_j)$ then set $d(c_i, c_j) = d(c_i, c_k) + d(c_k, c_j)$ and repeat until no more changes can be made. This is also a commonly studied class.

Tilted Drilling Machine Instances, Additive Norm (rtilt). These instances correspond to the following potential application. One wishes to drill a collection of holes on a tilted surface, and the drill is moved using two motors. The first moves the drill to its new x -coordinate, after which the second moves it to its new y -coordinate. Because the surface is tilted, the second motor can move faster when the y -coordinate is decreasing than when it is increasing. The generator starts with an instance of `rect` and modifies it based on three parameters: u_x , the multiplier on $|\Delta x|$ that tells how much time the first motor takes, and u_y^+ and u_y^- , the multipliers on $|\Delta y|$ when the direction is up/down. For this class, the parameters $u_x = 1$, $u_y^+ = 2$, and $u_y^- = 0$ were chosen, which yields the same optimal tour lengths as the original symmetric `rect` instances because in a cycle the sum of the upward movements is precisely balanced by the sum of the downward ones. Related classes based on perturbing the Euclidean metric were proposed by Kataoko and Morito in [501] to model the problem of scheduling temperature-dependent operations in a factory where cooling was faster than heating.

Tilted Drilling Machine Instances, Sup Norm (stilt). For many drilling machines, the motors operate in parallel and so the proper metric is the maximum of the times to move in the x and y directions rather than the sum. This generator has the same three parameters as for `rtilt`, although now the distance is the maximum of $u_x|\Delta x|$ and $u_y^-|\Delta y|$ (downward motion) or $u_y^+|\Delta y|$ (upward motion). For this class, the parameters $u_x = 2$, $u_y^+ = 4$, and $u_y^- = 1$ were chosen.

Random Euclidean Stacker Crane Instances (crane). In the *Stacker Crane Problem* one is given a collection of source-destination pairs s_i, d_i in a metric space where for each pair the crane must pick up an object at location s_i and deliver it to location d_i . The goal is to order these tasks so as to minimize the time spent by the crane going between tasks, i.e., moving from the destination of one pair to the source of the next one. This can be viewed as an ATSP in which city c_i corresponds to the pair s_i, d_i and the distance from c_i to c_j is the metric distance between d_i and s_j . The generator has a single parameter $u \geq 1$, and constructs its source-destination pairs as follows. The sources are generated uniformly in the same way that cities are generated in an instance of `rect`. Then for each source s we pick two integers x and y uniformly and independently from the interval $[-10^6/u, 10^6/u]$. The destination is the vector sum $s + (x, y)$. In order to preserve a sense of geometric locality, we let u vary as a function of N , choosing values so that the expected number of other sources that are closer to a given source than its destination is roughly a constant, independent of N . For more details see [200]. These instances do not necessarily obey the triangle inequality since the time for traveling from source to destination is not counted.

Disk Drive Instances (disk). These instances attempt to capture some of the structure of the problem of scheduling the read head on a computer disk. This problem is similar to the stacker crane problem in that the files to be read have a start position and an end position in their tracks. Sources are again generated as in `rect` instances, but now destinations have the same y -coordinates as their sources. To determine the x -coordinate of a destination, we generate a random integer $x \in [0, 10^6/u]$ and add it to the x -coordinate of its source modulo 10^6 , thus capturing the fact that tracks can wrap around the disk. The distance from a destination to the next source is computed based on the assumption that the disk is spinning in the x -direction at a given rate and that the time for moving in the y direction is proportional to the distance traveled at a significantly slower rate. To get to the next source we first move to the required y -coordinate and then wait for the spinning disk to deliver the x -coordinate to us. For more details see [200].

Pay Phone Coin Collection Instances (coin). These instances model the problem of collecting money from pay phones in a grid-like city. We assume that the city is a k by k grid of city blocks with 2-way streets running between them and a 1-way street running around the exterior boundary of the city. The pay phones are uniformly distributed over the boundaries of the blocks. We can only collect from a pay phone if it is on the same side of the street as we are currently driving on, and we cannot make “U-turns” either between or at street corners. Finding the shortest route is trivial if there are so many pay phones that most blocks have one on all four of their sides. This class is thus generated by letting k grow with N , in particular as the nearest integer to $10\sqrt{N}$.

No-Wait Flowshop Instances (shop). The no-wait flowshop was the application that inspired the local search heuristic of Kanellakis and Papadimitriou [492]. In a k -processor no-wait flowshop, a job \bar{u} consists of a sequence of tasks (u_1, u_2, \dots, u_k) that must be performed by a fixed sequence of machines. The processing of u_{i+1} must start on machine $i+1$ as soon as processing of u_i is complete on machine i . This models the processing of heated materials that must not be allowed to cool down and situations where there is no storage space to hold waiting jobs. These instances have $k = 50$ processors and task lengths are independently chosen random integers between 0 and 1000. The distance from job \bar{v} to job \bar{u} is the minimum possible amount by which the finish time for u_k can exceed that for v_k if \bar{u} is the next job to be started after \bar{v} .

Approx. Shortest Common Superstring Instances (super). This class is intended to capture some of the structure in a computational biology problem relevant to genome reconstruction. Given a collection of strings C , we wish to find a short superstring S in which all are (at least approximately) contained. If we did not allow mismatches the distance from string A to string B would be the length of B minus the length of the longest prefix of B that is also a suffix of A . Here we add a penalty equal to twice the number of mismatches, and the distance from string A to string B is the length of B minus $\max\{j + 2k: \text{there is a prefix of } B \text{ of length } j \text{ that matches a suffix of } A \text{ in all but } k \text{ positions}\}$. The generator uses this metric applied to random binary strings of length 20.

Specific Instances: TSPLIB and Other Sources (realworld). This collection includes the 13 ATSP instances with 100 or more cities currently in TSPLIB plus 16 new instances from additional applications. Applications covered include stacker crane problems (the `rbg` instances),

vehicle routing (the `ftv` instances), coin collection (`big702`), robotic motion planning (`atex600`, called `atex8` in [200]), tape drive reading (the `td` instances), code optimization (the `code` instances), and table compression (the `dc` instances). We omit the TSPLIB instance `kro124p` since it is simply a perturbed random Euclidean instance, rather than from an application. For more details, see [200].

In what follows we shall consider which measurable properties of instances correlate with heuristic performance. Likely candidates include (1) the gap between the AP and HK bounds, (2) the extent to which the distance metric departs from symmetry, and (3) the extent to which it violates the triangle inequality. The specific metrics we use for these properties are as follows. For (1) we use the percentage by which the AP bound falls short of the HK bound. For (2) we use the ratio of the average value of $|d(c_i, c_j) - d(c_j, c_i)|$ to the average value of $|d(c_i, c_j) + d(c_j, c_i)|$, a quantity that is 0 for symmetric matrices and has a maximum value of 1. For (3) we first compute, for each pair c_i, c_j of distinct cities, the minimum of $d(c_i, c_j)$ and $\min\{d(c_i, c_k) + d(c_k, c_j) : 1 \leq k \leq N\}$ (call it $d'(c_i, c_j)$). The metric is then the average, over all pairs c_i, c_j , of $(d(c_i, c_j) - d'(c_i, c_j))/d(c_i, c_j)$. A value of 0 implies that the instance obeys the triangle inequality. (The latter two metrics are different and perhaps more meaningful than the ones used in [200].)

Tables 10.2 and 10.3 report the values for these metrics on our randomly generated classes and real-world instances respectively. For the random instances, average values are given for $N = 100, 316, \text{ and } 1,000$. In Table 10.2 the classes are ordered by increasing value of the HK-AP gap for the 1,000-city entry. In Table 10.3 instances of the same type are ordered by increasing gap, and types are ordered by increasing average gap. For the random instance classes, there seems to be little correlation between the three metrics, although for some there is a dependency on the number N of cities. For the `realworld` instances there are some apparent correlations between metrics for instances of the same type, e.g., for the `ftv` instances, increasing HK-AP gap seems to go with increasing asymmetry, whereas for the `dc` instances increasing HK-AP gap seems to go with *decreasing* asymmetry (and increasing triangle inequality violations).

Table 10.3 also displays the percentage by which the optimal tour length exceeds the HK bound for our `realworld` instances. (The analogous information for our random instance classes is shown in the tables of Section 4.6.) Note that for these instances the OPT-HK gap is 0.03% or less whenever the HK-AP gap is less than 1%, a first suggestion of the significance of the latter metric.

	% HK-AP			Asymmetry			Triangle		
	100	316	1,000	100	316	1,000	100	316	1,000
tmat	.34	.16	.03	.232	.189	.165	-	-	-
amat	.64	.29	.05	.333	.332	.333	.633	.752	.837
shop	.50	.22	.15	.508	.498	.515	-	-	-
disk	2.28	.71	.34	.044	.045	.046	.250	.313	.354
super	1.04	1.02	1.17	.076	.075	.075	-	-	-
crane	7.19	6.34	5.21	.061	.035	.020	.101	.087	.066
coin	15.04	13.60	13.96	.010	.007	.003	-	-	-
stilt	18.41	14.98	14.65	.329	.333	.336	-	-	-
rtilt	20.42	17.75	17.17	.496	.500	.503	-	-	-
rect	20.42	17.75	17.17	-	-	-	-	-	-
smat	19.83	20.73	19.66	-	-	-	.632	.751	.837

Table 10.2. For the 100-, 316-, and 1000-city instances of each class, the average percentage shortfall of the AP bound from the HK bound and the average asymmetry and triangle inequality metrics as defined in the text. “-” stands for .000.

Instance	N	%OPT-HK	%HK-AP	Asymmetry	Triangle
rbg323	323	.00	.00	.206	.3892
rbg358	358	.00	.00	.226	.4251
rbg403	403	.00	.00	.231	.4489
rbg443	443	.00	.00	.229	.4358
td100	101	.00	.00	.115	.0001
td316	317	.00	.00	.115	.0002
td1000	1001	.00	.00	.117	.0006
big702	702	.00	.00	.131	-
dc849	849	.00	.09	.010	-
dc563	563	.01	.33	.008	-
dc134	134	.01	.36	.005	-
dc895	895	.03	.68	.003	-
dc176	176	.02	.81	.005	-
dc112	112	.02	.87	.002	-
dc188	188	.02	1.22	.001	-
dc932	932	.01	2.48	.001	.0002
dc126	126	.01	3.78	.001	.0017
ftv170	171	1.47	3.10	.114	-
ftv150	151	.77	3.17	.115	-
ftv160	161	1.36	3.29	.114	-
ftv130	131	.89	3.61	.112	-
ftv140	141	.83	4.12	.118	-
ftv110	111	1.79	4.18	.124	-
ftv120	121	1.68	4.93	.122	-
ftv100	101	1.16	5.52	.123	-
code198	198	.00	.09	1.000	.0224
code253	253	.74	17.50	1.000	.0598
atex600	600	1.13	97.69	.118	-

Table 10.3. Metrics for realworld test instances.

3. Heuristics

In this section we briefly describe the heuristics covered in this study and their implementations. Substantially fewer implementations are covered than in the STSP chapter. This is in part because we did not announce a formal challenge covering the ATSP, but mainly because there has been far less research on this more general problem. Thus a smaller sample of implementations can still include top-performing representatives of all the effective approaches. As was the case with the STSP, these effective approaches do not currently include any representatives from the world of metaheuristics (e.g., simulated annealing, tabu search, neural nets, etc.). The best ATSP heuristic from that field that we know of is a “memetic” algorithm of [140], but it is not yet fast enough to be competitive, given the quality of tours that it produces for most of our instance classes. We cover four basic groups of heuristics: Classical tour construction heuristics derived from those for the STSP, heuristics based on solutions to the Assignment Problem (these have no effective analogue in the case of the STSP), local search heuristics and repeated local search heuristics. We unfortunately do not have space to present all the algorithmic and implementation details for the heuristics (especially the more complicated local search and repeated local search heuristics), but we provide pointers to where more details can be found.

3.1. Classical Tour Construction Heuristics

Nearest Neighbor (NN) and Greedy (Greedy). These are ATSP versions of the classical Nearest Neighbor and Greedy heuristics for the STSP. In **NN** one starts from a random city and then successively goes to the nearest as-yet-unvisited city, returning at last to the starting city to complete the tour. In **Greedy**, we view the instance as a complete directed graph with arc lengths equal to the corresponding inter-city distances. Sort the arcs in order of increasing length. Call an arc *eligible* if it can be added to the current set of chosen arcs without creating a (non-Hamiltonian) cycle or causing an in- or out-degree to exceed one. The implementation covered here works by repeatedly choosing (randomly) one of the two shortest eligible arcs until a tour is constructed. Randomization is important as the implementations we cover are both part of a Johnson-McGeoch local search code to be described below, and a common way of using the latter is to perform a set of randomized runs and output the best tour found. The running times reported for these **NN** and **Greedy** implementations may be somewhat inflated from what they would be for stand-alone codes, as they include some preprocessing steps not strictly needed for tour construction alone.

3.2. Cycle Cover Heuristics

Recall from Section 2.1 that a solution to the Assignment Problem for the distance matrix of an ATSP instance corresponds to a minimum length cover of the cities by vertex-disjoint directed simple cycles. This section covers heuristics that solve such Assignment Problems as a sub-routine. Analogous heuristics for the STSP are not particularly effective, but one might hope for better results in the case of the ATSP, especially for those instances classes with a small gap between the AP bound and the optimal tour length.

Cycle Patching, Two Largest Cycle Variant (Patch). This heuristic computes the minimum cycle cover for the full instance and then patches the cycles together into a tour using a heuristic analyzed by Karp and Steele in [500]: Repeatedly select the two cycles containing the most cities and combine them into the shortest overall cycle that can be constructed by breaking one arc in each cycle and linking together the two resulting directed paths. The running time for this heuristic is typically dominated by the time needed to construct the initial cycle cover. *Patch* provides better results than related variants, such as repeatedly patching together the two *shortest* cycles. The implementation for which we report results is due to Zhang.

Repeated Assignment (RA). This heuristic was originally studied by Frieze, Galbiati, and Maffioli in [324] and currently has the best proven worst-case performance guarantee of any polynomial-time ATSP heuristic (assuming the triangle inequality holds): Its tour lengths are at most $\log_2 N$ times the optimal tour length. This is not impressive in comparison to the $3/2$ guarantee for the Christofides heuristic for the symmetric case, but nothing better has been found in two decades. The heuristic works by constructing a minimum cycle cover and then repeating the following until a connected graph is obtained. For each connected component of the current graph, select a representative vertex. Then compute a minimum cycle cover for the subgraph induced by these chosen vertices and add that to the current graph. A connected graph will be obtained before one has constructed more than $\log_2 N$ cycle covers, and each cycle cover can be no longer than the optimal ATSP tour which is itself a cycle cover. Thus the total arc length for the connected graph is at most $\log_2 N$ times the length of the optimal tour. Note also that it must be strongly connected and all vertices must have in-degree equal to out-degree. Thus it is Eulerian, and if one constructs an Euler tour and follows it, shortcutting past any vertex previously encountered, one obtains an ATSP tour that by the triangle inequality can be no longer than the total arc length of the graph. We report on

a Johnson-McGeoch implementation that in addition uses heuristics to find good choices of representatives and performs “greedy” shortcutting as described in relation to the Christofides heuristic in Chapter 9. The combination of these measures yields substantial improvements in tour length with a negligible increase in running time, as reported in [200] (which called this enhanced version of the heuristic “RA+”).

Contract or Patch (COP). This heuristic, as proposed by Glover, Gutin, Yeo, and Zverovich in [377] and refined by Gutin and Zverovich in [425], again begins by constructing a minimum length cycle cover. We then execute the following loop:

1. While the current cycle cover contains more than a single cycle and at least one cycle with fewer than t cities (*short cycle*):
 - (1.1) Delete the longest arc in each short cycle and “contract” the resulting path into a single composite city whose out-arcs corresponds to the out-arcs of the path’s head and whose in-arcs correspond to the in-arcs of the path’s tail.
 - (1.2) Compute a minimum length cycle cover from the resulting contracted graph.
2. Recursively expand the composite cities in the final cycle cover to obtain a cycle cover for the original instance.
3. Patch the cycle cover as in **Patch**.

The dependence of the performance of COP on the value of t was studied by Gutin and Zverovich in [425], who recommended setting $t = 3$, in which case only cycles of length two are contracted. The results presented here were obtained using their implementation with this choice of t .

Zhang’s Heuristic (Zhang). Zhang’s heuristic [837] works by truncating the computations of an AP-based branch-and-bound optimization algorithm that uses depth first search as its exploration strategy. We start by computing a minimum length cycle cover M_0 and determining an initial *champion* tour by patching as in **Patch**. If this tour is no longer than M_0 (for instance if M_0 was itself a tour), we halt and return it. Otherwise, call M_0 the initial *incumbent* cycle cover, and let I_0 , the set of *included* arcs, and X_0 , the set of *excluded* arcs, be initially empty. The variant we cover in this chapter proceeds as follows.

Inductively, the incumbent cycle cover M_i is the minimum length cycle cover that contains all arcs from I_i and none of the arcs from X_i , and we assume that M_i is shorter than the current champion tour and is not itself a tour. Let $C = \{e_1, e_2, \dots, e_k\}$ be a cycle in M_i that contains a minimum number of *free arcs* (arcs not in I_i). As pointed out in [166], there are k distinct ways of breaking this cycle: We can force the deletion of e_1 , retain e_1 and force the deletion of e_2 , retain e_1 and e_2 and force

the deletion of e_3 , etc. We solve a new assignment problem for each of these possibilities that is not forbidden by the requirement that all arcs in I_i be included. In particular, for all h , $1 \leq h \leq k$, such that e_h is not in I_i , we construct a minimum cycle cover that includes all the arcs in $I_i \cup \{e_j : 1 \leq j < h\}$ and includes none of the arcs in $X_i \cup \{e_h\}$. (The exclusion of the arcs in this latter set is forced by adjusting their lengths to a value exceeding the initial champion tour length.) If we retain the data structures used in the construction of M_i each new minimum cycle cover can be computed using only one augmenting path computation.

Let us call the resulting cycle covers the *children* of M_i . Call a child *viable* if its length is less than the current champion tour. If any of the viable children is a tour and is better than the current champion, we replace the champion by the best of these (which in turn will cause the set of viable children to shrink, since now none of the other children that are tours will be viable). If at this point there is no viable child, we halt and return the best tour seen so far. Otherwise, we let the new incumbent M_{i+1} be a viable child of minimum length. We then patch M_{i+1} and if the result is better than the current champion tour, update the latter. Then we update I_i and X_i to reflect the sets of included and excluded arcs specified in the construction of M_{i+1} and continue. This process must terminate after at most N^2 phases, since each phase adds at least one new arc to $I_i \cup X_i$, and so we must eventually either construct a tour or obtain a graph in which no cycle cover is shorter than the current champion tour.

The results reported here were obtained using Zhang's implementation. In [200], which called the above heuristic "ZHANG1," several variants were also considered. The most interesting of these, ZHANG2, differs from the above in that in each phase *all* viable children are patched to tours to see if a new champion can be produced. As reported there, this variant produces marginally better tours than does ZHANG1, but at a typical cost of roughly doubling the running time.

3.3. Local Search

Local search heuristics are typically defined in terms of a *neighborhood structure*, where tour B is a neighbor of tour A if it can be obtained from A by a specific type of perturbation or *move*. The standard local search heuristic uses a tour construction heuristic to generate an initial tour and then repeatedly looks for an improving move and, if it finds one, performs it to obtain a new and better tour. This process continues until no such move exists (or none can be found by the particular search process employed by the heuristic). We cover two local search heuristics.

3-Opt (3opt). In this heuristic, the neighborhood consists of all tours that can be obtained by deleting three arcs and permuting the three resulting paths. In contrast to the STSP version of 3opt, we do not consider moves in which any of the three paths is reversed. Reversing a path potentially changes the lengths of all the arcs in the path, and hence is much more expensive to evaluate than in the symmetric case. We present results for a Johnson-McGeoch implementation of 3-Opt that generates its starting tours using NN and employs many of the speedup tricks exploited by their implementation of symmetric 3opt. For details on these, see Chapter 9 and [461].

Kanellakis-Papadimitriou (KP). This heuristic, invented by Kanellakis and Papadimitriou in [492], attempts to mimic the Lin-Kernighan heuristic for the STSP [563], subject to the constraint that it does not reverse any tour segments. (The symmetric version of Lin-Kernighan is discussed in more detail in Chapters 8 and 9.) KP consists of two alternating search processes. The first process is a variable-depth search that tries to find an improving k -opt move (breaking k arcs and permuting the resulting k segments) for some odd $k \geq 3$ by a constrained sequential search procedure modeled on that in Lin-Kernighan. The second process is a search for an improving, non-sequential “double-bridge” 4-Opt move. In such a move, if (a, b, c, d) is the original ordering of the four tour segments formed by deleting four edges, the new tour contains them in the order (b, a, d, c) , without reversals. The implementation on which we report here is that of Johnson and McGeoch described in [200] and called KP4 there. The details are too complicated to go into here. Suffice it to say that many of the same speedup tricks as used in the Johnson-McGeoch implementation of Lin-Kernighan (see Chapter 9) are incorporated, and in the 4-Opt phase the *best* 4-Opt move is found using a dynamic programming approach suggested by [375] that takes only $\Theta(N^2)$ time.

3.4. Repeated Local Search Heuristics

Each of the heuristics in the previous section can be used as the engine for an “iterated” (or “chained”) local search procedure that obtains significantly better tours, as originally proposed by [588]. (Other ways of improving on basic local search heuristics, such as the dynamic programming approach of [80], do not seem to be as effective here, although hybrids of this approach with iteration might be worth further study.) In an iterated procedure, one starts by running the basic heuristic once to obtain an initial champion tour. Then one repeats the following process some predetermined number of times:

Apply a random double-bridge 4-Opt move to the current champion to obtain a new (and presumably worse) tour. Use this new tour as the starting tour for another run of the heuristic. If the resulting tour is better than the current champion, declare it to be the new champion.

If one is using the “don’t-look bit” speedup trick of Bentley [103] in the basic local search heuristic, the fact that a double-bridge move changes the neighbors of only 8 cities can be exploited to obtain further speedups, as described in Chapter 9. All the iterated local search heuristics on which we report here use unbiased random double-bridge moves to perturb the tour, although better results might be obtainable if one biases the choice toward better moves, as is done for example in [27] for the STSP. In this chapter we present results for three iterated local search heuristics plus one additional heuristic that uses a different repetition mechanism.

Iterated 3-Opt (I3opt). A Johnson-McGeoch implementation that performs $10N$ iterations, where N is the number of cities.

Iterated Kanellakis-Papadimitriou (IKP). A Johnson-McGeoch implementation that performs N iterations. The basic local search heuristic used is a variant on KP that searches more broadly in the variable-depth search phase. This was called IKP4F in [200].

Iterated HyperOpt (Ihyper- k). This heuristic uses for its basic local search heuristic an asymmetric variant on the HyperOpt heuristic for the STSP introduced by Burke, Cowling, and Keuthen in [149] and described in Chapter 9. The HyperOpt neighborhood with parameter k is a restricted version of $2k$ -Opt. In the variant used here and described more fully in [150], this neighborhood is augmented with (non-reversing) 3-Opt moves. Although that paper talks about using a “Variable Neighborhood Search” version of the above iteration process, the results we report are for an implementation from the authors that uses the standard double-bridge perturbation and performs N iterations. There is one added detail however: whenever a new champion tour is found in an iteration, the implementation attempts to improve it using 2-Opt moves. A 2-Opt move involves the reversal of a tour segment, so as remarked above this is expensive, but it does here yield a slight improvement in tour length for the near-symmetric instances. We report on the results for Ihyper-3. The tours found by Ihyper-4 are slightly better on average, but the running times are 2–10 times longer.

Helsgaun’s Heuristic (Helsgaun). This heuristic successfully applies the same sort of trick we used to compute Held-Karp bounds and optimal tour lengths for ATSP instances using Concorde: the ATSP instance is transformed into an equivalent STSP instances and then an

STSP code is applied, in this case the repeated search version of Helsgaun's heuristic [446] as described in Chapter 9. The transformation used is somewhat simpler than the one used for *Concorde*. Here we replace each city c_i by a pair of cities c_i^+ and c_i^- , and set $d(c_i^+, c_j^-) = d(c_i, c_j)$ and $d(c_j^+, c_i^-) = d(c_j, c_i)$. In addition we set $d(c_i^-, c_i^+)$ to $-\infty$ and all the remaining distances to $+\infty$, where ∞ is a sufficiently large number that all arcs with length $-\infty$ *must* be included in the tour and none of the arcs with length $+\infty$ can be used. (This transformation doesn't work for *Concorde* because *Concorde* has trouble with large negative arc lengths and currently has no other facility for forcing arcs into the tour.)

The results we report are for version LKH-1.1 of Helsgaun's code, with N iterations, called *Helsgaun*[N] in Chapter 9. (A significant number of iterations is needed just to get all the required arcs into the tour, so one *must* use the iterated version of *Helsgaun* heuristic.) In Helsgaun's approach, the starting tours for iterations after the first are not generated by double-bridge perturbations. Instead we run a full tour construction heuristic that biases its choices based on the arcs in the best tours seen so far. Note that since the ATSP to STSP transformation constructs an instance with $2N$ cities, the actual number of iterations is $2N$.

4. Results

In this section we summarize the results for the heuristics and instance classes describe above. We begin in Section 4.1 by using the two symmetric instance classes (*rect* and *smat*) to illuminate the question of how much more powerful symmetric codes can be than asymmetric ones. This is further examined in Section 4.2, which displays the best tour quality obtainable by any of our heuristics within given time bounds and compares this with what is possible in the symmetric case.

We then consider the relative advantages of the various heuristics. Section 4.3 looks at those heuristics that appear to be consistently fast (*NN*, *Greedy*, and *3opt*) and compares their performance. Section 4.4 considers the four heuristics based on the computation of minimum cycle covers. Section 4.5 then considers the remaining local and repeated local search heuristics *KP*, *Ihyper-3*, *I3opt*, *IKP*, and *Helsgaun*.

The final three "results" sections provide more detailed comparisons between heuristics. Section 4.6 gives a class-by-class comparison, presenting tables for each of the asymmetric classes ranking all the heuristics in the study. Section 4.7 compares the heuristics according to various robustness metrics. Section 4.8 then discusses the lessons learned from

the experiments on our random instance classes and sees how well they apply to our testbed of real world instances.

4.1. Symmetric Instances and Codes

In this section we illustrate our claim that ATSP heuristics are in a sense less powerful than STSP heuristics. We have already seen that they are more limited, for instance by the fact that moves involving the reversal of a tour segment are typically too expensive to evaluate. We see how this translates into performance by looking at pairs of related ATSP/STSP heuristics and comparing their results on our two classes of symmetric instances (`rect` and `smat`). See Table 10.4.

The only heuristic that appears to be as powerful in its asymmetric form as in its symmetric form is Nearest Neighbor (NN). This is not surprising, given that for any given starting city it will construct the same tour in either case. Differences in running time come from the differences in input representation. In contrast, our other tour construction heuristic, the Greedy heuristic, completely falls apart in the asymmetric case. This is because two paths that will eventually be merged by the symmetric Greedy heuristic may be constructed with inconsistent orientations by the asymmetric Greedy heuristic and hence be unmergeable.

The results for the remaining local search and repeated local search pairs all show an advantage in tour quality for the symmetric members of each pair, reflecting the richer neighborhoods that they search. Note that this is true in the KP versus Lin-Kernighan comparison, even though the KP implementations includes a double-bridge 4-Opt phase absent from the LK implementation. Note also that the ATSP variants on repeated local search pay a substantial running time penalty (far greater than the simple difference attributable to the larger size of the asymmetric instance representation).

4.2. Asymmetric State of the Art

In this section we further address the differences in the states of the art for the STSP and ATSP by examining what can be accomplished within given time thresholds, assuming we can choose the most appropriate heuristic for each instance class and number of cities. See Table 10.5, which for $N = 316, 1,000,$ and $3,162$ and appropriate time thresholds reports the best average excess over the HK bound obtainable in normalized time less than that threshold and the heuristic that obtains it. For space reasons, we do not list results for our 100-city instances. We note, however, that for all nine classes Helsgaun's average running time on 100-city instances is 2 seconds or less and it never averages more

rect

Heuristic	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
NN-S	26.39	27.35	26.07	26.68	.00	.01	.0	0
NN-A	26.39	27.51	26.11	26.55	.04	.26	1.9	22
Greedy-S	18.72	19.35	17.91	16.46	.01	.01	.0	0
Greedy-A	153.78	292.21	547.94	1005.68	.04	.27	1.9	22
3opt-S	2.81	2.86	2.88	3.25	.04	.12	.2	0
3opt-A	7.89	8.30	8.37	8.76	.22	1.82	5.9	22
I3opt-S	.86	.94	1.11	1.29	.41	1.34	4.1	18
I3opt-A	2.02	2.57	2.66	3.09	.44	2.19	11.4	124
LK-S	1.32	1.55	1.77	1.92	.05	.14	.3	0
KP-A	5.11	5.06	5.00	5.17	.04	.31	2.0	23
I3opt-S	.86	.94	1.11	1.29	.41	1.34	4.1	18
I3opt-A	2.02	2.57	2.66	3.09	.44	2.19	11.4	124
Hgaun-S	.68	.67	.69	.62	.47	4.67	55.0	887
Hgaun-A	.68	1.00	1.62	2.35	2.00	21.50	247.3	6244

smat

Heuristic	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
NN-S	139.28	180.75	235.51	298.05	.02	.13	1.0	12
NN-A	139.28	181.18	233.10	307.11	.03	.26	1.9	21
Greedy-S	103.48	136.78	177.17	213.90	.02	.16	1.0	12
Greedy-A	653.13	1842.80	5396.52	12073.47	.03	.31	1.9	21
3opt-S	11.38	19.18	31.29	46.59	.11	1.03	3.1	12
3opt-A	32.36	44.31	61.69	85.58	.19	1.76	5.8	21
I3opt-S	1.21	3.18	6.34	10.83	.45	1.84	7.8	58
I3opt-A	8.43	15.57	25.55	39.84	.50	2.63	14.0	121
LK-S	1.72	2.22	3.43	4.68	.12	.97	4.2	12
KP-A	10.59	11.50	13.13	15.31	.04	.36	2.2	26
Hgaun-S	.10	.04	.01	.00	.27	1.94	13.8	302
Hgaun-A	.88	2.16	3.78	5.72	1.40	16.45	180.9	4756

Table 10.4. Tour quality and normalized times for corresponding symmetric (-S) and asymmetric (-A) codes. Hgaun is an abbreviation for Helsgaun. LK is the Johnson-McGeoch implementation of (symmetric) Lin-Kernighan covered in Chapter 9.

than .1% above optimal. More detailed results on the tradeoffs for each class can be found in the tables of Section 4.6.

Note that almost all our heuristics are represented in the table for some combination of class and N . Only Greedy, RA, and Ihyper-3 are missing, and the latter would have made it into the table for rtilt and $N \geq 1,000$ had we chosen slightly different thresholds. Thus depending on the application and instance size, almost any of the heuristics we cover might be useful.

The tradeoffs between running time and tour quality are not as good as in the case of the STSP, however. We don't have the space here for a detailed comparison, but let us simply consider the behavior of the STSP heuristic Helsgaun[.1N] as discussed in Chapter 9. For the 1,000-city instances of all four classes discussed in that chapter, Helsgaun[.1N] averages within 1.05% of the HK bound in 12 seconds or less (normalized to the same target machine used in this chapter). Here that level of behavior can't be attained in 2 minutes for four of the classes and 10 minutes for three. For the 3,162-city instances, Helsgaun[.1N] averages within .87% of the HK bound in less than 2 minutes for each of the four STSP classes, whereas here that level of behavior cannot be attained in 3 hours for four of our nine classes. It would thus seem that at least some potential ATSP applications may be more difficult to handle than the current applications of the STSP. We will return to this question when we consider "real-world" instances in Section 4.8.

4.3. Consistently Fast Heuristics

In this section we consider our options if we need to obtain tours very quickly, i.e., within a small multiple of the time to read the instance. In the case of geometric instances of the STSP, this restriction was quite severe and only allowed us to use heuristics that did little more than sort. For the general ATSP, with its $\Theta(N^2)$ instance size, much more sophisticated heuristics satisfy this restriction, at least asymptotically: not only tour construction heuristics like NN and Greedy, but even simple local optimization heuristics like 3opt. Table 10.6 presents the results for these three heuristics on our nine truly asymmetric instance classes, the classes ordered by increasing values of the gap between AP and HK bounds. In comparison, the time simply to read and store the distance matrix using standard C input/output routines is .02–.03 (normalized) seconds for $N = 100$, .14–.18 seconds for $N = 316$, .9–1.2 seconds for $N = 1,000$, and 10–14 seconds for $N = 3,162$, depending on the class. Observe that NN and Greedy typically use at most 2.5 times the read time, and although 3opt has higher multiples for small N , it appears to

316 Cities

Class	< .33 Seconds		< 2 Seconds		< 10 seconds		< 30 seconds	
tmat	.01	Zhang	.01	Zhang	.01	Zhang	.00	Helsgaun
amat	3.15	COP	.16	Zhang	.08	Helsgaun	.08	Helsgaun
shop	14.65	NN	.08	Zhang	.08	Zhang	.02	Helsgaun
disk	1.13	COP	.27	Zhang	.06	Helsgaun	.06	Helsgaun
super	1.20	COP	.15	IKP	.04	Helsgaun	.04	Helsgaun
crane	4.45	KP	4.29	Zhang	1.79	IKP	1.30	Helsgaun
coin	6.59	KP	6.59	KP	2.99	IKP	1.47	Helsgaun
stilt	22.79	Patch	8.79	KP	3.95	I3opt	2.33	Helsgaun
rtilt	18.91	Patch	18.91	Patch	7.35	KP	2.76	Helsgaun

1000 Cities

Class	< 5 Seconds		< 30 Seconds		< 2 Minutes		< 10 Minutes	
tmat	.00	Zhang	.00	Zhang	.00	Zhang	.00	Zhang
amat	2.66	COP	1.29	IKP	.03	Helsgaun	.03	Helsgaun
shop	13.29	NN	.03	Helsgaun	.03	Helsgaun	.01	Helsgaun
disk	.88	Patch	.02	Zhang	.01	Helsgaun	.01	Helsgaun
super	1.22	COP	.21	Zhang	.05	Helsgaun	.05	Helsgaun
crane	4.78	KP	1.27	IKP	1.27	IKP	1.02	Helsgaun
coin	6.15	KP	2.66	IKP	2.66	IKP	1.61	Helsgaun
stilt	8.80	KP	4.31	I3opt	4.31	I3opt	2.61	Helsgaun
rtilt	18.38	Patch	8.33	KP	8.33	KP	1.50	Helsgaun

3,162 Cities

Class	< 1 minute		< 5 minutes		< 30 Minutes		< 3 hours	
tmat	.00	Zhang	.00	Zhang	.00	Zhang	.00	Zhang
amat	1.01	COP	.04	Zhang	.04	Zhang	.03	Helsgaun
shop	10.88	3opt	.24	Patch	.01	Zhang	.01	Zhang
disk	25.64	3opt	.01	Zhang	.01	Zhang	.01	Helsgaun
super	1.88	I3opt	.52	IKP	.43	Zhang	.13	Helsgaun
crane	4.26	KP	1.36	IKP	1.36	IKP	.92	Helsgaun
coin	6.34	KP	2.87	IKP	2.87	IKP	2.16	Helsgaun
stilt	8.15	KP	4.28	I3opt	4.28	I3opt	3.39	Helsgaun
rtilt	19.83	3opt	9.68	KP	9.68	KP	2.76	Helsgaun

Table 10.5. Best average percentage above the HK bound obtained within various normalized running time thresholds for the 316-, 1,000-, and 3162-city instances of the nine asymmetric classes.

be settling asymptotically to a similar multiple. Also, for none of the three heuristics does running time vary markedly from class to class.

As for tour quality, observe first that the **Greedy** heuristic continues to perform poorly, being significantly worse than **NN** on all but three classes (**tmat** and **super**, where it is better, and **crane**, where it is roughly equivalent). Moreover, on four classes (**amat**, **disk**, **stilt**, and **rtilt**) **Greedy** might be said to self-destruct, as the ratio of its tour length to the HK bound grows rapidly as N increases, whereas **NN** only self-destructs on two of these and actually improves as N gets larger on the other two (**stilt** and **rtilt**). Consequently, although symmetric **Greedy** typically outperforms symmetric **NN**, their roles are reversed in the asymmetric case. This is why the Johnson-McGeoch local search implementations for the ATSP use **NN** to generate their starting tours rather than **Greedy**, which was the default for their STSP implementations. Although for three of the asymmetric instances classes (**tmat**, **super**, and **crane** again) **Greedy** manages to provide more effective (although not necessarily shorter) starting tours than **NN**, it is not sufficiently robust to be used as a general-purpose starting tour generator.

Consider now **3opt**. Note that the ratio of its overall running time to that for its tour generator **NN** declines from a factor of 6 or more when $N \leq 316$ to little more than a factor of 1 when $N = 3,162$. On the other hand, the amount by which its average tour length improves over its **NN** starting tour also declines as N increases for all but two of the classes, those being the two classes where **NN** self-destructs (**amat** and **disk**). The improvements over **NN** do however remain significant, even for $N = 3,162$, although the amount of significance varies with class. **3opt** would thus be the heuristic of choice among these three unless high speed on small instances is a requirement. Indeed, as we shall see, it might well be a best overall choice in many situations, given that it averages within 27% of the HK bound in less than 30 seconds for all classes except the totally unstructured instances of **amat**.

Let us now consider the extent to which these results correlate with the abstract instance metrics in Tables 10.2 and 10.3. A first observation is that for none of the three heuristics is there much of a correlation with the gap between AP and HK bounds, which is less than 2% for the first five classes in the table and greater than 13% for the last three. Such correlations will not be evident until we consider the heuristics of the next section. On the other hand, the two classes where both **NN** and **Greedy** self-destruct are those with the biggest triangle inequality violations (**amat** and **disk**), and **Greedy** self-destructs on all but one of the four classes with the highest asymmetry metrics (**amat**, **shop**, **stilt**, and **rtilt**). Whether such possible correlations are meaningful can only

Greedy

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	31.23	29.04	26.53	26.25	.03	.26	1.7	20
amat	243.09	362.86	418.56	695.29	.04	.27	1.9	21
shop	49.34	56.07	61.55	66.29	.03	.26	2.1	40
disk	188.82	307.14	625.76	1171.62	.03	.28	2.7	23
super	6.03	5.40	5.16	5.79	.03	.22	1.5	18
crane	41.86	44.09	39.70	41.60	.03	.27	1.9	21
coin	48.73	46.76	42.33	35.94	.04	.24	1.7	20
stilt	106.25	143.89	178.34	215.84	.04	.28	1.9	23
rtilt	350.12	705.56	1290.63	2350.38	.03	.28	2.0	23

NN

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	38.20	37.10	37.55	36.66	.03	.24	1.7	20
amat	195.23	253.97	318.79	384.90	.03	.26	1.9	21
shop	16.97	14.65	13.29	11.87	.03	.23	2.5	20
disk	96.24	102.54	115.51	161.99	.04	.27	1.9	23
super	8.57	8.98	9.75	10.62	.03	.21	1.5	18
crane	40.72	41.66	43.88	43.18	.03	.26	1.9	21
coin	26.08	26.71	26.80	25.60	.03	.23	1.7	20
stilt	30.31	30.56	27.62	24.79	.03	.30	1.9	22
rtilt	28.47	28.28	27.52	24.60	.04	.26	1.9	22

3opt

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	6.44	9.59	12.66	16.20	.19	1.71	5.5	20
amat	39.23	58.57	83.77	112.08	.19	1.75	5.8	21
shop	3.02	7.25	10.22	10.88	.23	1.78	5.6	21
disk	12.11	16.96	20.85	25.64	.19	1.82	6.1	23
super	3.12	4.30	5.90	7.94	.15	1.43	4.8	18
crane	9.48	9.41	10.65	10.64	.19	1.76	7.3	22
coin	8.06	9.39	9.86	9.92	.18	1.62	5.3	20
stilt	11.39	12.65	12.62	12.27	.19	1.80	8.2	22
rtilt	10.04	13.09	18.00	19.83	.19	2.05	6.6	23

Table 10.6. Tour quality and normalized running times for fast heuristics.

be determined once we have a more extensive set of instance classes to study.

4.4. Cycle Cover Heuristics

Table 10.7 presents the results for three of the four heuristics of Section 3.2: the simple Cycle Patching heuristic (**Patch**), the Contractor-Patch heuristic (**COP**), and Zhang's heuristic (**Zhang**). We omit the Repeated Assignment heuristic (**RA**) as for every class it finds worse tours and takes more time than **Patch**, despite the fact that it is the only polynomial-time heuristic known to have a reasonable worst-case guarantee for instances obeying the triangle inequality. (Its domination by **Patch** is shown in the class-based summary tables of Section 4.6.)

Note that for all three heuristics in the table, there is a strong correlation between good average tour length and the gap between the HK and AP bounds. (Once again, the classes in the table are ordered by increasing gap.) The only exceptions are the **amat** class, whose lack of structure appears to cause trouble for **Patch** and **COP** on the smaller instances, and the $N = 100$ entry for **disk**. Note, however, that for $N = 100$ **disk** has a bigger average gap than **super**. All three heuristics find better tours than does **3opt** on the first five classes and the larger instances of the sixth, while **Patch** and **COP** are worse than **3opt** on the last three classes. **Zhang**, although better than **3opt** on one of the last three classes (**rtilt**), has a running time that is over 400 times longer.

The running times for all three cycle cover heuristics are highly dependent on instance class. For **PATCH** and to a lesser extent **COP**, this primarily reflects the fact that the Assignment Problem code used as a subroutine has substantially larger running times for some instance classes than others. See Table 10.8, which gives the normalized running times for the Assignment Problem codes used by **COP** and by **Patch** (and by **Zhang**). Note first that the times for the latter code represents a major proportion of the times reported for **Patch** in Table 10.7.

The speeds of both AP codes vary substantially with instance class, and although the first code tends to be faster, the two are slowest on the same instance classes: **shop**, **disk**, and **rtilt**, with the times for the 3162-city instances of **shop** being from 18 to 87 times slower than those for **coin**, depending on the subroutine used. As to running time growth rates, note that if the rate for a class were merely quadratic, then the average times would go up roughly by a factor of 10 from one instance size to the next, while if the rate were cubic (the worst-case bound) it would go up by factors of about 32. The times on the **shop** and **disk** classes for both codes are thus consistent with a cubic growth

Patch

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	.84	.64	.17	.00	.03	.22	1.8	29
amat	10.95	6.50	2.66	1.88	.03	.22	1.9	18
shop	1.15	.59	.39	.24	.04	.48	8.4	260
disk	9.40	2.35	.88	.30	.03	.26	2.9	75
super	1.86	2.84	3.99	6.22	.02	.19	1.7	29
crane	9.40	10.18	9.45	8.24	.03	.21	1.5	23
coin	16.48	16.97	17.45	18.20	.02	.18	1.4	17
stilt	23.33	22.79	23.18	24.41	.03	.24	2.2	29
rtilt	17.03	18.91	18.38	19.39	.03	.28	2.9	54

COP

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	.57	.36	.16	.00	.01	.12	.7	15
amat	9.31	3.15	2.66	1.01	.01	.15	.6	26
shop	.68	.36	.19	.10	.08	1.41	29.1	1152
disk	6.00	1.13	.51	.15	.03	.31	8.7	297
super	1.01	1.20	1.22	2.06	.03	.24	4.6	243
crane	10.32	9.08	7.28	6.21	.04	.44	3.5	53
coin	16.44	17.68	16.23	16.06	.02	.10	1.2	22
stilt	22.48	23.31	22.80	22.90	.07	.94	8.1	105
rtilt	19.62	22.86	20.95	20.37	.05	.33	5.6	117

Zhang

Class	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	.06	.01	.00	.00	.03	.27	2.5	30
amat	.97	.16	.04	.04	.04	.47	7.6	296
shop	.20	.08	.03	.01	.06	1.02	19.6	460
disk	1.51	.27	.02	.01	.05	.56	6.4	105
super	.27	.17	.21	.43	.04	.61	20.4	995
crane	4.36	4.29	4.05	4.10	.07	1.96	66.7	3176
coin	8.20	11.03	11.14	11.42	.10	3.82	168.4	9610
stilt	10.75	13.99	12.66	12.86	.11	4.11	163.7	4184
rtilt	9.82	12.20	11.81	11.45	.13	4.37	178.0	9594

Table 10.7. Tour quality and normalized running times for Cycle Cover Heuristics.

Class	COP AP Code				Patch and Zhang AP Code			
	100	316	1000	3162	100	316	1000	3162
tmat	.00	.05	.6	12	.03	.22	1.8	26
amat	.01	.06	.5	9	.03	.22	1.8	17
shop	.02	.37	8.1	262	.03	.47	8.3	251
disk	.00	.08	1.7	63	.03	.26	2.8	72
super	.00	.03	.5	9	.02	.18	1.7	26
crane	.02	.09	.8	12	.03	.21	1.5	19
coin	.00	.02	.1	3	.02	.17	1.2	14
stilt	.01	.26	2.0	27	.03	.23	2.0	26
rtilt	.01	.15	2.1	43	.03	.27	2.7	49

Table 10.8. Normalized running times for reading an instance and solving the corresponding Assignment Problem with the AP codes used by COP, Patch, and Zhang.

rate, whereas the times for **amat**, **crane**, **coin**, and **stilt** are much closer to quadratic, at least in the case of the AP code used by Patch.

These growth rates should be compared to those for **3opt** in Table 10.6, which seems to be subquadratic, at least within this range. (Asymptotically it must be at least quadratic, because it needs to read the instance, but within this range reading time is not the dominant running time component.) Thus, although Patch is faster than **3opt** for all classes when $N \leq 316$, it begins losing ground thereafter. It is significantly slower on **shop** when $N = 1,000$, and on five additional classes when $N = 3,162$.

When we turn to COP and Zhang, additional factors affect running times. Most importantly, these heuristics may call the AP code more than once, and the number of calls also varies among instance classes, adding further to overall running time variability. For Patch the ratio between slowest and fastest class running times at 3,162 cities is about 15, whereas for COP it is 77 and for Zhang it is 320.

Because of its extra AP calls and despite its often-faster AP code, COP is typically slower than Patch. The only exceptions are those classes where the initial cycle cover typically has no 2-cycles, as appears to be the case for **tmat**. In such cases COP is just a version of PATCH with different tie-breaking rules. COP does, however, get better results than PATCH on all but one of the classes (**rtilt**), although there isn't much room for improvements on Patch's results for **tmat**, **shop**, and the larger instances of **disk**.

Zhang finds better tours than Patch and COP for all classes. (In the first case this is unsurprising, since the first step of Zhang is to perform Patch.) Where there is room for substantial improvements, Zhang's

improvements are comparatively larger (although still not enough to beat 3opt on the `coin` and `stilt` instances). Zhang's better tour quality often comes at a significant running time cost, however. For example, Zhang is over 100 times slower than COP on the larger `coin` instances. Its running time growth rates look worse than quadratic for all classes except possibly `tmat`, and worse even than cubic for `amat`, `super`, `crane`, `coin`, and `rtilt`. Only two of these five (`amat` and `super`) have small HK-AP gaps, however, and in neither of these two cases is the time at $N = 3,162$ nearly as bad as for the last four classes. Moreover, for the `shop` and larger `disk` instances Zhang is actually faster than COP. Indeed, for the five classes with small HK-AP gaps and instances with 1,000 or fewer cities, Zhang gets its good results while never averaging more than 20.4 normalized seconds. Thus assuming one is not dealing with larger instances and one has small gaps, Zhang might well be the heuristic of choice. For instance classes that have larger HK-AP gaps, however, there are better alternatives, as we shall see in the next section.

4.5. Local and Repeated Local Search Heuristics

Table 10.9 presents the results for the Johnson-McGeoch implementation of the Kanellakis-Papadimitriou heuristic (KP) and iterated versions of two simpler local search heuristics: the Johnson-McGeoch implementation of Iterated 3-Opt (I3opt) and the Burke, Cowling, and Keuthen implementation of their Iterated HyperOpt heuristic (Ihyper-3). All three implementations perform 3-Opt as part of their search, so it is no surprise that they find better tours than does 3opt for all classes. For KP, this often comes with only a small running time penalty. For five of the classes (`amat`, `super`, `crane`, and `stilt`), its running time is within 50% or less of that for 3opt.

Comparisons to the cycle cover heuristics are more complicated, but reflect the correlation for the latter between good behavior and small HK-AP gaps. For the first four classes, KP is outperformed by the faster Patch heuristic and the comparably fast COP heuristic on all but the 100-city instances of `amat` and `disk`, where KP finds significantly better tours, although not ones as good as those found by Zhang. Interestingly, for all four classes KP's tours get progressively worse as N increases, while those of the cycle cover heuristics get progressively better. On the other hand, for the last three classes KP finds much better tours than Zhang in much less time, with the time advantage growing dramatically with N . For `crane` it finds tours that are almost as good as Zhang's with a time advantage that grows to a factor of over 100 for $N = 3,162$.

KP

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	1.41	2.23	3.09	4.03	.11	.64	4.2	49
amat	5.82	6.95	8.99	11.51	.04	.36	2.3	27
shop	1.57	2.92	3.88	4.54	2.08	17.46	118.2	1005
disk	2.99	3.81	5.81	9.17	.05	.55	8.2	324
super	1.05	1.29	1.59	2.10	.04	.24	1.7	19
crane	4.58	4.45	4.78	4.26	.05	.32	2.0	22
coin	5.74	6.59	6.15	6.34	.04	.27	1.8	20
stilt	8.57	8.79	8.80	8.15	.09	.56	3.0	31
rtilt	6.06	7.35	8.33	9.68	.27	2.15	14.9	133

I3opt

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	.30	.85	1.63	2.25	1.02	3.93	15.5	98
amat	5.10	13.27	27.12	45.53	.50	2.59	14.3	116
shop	.49	4.02	9.23	10.76	9.90	37.73	102.9	401
disk	.97	2.32	3.89	5.29	.49	3.14	24.1	293
super	.28	.61	1.06	1.88	.40	1.77	7.8	59
crane	1.98	2.27	1.95	2.12	.45	2.22	12.1	115
coin	2.98	3.37	3.48	3.83	.43	2.06	10.9	110
stilt	3.29	3.95	4.32	4.28	.92	5.21	29.1	256
rtilt	1.69	4.22	12.97	18.41	3.08	24.22	98.2	412

Ihyper-3

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
tmat	1.29	2.71	4.96	7.29	2.82	15.45	62.7	1048
amat	15.29	34.74	66.66	103.10	1.93	15.78	81.9	529
shop	.46	1.85	5.05	9.36	10.07	75.38	300.3	1077
disk	2.02	6.53	10.52	14.77	2.02	21.95	166.5	1812
super	.62	1.53	2.81	4.69	1.35	9.09	47.6	741
crane	1.83	2.46	2.66	3.28	1.58	11.37	75.8	920
coin	2.22	3.09	3.83	4.60	1.44	9.99	62.6	675
stilt	3.20	4.32	5.67	6.45	2.38	18.35	91.0	869
rtilt	1.31	2.16	4.63	8.10	4.33	42.26	300.2	2531

Table 10.9. Tour quality and normalized running times for KP, I3opt, and Ihyper-3.

I3opt is slower than KP for all classes except `disk`. For most classes its running time is about 10 times slower for 100 cities, but the ratio declines as N increases. On the other hand, its tours for classes `crane`, `coin`, and `stilt` are the best we have seen so far, and its tours for `super` have only been bested by Zhang which took more than 10 times as long. It does perform relatively poorly, however, on the four classes with smallest HK-AP gap and on `rtilt`.

Turning to `Ihyper-3`, we see substantially greater running times than for either KP or I3opt. For the four classes with smallest HK-AP gaps, it is also significantly slower than Zhang and produces worse results. For the remaining classes it is outperformed by I3opt except for the `rtilt` class and the 100-city `crane` and `stilt` instances, where it produces the best results seen so far.

Table 10.10 presents results for the two most sophisticated repeated local search codes in this study: The Johnson-McGeoch implementation of

IKP

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
<code>tmat</code>	.09	.14	.41	.65	5.91	31.21	189.1	1934
<code>amat</code>	.56	.74	1.29	2.43	.34	3.19	29.6	488
<code>shop</code>	.49	2.36	3.66	4.55	214.48	1585.19	7449.1	31738
<code>disk</code>	.56	.48	.96	1.77	.43	12.57	479.4	26277
<code>super</code>	.13	.15	.28	.52	.17	1.33	10.2	138
<code>crane</code>	1.46	1.79	1.27	1.36	.44	3.52	23.5	297
<code>coin</code>	2.71	2.99	2.66	2.87	.35	2.35	16.9	231
<code>stilt</code>	3.00	3.54	3.96	4.13	4.78	89.67	1577.0	30916
<code>rtilt</code>	1.80	4.12	7.29	8.89	25.93	657.80	7420.4	61939

Helsgaun

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
<code>tmat</code>	.03	.00	.00	.00	1.00	10.22	91.8	1634
<code>amat</code>	.29	.08	.03	.03	.80	7.89	75.6	1802
<code>shop</code>	.05	.02	.01	.02	.87	20.17	483.0	13671
<code>disk</code>	.24	.06	.01	.01	1.00	9.39	93.9	1914
<code>super</code>	.06	.04	.05	.13	1.07	7.67	77.3	1828
<code>crane</code>	1.21	1.30	1.02	.92	1.20	14.89	162.2	4040
<code>coin</code>	1.15	1.47	1.61	2.16	1.40	17.45	205.4	4975
<code>stilt</code>	1.95	2.33	2.61	3.39	1.67	18.67	221.0	5863
<code>rtilt</code>	.69	.99	1.50	2.76	2.00	23.45	318.2	8574

Table 10.10. Tour quality and normalized running times for IKP and Helsgaun.

Iterated Kanellakis-Papadimitriou (IKP) and Helsgaun’s Helsgaun variant on Lin-Kernighan.

Note that IKP finds better tours than does I3opt for all classes, but at much greater running time cost. It performs poorly in the same places, however. For the four classes with smallest HK-AP gap Zhang finds better tours in less time. For the intermediate class *super*, the two heuristics are roughly equal in tour quality while IKP takes twice as long (on the same machine). On the problematic *rtilt* class, IKP is marginally beaten by *Ihyper-3*, which takes much less time.

The final heuristic Helsgaun has the best average tour length for all classes and all *N* (except the 3162-city *shop* instance, where it was beaten by Zhang, which was 0.1% over HK as opposed to Helsgaun’s 0.2%). Helsgaun also has substantial running times, although ones that are exceeded by those for Zhang on two classes and by IKP on four. This is clearly the general-purpose algorithm of choice when time is not an issue or when instances are small.

As a final point of comparison, Table 10.11 presents results for the “heuristic” that computes the optimal tour length using Concorde’s STSP optimization code via our ATSP-to-STSP transformation, with Zhang or Helsgaun run to provide an initial upper bound. Note that for the four classes with smallest HK-AP gap, the running time for optimization, if exponential, is not yet showing strong evidence of that behavior, although the data suggests such behavior for *coin*. Indeed, optimization is actually faster than IKP for all sizes of classes *tmat* and *shop* and for the larger sizes of *disk*. It also beats Helsgaun on the

Optimization via Concorde

Heur	Percent above HK				Normalized Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
<i>tmat</i>	.03	.00	.00	.00	4.14	11.50	38.0	568
<i>amat</i>	.29	.08	.02	.01	16.60	109.00	399.0	40317
<i>shop</i>	.05	.02	.01	.00	54.80	315.00	1039.0	20234
<i>disk</i>	.24	.06	.01	.01	15.20	40.00	160.0	1176
<i>super</i>	.05	.03	.01	–	4.30	33.60	1629.0	–
<i>crane</i>	1.21	1.30	–	–	117.00	7081.00	–	–
<i>coin</i>	1.05	1.36	–	–	193.00	83285.00	–	–
<i>stilt</i>	1.86	–	–	–	1119.00	–	–	–
<i>rtilt</i>	.68	.67	–	–	82.00	1734.00	–	–

Table 10.11. Results for optimization via Concorde. A “–” entry indicates that optimization was not feasible via this approach. Running times include the time for running Zhang (first four classes) or Helsgaun (last five) to obtain an initial upper bound, but not the (relatively small) time for the ATSP-to-STSP transformation.

larger `tmat` instances, and is even faster than computing the HK bound for these and the largest `disk` instance. Average times should not be trusted, however, especially for the last five classes, where running times are far more variable than were the times for our heuristics. The times for the 100-city `stilt` instances ranged from 5 seconds to 3500 seconds. The gap between the optimal tour length and the HK bound for a given class and value of N was relatively consistent, however. Note that the optimal solution value is quite close to the HK bound for classes with small HK-AP gap, but somewhat farther away when the gap is larger.

4.6. Results Organized by Instance Class

In the last three “Results” sections, we more directly address the question of which heuristics one should use in practice. Here we consider what one should do if one knows a great deal about the application in question, as we now do in the case of our nine random asymmetric instance classes. The choice of what to use of course depends on the relative importance one places on tour quality and running time. In Tables 10.12 through 10.16 we present for each of those nine classes the average tour quality and normalized running times for all the heuristics in this study. This expands on the more restricted class-based results of Section 4.2 by showing the full range of quality/time tradeoffs for each class. We include the previously-ignored results for the Greedy and Repeated Assignment heuristics and data on the HK-AP and Opt-HK gaps and the time for computing the associated bounds. The AP bounds were computed using the AP code of `Patch`. All codes except `COP` and `Ihyper-3` were run on the same 196 Mhz MIPS R10000 processors, and running time comparisons among all but those two are thus unaffected by normalization errors (although the actual values presented may be affected by their translation to the target machine).

In order to fit the tables to the width of the page, we use the abbreviations “`Hgaun`” for `Helsgaun` and “`Ihyper`” for `Ihyper-3`. The identities of the algorithms behind the names are all explained in Section 3. For each class we have sorted the heuristics by their average excess over the HK bound on 3,162-city instances. Note that the ordering would often have been significantly different had we sorted on the 100-city results. Beyond these remarks, we will let the results speak for themselves.

4.7. Robustness of the Heuristics

In this section we consider the question of which heuristics to choose for general purpose ATSP tour generation in cases where one may not yet know much about the instance structure. For such uses, one would

tmat

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
NN	38.20	37.10	37.55	36.66	.03	.24	1.7	20
Greedy	31.23	29.04	26.53	26.25	.03	.26	1.7	20
3opt	6.44	9.59	12.66	16.20	.19	1.71	5.5	20
Ihyper	1.29	2.71	4.96	7.29	2.82	15.45	62.7	1048
KP	1.41	2.23	3.09	4.03	.11	.64	4.2	49
I3opt	.30	.85	1.63	2.25	1.02	3.93	15.5	98
IKP	.09	.14	.41	.65	5.91	31.21	189.1	1934
RA	4.88	3.10	1.55	.46	.06	.63	7.8	278
Patch	.84	.64	.17	.00	.03	.22	1.8	29
COP	.57	.36	.16	.00	.01	.12	.7	15
Zhang	.06	.01	.00	.00	.03	.27	2.5	30
Hgaun	.03	.00	.00	.00	1.00	10.22	91.8	1634
OPT	.03	.00	.00	.00	4.14	11.50	38.0	568
HK	.00	.00	.00	.00	.77	4.46	43.8	968
AP	-.34	-.16	-.03	.00	.03	.22	1.8	26

amat

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	243.09	362.86	418.56	695.29	.04	.27	1.9	21
NN	195.23	253.97	318.79	384.90	.03	.26	1.9	21
RA	86.60	100.61	110.38	134.14	.07	.69	10.2	265
3opt	39.23	58.57	83.77	112.08	.19	1.75	5.8	21
Ihyper	15.29	34.74	66.66	103.10	1.93	15.78	81.9	529
I3opt	5.10	13.27	27.12	45.53	.50	2.59	14.3	116
KP	5.82	6.95	8.99	11.51	.04	.36	2.3	27
IKP	.56	.74	1.29	2.43	.34	3.19	29.6	488
Patch	10.95	6.50	2.66	1.88	.03	.22	1.9	18
COP	9.31	3.15	2.66	1.01	.01	.15	.6	26
Zhang	.97	.16	.04	.04	.04	.47	7.6	296
Hgaun	.29	.08	.03	.03	.80	7.89	75.6	1802
OPT	.29	.08	.02	.01	16.60	109.00	399.0	40317
HK	.00	.00	.00	.00	.70	4.33	36.7	703
AP	-.65	-.29	-.04	-.04	.03	.22	1.8	17

Table 10.12. Tour quality and normalized running times for classes amat (Random Asymmetric Matrices) and tmat (Random Asymmetric Matrices closed under shortest paths).

shop

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	49.34	56.07	61.55	66.29	.03	.26	2.1	40
NN	16.97	14.65	13.29	11.87	.03	.23	2.5	20
3opt	3.02	7.25	10.22	10.88	.23	1.78	5.6	21
I3opt	.49	4.02	9.23	10.76	9.90	37.73	102.9	401
Ihyper	.46	1.85	5.05	9.36	10.07	75.38	300.4	1077
IKP	.49	2.36	3.66	4.55	214.48	1585.19	7449.1	31738
KP	1.57	2.92	3.88	4.54	2.08	17.46	118.2	1005
RA	4.77	2.77	1.69	1.05	.10	1.68	28.7	1103
Patch	1.15	.59	.39	.24	.04	.48	8.4	260
COP	.68	.36	.19	.10	.08	1.41	29.1	1152
Hgaun	.05	.02	.01	.02	.87	20.17	483.0	13671
Zhang	.20	.08	.03	.01	.06	1.02	19.6	460
OPT	.05	.02	.01	.00	54.80	315.00	1039.0	20234
HK	.00	.00	.00	.00	1.53	10.12	104.3	1581
AP	-.50	-.22	-.15	-.07	.03	.47	8.3	251

disk

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	188.82	307.14	625.76	1171.62	.03	.28	2.7	23
NN	96.24	102.54	115.51	161.99	.04	.27	1.9	23
3opt	12.11	16.96	20.85	25.64	.19	1.82	6.1	23
RA	86.12	58.27	42.45	25.32	.07	.92	18.9	658
Ihyper	2.02	6.53	10.52	14.77	2.02	21.95	166.5	1812
KP	2.99	3.81	5.81	9.17	.05	.55	8.2	324
I3opt	.97	2.32	3.89	5.29	.49	3.14	24.1	293
IKP	.56	.48	.96	1.77	.43	12.57	479.4	26277
Patch	9.40	2.35	.88	.30	.03	.26	2.9	75
COP	6.00	1.13	.51	.15	.03	.31	8.7	297
Zhang	1.51	.27	.02	.01	.05	.56	6.4	105
Hgaun	.24	.06	.01	.01	1.00	9.39	93.9	1914
OPT	.24	.06	.01	.01	15.20	40.00	160.0	1176
HK	.00	.00	.00	.00	.85	4.84	53.7	1929
AP	-2.28	-.71	-.34	-.11	.03	.26	2.8	72

Table 10.13. Tour quality and normalized running times for classes shop (50 Processor No-Wait Flowshop Scheduling) and disk (Random Disk Head Motion Scheduling).

super

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	48.73	46.76	42.33	35.94	.04	.24	1.7	20
NN	8.57	8.98	9.75	10.62	.03	.21	1.5	18
RA	4.24	5.22	6.59	8.34	.01	.47	2.1	167
3opt	3.12	4.30	5.90	7.94	.15	1.43	4.8	18
Patch	1.86	2.84	3.99	6.22	.02	.19	1.7	29
Ihyper	.62	1.53	2.81	4.69	1.36	9.09	47.6	741
KP	1.05	1.29	1.59	2.10	.04	.24	1.7	19
COP	1.01	1.20	1.22	2.06	.03	.24	4.6	243
I3opt	.28	.61	1.06	1.88	.40	1.77	7.8	59
IKP	.13	.15	.28	.52	.17	1.33	10.2	138
Zhang	.27	.17	.21	.43	.04	.61	20.4	995
Hgaun	.06	.04	.05	.13	1.07	7.67	77.3	1828
OPT	.05	.03	.01	-	4.30	33.60	1629.0	-
HK	.00	.00	.00	.00	.68	4.52	40.4	900
AP	-1.04	-1.02	-1.17	-1.61	.02	.18	1.7	26

crane

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
RA	40.80	50.33	53.60	54.91	.07	.67	7.0	251
NN	40.72	41.66	43.88	43.18	.03	.26	1.9	21
Greedy	41.86	44.09	39.70	41.60	.03	.27	1.9	21
3opt	9.48	9.41	10.65	10.64	.19	1.76	7.3	22
Patch	9.40	10.18	9.45	8.24	.03	.21	1.5	23
COP	10.32	9.08	7.28	6.21	.04	.44	3.5	53
KP	4.58	4.45	4.78	4.26	.05	.32	2.0	22
Zhang	4.36	4.29	4.05	4.10	.07	1.96	66.7	3176
Ihyper	1.83	2.46	2.66	3.28	1.58	11.37	75.8	920
I3opt	1.98	2.27	1.95	2.12	.45	2.22	12.1	115
IKP	1.46	1.79	1.27	1.36	.44	3.52	23.5	297
Hgaun	1.21	1.30	1.02	.92	1.20	14.89	162.2	4040
OPT	1.21	1.30	-	-	117.00	7081.00	-	-
HK	.00	.00	.00	.00	.95	7.11	357.4	669
AP	-7.19	-6.34	-5.21	-4.43	.03	.21	1.5	19

Table 10.14. Tour quality and normalized running times for classes **super** (Approximate Shortest Common Superstring) and **crane** (Stacker-Crane Scheduling).

coin

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
RA	52.74	64.95	68.78	71.20	.06	.56	6.3	140
Greedy	48.73	46.76	42.33	35.94	.04	.24	1.7	20
NN	26.08	26.71	26.80	25.60	.03	.23	1.7	20
Patch	16.48	16.97	17.45	18.20	.02	.18	1.4	17
COP	16.44	17.68	16.23	16.06	.02	.10	1.2	22
Zhang	8.20	11.03	11.14	11.42	.10	3.82	168.4	9610
3opt	8.06	9.39	9.86	9.92	.18	1.62	5.3	20
KP	5.74	6.59	6.15	6.34	.04	.27	1.8	20
Ihyper	2.22	3.09	3.83	4.60	1.43	9.99	62.5	674
I3opt	2.98	3.37	3.48	3.83	.43	2.06	10.9	110
IKP	2.71	2.99	2.66	2.87	.35	2.35	16.9	231
Hgaun	1.15	1.47	1.61	2.16	1.40	17.45	205.4	4975
OPT	1.05	1.36	–	–	193.00	83285	–	–
HK	.00	.00	.00	.00	1.11	7.64	54.8	394
AP	-15.04	-13.60	-13.96	-13.09	.02	.17	1.2	14

stilt

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	106.25	143.89	178.34	215.84	.04	.28	1.9	23
RA	55.79	62.76	65.03	71.48	.07	.61	8.5	241
NN	30.31	30.56	27.62	24.42	.03	.30	1.9	48
Patch	23.33	22.79	23.18	24.41	.03	.24	2.2	29
COP	22.48	23.31	22.80	22.90	.07	.94	8.1	105
Zhang	10.75	13.99	12.66	12.86	.11	4.11	163.7	4184
3opt	11.39	12.65	12.62	12.27	.19	1.80	8.2	22
KP	8.57	8.79	8.80	8.15	.09	.56	3.0	31
Ihyper	3.20	4.32	5.67	6.45	2.38	18.35	91.0	870
I3opt	3.29	3.95	4.32	4.28	.92	5.21	29.1	256
IKP	3.00	3.54	3.96	4.13	4.78	89.67	1577.0	30916
Hgaun	1.95	2.33	2.61	3.39	1.67	18.67	221.0	5863
OPT	1.86	–	–	–	1119.00	–	–	–
HK	.00	.00	.00	.00	1.24	8.63	67.4	368
AP	-18.41	-14.98	-14.65	-14.04	.03	.23	2.0	26

Table 10.15. Tour quality and normalized running times for classes coin (Coinbox Collection Routing) and stilt (Tilted Supnorm Routing).

rtilt

Heur	Percent above HK				Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
Greedy	350.12	705.56	1290.63	2350.38	.03	.28	2.0	23
RA	61.95	73.47	78.27	82.03	.08	1.06	19.3	607
NN	28.47	28.28	27.52	24.60	.04	.26	1.9	22
COP	19.62	22.86	20.95	20.37	.05	.33	5.6	117
3opt	10.04	13.09	18.00	19.83	.19	2.05	6.6	23
Patch	17.03	18.91	18.38	19.39	.03	.28	2.9	54
I3opt	1.69	4.22	12.97	18.41	3.08	24.22	98.2	412
Zhang	9.82	12.20	11.81	11.45	.13	4.37	178.0	9594
KP	6.06	7.35	8.33	9.68	.27	2.15	14.9	133
IKP	1.80	4.12	7.29	8.89	25.93	657.80	7420.4	61939
Ihyper	1.31	2.16	4.63	8.10	4.32	42.26	300.2	2531
Hgaun	.69	.99	1.50	2.76	2.00	23.45	318.2	8574
OPT	.68	.67	.68	-	82.00	1734.00	-	-
HK	.00	.00	.00	.00	1.37	8.88	87.5	373
AP	-20.42	-17.75	-17.17	-16.84	.03	.27	2.7	49

Table 10.16. Tour quality and normalized running times for class *rtilt* (Tilted Rectilinear Routing). Optima for 1000-city instances were computed by symmetric code applied to equivalent *rect*.

want a heuristic that was relatively robust, producing good results in reasonable time for a wide variety of instance types, the meaning of “good” and “reasonable” of course depending on the situation. As a first attempt at providing such advice, we propose using the following empirical metric.

In Table 10.17 we summarize the worst average results (both excess over the HK bound and normalized running time) obtained over three different subsets of the classes, with heuristics ordered by their tour quality for 3,162-city instances. The first subset consists of all of the nine asymmetric instance classes except the Random Distance Matrix class *amat*. (We omit *amat* because such structureless instances are unlikely to arise in any real world application, and if one wants to study them for some mathematical reason, we already have provided detailed information about which heuristics to use in the previous section.) This is the robustness criterion that might apply if one knows nothing about the instance(s) for which tours are desired. The second and third subsets yield robustness criteria that might be relevant if we at least knew something about the HK-AP gaps for the instances in question. The second subset consists of the four classes (other than *amat*) for which the average HK-AP gap is less than 2% for all values of *N* we cover, and

the third consists of the remaining four classes, for which the gaps range from 4% to over 20%.

The table omits the **Greedy** and **NN** tour construction heuristics because both are dominated with respect to the robustness metrics by **3opt**. We also omit the cycle cover heuristic **RA**, which is dominated for all instance classes by **Patch**.

A first remark is that many of the values in the table would improve if we could simply delete the classes **shop** and **rtilt** from the subsets, as these tend to cause the most trouble for many of the heuristics. However, as the results stand, certain heuristics do jump out as good performers. **Helsgaun** is of course the best tour generator if running time is not an issue or N is small. **Zhang** and **Patch** both might offer appealing trade-offs for instances with small HK-AP gaps when less time is available, as would **KP** for instances with large gaps. Two other contenders are **COP** for small-gap instances and **Ihyper-3** for large-gap instances, but the first is perhaps not sufficiently faster than **Zhang** to justify its poorer performance, and the second is not sufficiently faster than **Helsgaun**.

In the next section we shall see how well this advice serves us, by applying **Patch**, **Zhang**, **KP**, and **Helsgaun** to our suite of **realworld** instances.

4.8. Performance on Real-World Instances

In Table 10.18 we present the excesses over optimal and the normalized running times for the codes **Patch**, **KP**, **Zhang**, and **Helsgaun** on our testbed of real world instances. Within groups, the instances are ordered by increasing value of HK-AP gap, and groups are ordered by increasing *average* HK-AP gap. Note that here we are presenting the excess over the *optimal* tour length, not the HK bound, since the former has been successfully computed for all these instances (often in time less than that required by **Helsgaun**). All codes were run on the same machine so normalization errors do not affect running time comparisons. However, since instance sizes do not align precisely with the sizes for which our benchmark normalization runs were performed we settled for a single normalization factor of 0.5. This is a reasonable compromise between the actual factors for the 196 Mhz MIPS R10000 to 500 Mhz Alpha normalization, which were .6667, .5556, .3863, and .4262 for $N = 100, 316, 1,000,$ and $3,162$ respectively.

Note that the **Zhang** does much better than might have been expected based on our results for random instance classes, getting within 1% of optimal for all but one instance (**atex600**), for which it is still only 2.6% over optimal. Moreover, it never takes more than a minute on

Worst Results over All Asymmetric Classes except amat

Heur	Percent over HK				Normalized Time in Seconds			
	100	316	1000	3162	100	316	1000	3162
3opt	12.11	16.96	20.85	25.64	.23	2.05	8.2	23
Patch	23.33	22.79	23.18	24.41	.04	.48	8.4	260
COP	22.48	23.31	22.80	22.90	.08	1.41	29.1	1152
I3opt	3.29	4.22	12.97	18.41	9.90	37.73	102.9	412
Ihyper-3	3.20	6.53	10.52	14.77	10.07	75.38	300.3	2531
Zhang	10.75	13.99	12.66	12.86	.13	4.37	178.0	9610
KP	8.57	8.79	8.80	9.68	2.08	17.46	118.2	1005
IKP	3.00	4.12	7.29	8.89	214.48	1585.19	7449.1	61939
Helsgaun	1.95	2.33	2.61	3.39	2.00	23.45	483.0	13671

Worst Results over {tmat, shop, disk, super}

3opt	12.11	16.96	20.85	25.64	.23	2.05	8.2	23
Ihyper-3	2.02	6.53	10.52	14.77	10.07	75.38	300.3	1812
I3opt	.97	4.02	9.23	10.76	9.90	37.73	102.9	401
KP	2.99	3.81	5.81	9.17	2.08	17.46	118.2	1005
Patch	9.40	2.84	3.99	6.22	.04	.48	8.4	260
IKP	.56	2.36	3.66	4.55	214.48	1585.19	7449.1	31738
COP	6.00	1.20	1.22	2.06	.08	1.41	29.1	1152
Zhang	1.51	.27	.21	.43	.06	1.02	20.4	995
Helsgaun	.24	.06	.05	.13	1.07	20.17	483.0	13671

Worst Results over {crane, coin, stilt, and rtilt}

Patch	23.33	22.79	23.18	24.41	.03	.28	2.9	54
COP	22.48	23.31	22.80	22.90	.07	.94	8.1	117
3opt	11.39	13.09	18.00	19.83	.19	2.05	8.2	23
I3opt	3.29	4.22	12.97	18.41	3.08	24.22	98.2	412
Zhang	10.75	13.99	12.66	12.86	.13	4.37	178.0	9610
KP	8.57	8.79	8.80	9.68	.27	2.15	14.9	133
IKP	3.00	4.12	7.29	8.89	25.93	657.80	7420.4	61939
Ihyper-3	3.20	4.32	5.67	8.10	4.33	42.26	300.2	2531
Helsgaun	1.95	2.33	2.61	3.39	2.00	23.45	318.2	8574

Table 10.17. Robustness metrics for the heuristics in this study.

Instance	% Excess over Optimal				Normalized Running Time				
	P	KP	Z	H	P	KP	Z	H	OPT
rbg323	.00	.78	.00	.00	.22	3.71	.22	46.5	35.5
rbg358	.00	1.50	.00	.00	.27	3.33	.27	120.5	19.5
rbg403	.00	.22	.00	.00	.48	9.00	.48	221.5	22.9
rbg443	.00	.11	.00	.00	.53	11.74	.52	164.0	20.6
td100	.00	.00	.00	.00	.02	.20	.02	.5	4.1
td1000	.00	.01	.00	.00	1.87	7.29	1.87	140.5	370.5
td316	.00	.00	.00	.11	.20	3.87	.21	24.5	123.5
big702	.00	2.10	.00	.41	.98	6.04	.97	119.0	149.6
dc849	.04	.62	.00	.23	2.67	114.80	26.33	2180.0	378.3
dc563	.39	.79	.09	.12	1.49	111.95	10.23	2231.5	1449.7
dc134	.25	.57	.18	.02	.04	13.43	.14	6.5	63.7
dc895	.55	.60	.42	.25	4.74	144.43	56.54	22077.0	35926.1
dc176	.81	.67	.09	.49	.07	20.48	.19	105.0	195.1
dc112	.26	.39	.13	.28	.03	15.47	.11	4.5	76.2
dc188	.57	.59	.23	.13	.08	12.98	.19	28.5	122.8
dc932	.30	.26	.13	.26	4.98	119.17	43.86	72373.0	14722.4
dc126	.94	.65	.21	.54	.03	22.69	.15	5.5	48.3
ftv170	1.38	4.44	.36	.00	.06	.09	.10	2.5	66.3
ftv150	2.57	4.43	.00	.00	.04	.07	.05	1.0	17.4
ftv160	.60	5.89	.11	.00	.04	.07	.07	2.0	40.7
ftv130	3.64	2.16	.00	.00	.03	.06	.07	1.0	26.5
ftv140	2.77	3.15	.00	.00	.03	.06	.05	1.5	29.5
ftv110	3.32	4.04	.00	.00	.02	.04	.04	1.0	24.9
ftv120	3.09	3.12	.00	.00	.02	.05	.05	1.0	33.5
ftv100	2.69	3.11	.00	.00	.02	.04	.03	1.0	22.6
code198	.00	.00	.00	.00	.05	.54	.06	6.5	9.0
code253	3.52	.10	.28	.28	.09	1.09	.23	24.5	22.6
atex600	34.94	4.25	2.60	.82	.54	3.38	24.03	123.0	-

Table 10.18. Results for realworld instances. Patch, Zhang, and Helsgaun are abbreviated by P, Z, and H respectively. The running time for OPT includes both the time to obtain an initial upper bound using Zhang and the time to run Concorde using its default settings on the transformed instance. The relatively small time needed to perform the ATSP to STSP transformation is not included. Instance atex600 could not be optimized in reasonable time using Concorde's default settings.

any instance. This is in contrast to Helsgaun which, although it is within 1% of optimum for all the instances, has gigantic running times for many of them (over 20 hours for dc932). Of our two choices for fast heuristics, Patch is by far the faster overall, often a factor of 10 faster than KP, and provides comparable results, with the exception of atex600, where it is almost 35% over optimal compared to 4.25% for KP. Moreover, Patch is itself within 1% of optimal for all the instances through the dc class, and is often significantly faster than Zhang. (The two have roughly equivalent running times for the first eight instances,

with **Zhang** occasionally appearing slightly faster because of fluctuations in running times from run to run.)

This suggests various hybrid approaches. For example, one could use **Patch** unless the solution gets too far above the AP bound, in which case **KP** could be run as a backup. For the current testbed, this strategy would always get within 4.25% of optimal and never take more than 8 normalized seconds per instance. An analogous **Zhang/Helsgaun** combination would always get within 1% in no more than about 2 minutes.

5. Conclusions and Further Research

In this chapter we have evaluated implementations of a broad range of heuristics for the ATSP, including the best ones currently available. We have seen wide varieties of behavior (in tour quality and/or running time) for the same heuristic, depending on instance class, and we have reached some tentative conclusions about what strategies to try when confronting a real-world application, depending on the trade-off one is willing to make between tour quality and running time.

Our conclusions should be viewed only as preliminary, however. First, we do not yet know how typical are the random instance classes and real-world instances in the testbeds covered in this study. As we have seen, there can be large performance differences depending on instance structure, and we cannot claim to have contemplated all likely applications. Based on the results of Section 4.8, one might suppose that the real world is actually somewhat easier than the hardest of our random instance classes (e.g. **shop** and **rtilt**) assuming one chooses the appropriate heuristic. However, there is no guarantee this will always be true. Conversely, we have been using codes with their default settings, and it may well be that better performance (e.g., faster times for the same tour quality) might be obtained by general or class-specific tweaking.

One definite challenge for the future concerns large instances. Here we set the bound at 3,162 cities, because of the memory constraints for storing instances using the $\Theta(N^2)$ full distance matrix representation. For many applications (including the ones behind most of our classes), there is actually an application-specific linear-space representation for instances, and there might well be much larger instances for which tours are needed. Given that essentially all the heuristics we studied here have running time growth rates of $\Theta(N^2)$, $\Theta(N^3)$, or worse, new algorithmic ideas may well be needed if we are to deal effectively with such situations.

Acknowledgment. The research of Gregory Gutin was supported in part by an EPSRC grant. The research of Anders Yeo was supported in part by the grant “Research Activities in Discrete Mathematics” from

the Danish Natural Science Research Council. The research of Weixiong Zhang was supported in part by NSF grants #IRI-9619554, #IIS-0196057, and #E1A-0113618 and by DARPA cooperative agreements F30602-00-2-0531 and F33615-01-C-1897. The authors thank Pablo Moscato for helpful comments on an early draft of the chapter.

Chapter 11

POLYNOMIALLY SOLVABLE CASES OF THE TSP

Santosh N. Kabadi

Faculty of Administration

University of New Brunswick

Fredericton, New Brunswick

Canada E3B 5A3

kabadi@unb.ca

This chapter is dedicated to the memory of my parents
Mr. Narayan D Kabadi and Mrs Sushilabai N Kabadi.

1. Introduction

In this chapter we use the permutation definition of TSP introduced in Chapter 1. Given a cost matrix C of size n , the Traveling Salesman Problem (TSP) is thus to find a tour γ on $N = \{1, 2, \dots, n\}$ such that $c(\gamma) = \sum_{i \in N} c_{i, \gamma(i)}$ is minimum. We allow some non-diagonal elements of C to be ∞ . For $E = \{(i, j) : c_{ij} \text{ is finite}\}$, $G = (N, E)$ is called the digraph associated with C and we call C *compatible* with G . If C is symmetric, then we define G as an undirected graph. As in previous chapters, TSP can be equivalently stated as the problem of finding a tour \mathcal{H} in G such that $c(\mathcal{H}) = \sum_{(i,j) \in \mathcal{H}} c_{ij}$ is minimum. We denote this problem by $TSP(G, C)$. When G is a complete digraph or graph (or equivalently, all the non-diagonal entries of C are finite), we denote the problem by $TSP(C)$.

2. Constant TSP and its generalizations

We start with a very special subclass of the TSP called *constant-TSP*. An instance $TSP(G, C)$ is called a *constant-TSP* if and only if all the

tours in G have the same cost with respect to C . We call a matrix C with finite non-diagonal entries a *constant tour matrix* (CT-matrix) if and only if $TSP(C)$ is a constant-TSP.

A complete characterization of matrices C for which $TSP(G, C)$ is constant-TSP is an open problem. For any digraph $G = (N, E)$ and arbitrary values $\{a_i, b_i : i \in N\}$, if C is defined as $c_{ij} = a_i + b_j$ for all $(i, j) \in E$, and $c_{ij} = \infty$ for all $(i, j) \notin E$, then $TSP(G, C)$ is obviously a constant-TSP. Gabovich [341] proved that the reverse implication holds when G is complete. He attributes the special case of this result for the case of symmetric cost matrices to Rublineckii [733] and Leontev [557]. Several other independent proofs of this result have since then been reported [176, 361, 489, 558]. An independent proof for only the symmetric case is also reported in [524].

Theorem 1 *A cost matrix C is a CT-matrix if and only if there exist $\{a_i, b_i : i \in N\}$ such that $c_{ij} = a_i + b_j$ for all $i, j, i \neq j$. If C is symmetric, then it is a CT-matrix if and only if there exist $\{a_i : i \in N\}$ such that $c_{ij} = a_i + a_j$ for all $i, j, i \neq j$.*

Proof. The sufficiency part of the theorem is easy to prove.

The following simple proof of necessity of the condition is taken from [489]. Let C be a CT-matrix and let G be a complete digraph on node set N . Define $p = c_{1,2} - c_{1,n} - c_{n,2}$; $a_i = c_{i,n} + p/2$ and $b_i = c_{n,i} + p/2$ for $i = 1, \dots, n-1$; and $a_n = b_n = -p/2$. When the matrix C is symmetric, we have $a_i = b_i$ for all i . Define matrix C' as $c'_{ij} = c_{ij} - a_i - b_j$ for all i, j . Then $c'_{1,2} = 0$ and for all $1 \leq i < n$, $c'_{i,n} = c'_{n,i} = 0$. Let G' be the directed subgraph of G obtained by deleting node n and let C'' be obtained from C' by deleting its n th row and column. Then every Hamiltonian path in G' has the same cost, say u , with respect to the cost matrix C'' . Let γ be an arbitrary tour in G' . For each $i \in N - \{n\}$, γ defines a unique Hamiltonian path in G' from $\gamma(i)$ to i with total cost $c''(\gamma) - c''_{i,\gamma(i)} = u$. Hence $c''_{i,\gamma(i)} = u/(n-2)$ for all $i \in N - \{n\}$. Since the tour γ was selected arbitrarily, it follows that each non-diagonal element of C'' has value $u/(n-2)$. Since $c''_{1,2} = 0$, this implies that each non-diagonal element of C'' , and therefore each non-diagonal element of C' , has a value of 0. This proves the theorem. ■

The above result can be equivalently stated in terms of the dimension of the TSP polytope of a complete digraph (graph). This is discussed in Chapter 2.

Suppose $G = (N, E)$ is the undirected tour $(1, 2, \dots, n, 1)$. Then $TSP(G, C)$ is a constant-TSP for any symmetric matrix C compatible with G . It was observed by Krynski [524] that when n is an even integer, there exist symmetric matrices C compatible with G for which

there do not exist $\{a_i : i \in N\}$ such that $c_{ij} = a_i + a_j$ for all $(i, j) \in E$. Thus, the above characterization of CT-matrices does not extend to matrices compatible with an arbitrary graph or digraph. We call a digraph $G = (N, E)$ a *constant tour digraph (CT-digraph)* if and only if whenever $TSP(G, C)$ is a constant-TSP, there exist $\{a_i, b_i : i \in N\}$ such that for all $(i, j) \in E$, $c_{ij} = a_i + b_j$. An undirected graph $G = (N, E)$ is called CT-graph if and only if for any symmetric matrix C compatible with G such that $TSP(G, C)$ is a constant-TSP, there exist $\{a_i : i \in N\}$ such that for all $(i, j) \in E$, $c_{ij} = a_i + a_j$. A complete characterization of CT-graphs and CT-digraphs is an open problem. Note that this problem is different from the problem, mentioned earlier, of characterizing matrices C for which $TSP(G, C)$ is constant TSP. We give below results on subclasses of CT-graphs and CT-digraphs reported in [489].

Let (V_1, V_2) be a pair of disjoint, finite sets. For any two digraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *join* of G_1 and G_2 is the digraph $G_1 + G_2 = (V_1 \cup V_2, E')$ where $E' = E_1 \cup E_2 \cup \{(i, j), (j, i) : i \in V_1, j \in V_2\}$. For any two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *join* of G_1 and G_2 is the undirected graph $G_1 + G_2 = (V_1 \cup V_2, E')$, where $E' = E_1 \cup E_2 \cup \{(i, j) : i \in V_1, j \in V_2\}$. If $|V_2| = 1$ then we call $G_1 + G_2$ the *1-extension* of G_1 [489].

The only properties of the digraph G that are used in the proof of Theorem 1 are : (i) $\{(i, n), (n, i)\} \subseteq E$ for all $i \in \{1, \dots, n - 1\}$ and (ii) the digraph G' obtained from G by deleting the node n is strongly Hamiltonian (that is, every arc in G' lies on some tour). The same argument can therefore be used to prove Theorem 2 below.

Theorem 2 [489] *The 1-extension of a strongly Hamiltonian digraph is a CT-digraph.*

For the undirected case, we can get the following slightly stronger result.

Theorem 3 [489] *The 1-extension of a Hamiltonian graph is a CT-graph.*

It may be noted that for $n \geq 3$, the complete digraph (graph) on node set N is 1-extension of the complete digraph (graph) on node set $N - \{n\}$. Theorem 1 is thus a simple corollary of theorems 2 and 3.

For two bipartite digraphs (graphs) $G_1 = (S_1 \cup T_1, E_1)$ and $G_2 = (S_2 \cup T_2, E_2)$ on disjoint node sets, *B-join* of G_1 and G_2 , denoted by $G_1 +_B G_2$, is the bipartite digraph (graph) with node set $(S_1 \cup S_2) \cup (T_1 \cup T_2)$ and arc set (edge set) $E_1 \cup E_2 \cup \{(i, j), (j, i) : i \in S_1, j \in T_2\} \cup \{(i, j), (j, i) : i \in S_2, j \in T_1\} \cup \{(i, j) : i \in S_1; j \in T_2\} \cup \{(i, j) : i \in S_2; j \in T_1\}$. If

G_2 is the complete digraph (graph) on two nodes then we call $G_1 +_B G_2$ the 1-1 extension of G_1 .

Theorem 4 [489] *The 1 – 1 extension of a strongly Hamiltonian bipartite digraph is a CT-digraph. The 1 – 1 extension of a Hamiltonian bipartite graph is a CT-graph.*

Since the complete bipartite digraph (graph) $\vec{k}(n, n)$ ($K(n, n)$) is the 1 – 1 extension of $\vec{k}(n - 1, n - 1)$ ($K(n - 1, n - 1)$), it follows from Theorem 4 that $\vec{k}(n, n)$ ($K(n, n)$) is a CT-digraph (graph) for all n .

The following additional classes of CT-graphs and digraphs are identified in [489].

Theorem 5 [489] *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two digraphs (graphs) on disjoint node sets. Then under each of the following conditions, $G_1 + G_2$ is a CT-digraph (graph).*

- (i) G_1 and G_2 have at least one arc (edge) each and $3 \leq |V_1| = |V_2|$.
- (ii) $2 \leq |V_1| < |V_2|$, $|E_1| \neq \emptyset$ and G_2 is Hamiltonian.
- (iii) $3 \leq |V_1| < |V_2|$, $|E_1| \neq \emptyset$, and G_2 has k node-disjoint simple paths which cover all the nodes in V_2 for some $k < |V_1| - 1$.

At first sight, it may seem that a necessary condition for an undirected graph G to be a CT-graph would be that every edge in G lies in some tour in G [524]. A counter-example to this can be produced using the following observation.

Observation 6 [489] *Let $G = (S \cup T, E)$ be a CT-graph. Then $G' = (S \cup T, E \cup \{(u, v)\})$ is a CT-graph, where u, v either both belong to S or they both belong to T .*

For additional results on subclasses of CT-digraphs (graphs) as well as interesting classes of digraphs (graphs) that are not CT-digraphs (graphs), the reader is referred to [489].

The significance of the class constant-TSP in the study of polynomially solvable cases of the TSP lies in the fact that if $TSP(G, C')$ is a constant-TSP, then for any other matrix C compatible with G and for arbitrary tours γ and ψ in G , $c(\gamma) - c(\psi) = c'(\gamma) - c'(\psi)$, where $C'' = C + C'$. For any cost matrix C with associated digraph G , let us define the equivalence class

$$\text{const}(C) = \{C + C' : C' \text{ is compatible with } G \text{ and } TSP(G, C') \text{ is a constant-TSP}\}.$$

We conjecture that the class of polynomially solvable cases of TSP is closed with respect to this equivalence relation, (that is, if $TSP(C)$ is polynomially solvable then $TSP(\bar{C})$ is polynomially solvable for any $\bar{C} \in const(C)$).

We define below the concept of *density matrix* of a matrix with finite entries¹, which is used extensively throughout this chapter.

Definition 7 : For any $n \times n$ matrix A with finite entries (including the diagonal entries), the density matrix D of A is an $(n - 1) \times (n - 1)$ matrix defined as

$$d_{ij} = a_{i,j+1} + a_{i+1,j} - a_{ij} - a_{i+1,j+1} \quad \forall 1 \leq i, j < n.$$

For example, if $A = \begin{bmatrix} 4 & 5 & 4 \\ 5 & 3 & 6 \\ 4 & 1 & 2 \end{bmatrix}$ then its density matrix is

$$D = \begin{bmatrix} 3 & -4 \\ 1 & 2 \end{bmatrix}.$$

Observation 8 [51, 479, 483] Two $n \times n$ matrices A and B have the same density matrix if and only if there exist $\{u_i, v_i : i \in N\}$ such that $a_{ij} = b_{ij} + u_i + v_j$ for all i, j .

For any finite matrix C , let us define the equivalence class $dens(C)$ as the set of all matrices with the same density matrix as C . From Observation 8, it follows that $dens(C) \subseteq const(C)$. As we shall see later, many of the known polynomially solvable classes of TSP are closed under this stronger equivalence relation.

We end this section with an interesting generalization of Theorem 1 given in [785]. We call a matrix C with finite non-diagonal elements a *bi-constant tour matrix* (*BCT-matrix*) if and only if $|\{c(\gamma) : \gamma \text{ is a tour on } N\}| \leq 2$.

Theorem 9 [785] A matrix C is a BCT-matrix if and only if $C \in const(C')$, where non-diagonal entries of C' have at the most two distinct values, say 0 and a , and either all non-diagonal entries with value a lie in a single row or column, or there are only two non-diagonal entries of value a and these are of the type c'_{ij} and c'_{ji} .

A different characterization of BCT-matrices with an elementary proof is reported in [488].

¹Though diagonal elements of the cost matrix C do not play any role in the definition of TSP, strangely, many of the algorithms for solvable cases of TSP (for example the Gilmore-Gomory algorithm) require diagonal elements to be finite and satisfy specific properties.

3. The Gilmore-Gomory TSP (GG-TSP)

The Gilmore-Gomory TSP [360] is one of the most celebrated polynomially solvable cases of the TSP. Besides being one of the first known, non-trivial polynomially solvable cases, it is also the first such case with significant real-world applications. A very simple polynomial time algorithm for this case, with a simple proof of its validity, is given by Ball et al [83] and we shall discuss it in Section 5. The algorithm that Gilmore and Gomory developed for the problem is however more efficient and it is also non-trivial with a fairly non-trivial proof of its validity. We discuss this algorithm in this section and in Section 4 we show that a slight generalization of this algorithm produces an optimal solution to a fairly large subclass of the TSP.

Gilmore and Gomory [360] considered the following case. A set of n given jobs are to be heat-treated in a furnace and only one job can be treated in the furnace at a time. The treatment of the i th job involves introducing it into the furnace at a given temperature a_i and heating/cooling it in the furnace to a given temperature b_i . The costs of heating and cooling the furnace are given by functions $f(\cdot)$ and $g(\cdot)$, respectively. Thus, for any u, v in \mathbb{R} , $u < v$, the cost of heating the furnace from temperature u to temperature v is $\int_u^v f(x)dx$, while the cost of cooling the furnace from v to u is $\int_u^v g(x)dx$. Gilmore and Gomory impose the realistic condition that

$$\text{for any } x \in \mathbb{R}, \quad f(x) + g(x) \geq 0.$$

For each ordered pair (i, j) of jobs, if we decide to heat-treat job j immediately after job i , then the furnace temperature has to be changed from b_i to a_j . This cost, which we call the change-over cost and denote by c_{ij} , is given by

$$c_{ij} = \begin{cases} \int_{b_i}^{a_j} f(x)dx & \text{if } b_i \leq a_j \\ \int_{a_j}^{b_i} g(x)dx & \text{if } a_j < b_i. \end{cases}$$

Starting with the furnace temperature of a_1 and processing job 1 first, we want to sequentially heat-treat all the jobs and end by returning the furnace temperature to a_1 . The problem is to decide the order in which the jobs should be treated in the furnace so as to minimize the total change-over cost.

Gilmore and Gomory point out that if the starting temperature of the furnace is some other temperature a_0 and after processing all the jobs we want the ending temperature of the furnace to be say b_0 , then the problem can be converted to the above case by introducing an $(n + 1)$ th job with $a_{n+1} = b_0$ and $b_{n+1} = a_0$.

3.1. Gilmore-Gomory scheme for GG-TSP

Let us now discuss the Gilmore-Gomory patching algorithm [360] (with minor modifications) for their special case of TSP.

We associate with any permutation π on N a digraph $G_\pi = (N, E_\pi)$, where $E_\pi = \{(i, \pi(i)) : i \in N\}$. Let G_1, G_2, \dots, G_r be the connected components of G_π on node sets N_1, N_2, \dots, N_r , respectively. Then each G_i defines a subtour \mathfrak{C}_i on the node set N_i . We call $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_r$ the *subtours of π* . If $r = 1$ then π defines a *tour* on N and we call such a permutation a *tour*. If $|N_i| > 1$ then the subtour \mathfrak{C}_i is called a *non-trivial subtours of π* . A permutation with a single non-trivial subtour is called a *circuit*. A circuit with its only non-trivial subtour of the form (i, j, i) is called a *transposition* and is denoted by α_{ij} . A transposition of the form $\alpha_{i, i+1} = \alpha_{i+1, i}$ is called an *adjacent transposition* and is denoted by β_i . We denote by ξ the identity permutation, (that is, $\xi(i) = i$ for all i in N). For any two permutations π and ψ on N , we define $\pi \circ \psi$, *product of π with ψ* , as $\pi \circ \psi(i) = \pi(\psi(i))$ for all $i \in N$.

For any cost matrix C and any two permutations π and ψ on N , we define the *permuted cost matrix* $C^{\pi, \psi}$ as

$$c_{ij}^{\pi, \psi} = c_{\pi(i), \psi(j)} \quad \forall i, j.$$

We denote $C^{\xi, \psi}$ by C^ψ . Thus, $c(\pi \circ \psi) = c^\pi(\psi) = \sum_{i \in N} c_{i, \psi(i)}^\pi$.

Definition 10 Suppose the digraph G_π associated with a permutation π has m connected components on node sets N_1, N_2, \dots, N_m . Then $G_p^\pi = (N_p^\pi, E_p^\pi)$, the *patching pseudograph of π* , is defined as $N_p^\pi = \{1, \dots, m\}$, and $E_p^\pi = \{e_i = (u, v) : i \in \{1, 2, \dots, n-1\}, i \in N_u, (i+1) \in N_v\}$. For any $S \subseteq \{1, 2, \dots, n-1\}$ we denote by $E_p^\pi[S]$ the set $\{e_i \in E_p^\pi : i \in S\}$.

It may be noted that the edges $\{e_1, e_2, \dots, e_{n-1}\}$, when traversed in that order, form an Eulerian trail in G_p^π . Figure 11.1 shows the patching pseudograph for the case : $n = 8, \pi(1) = 4, \pi(2) = 3, \pi(3) = 5, \pi(4) = 1, \pi(5) = 2, \pi(6) = 8, \pi(7) = 7, \pi(8) = 6$.

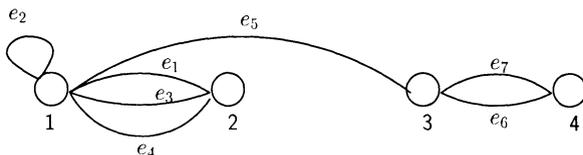


Figure 11.1. Patching pseudograph G_p^π

An optimal tour γ^* is an optimal assignment of a successor job to each job i which results in a cyclic ordering of the jobs. This can be expressed

as an optimal matching of the values $\{b_i : i \in N\}$ to values $\{a_i : i \in N\}$ which results in a cyclic job ordering. The Gilmore-Gomory scheme starts by optimally matching of the b_i 's with the a_i 's, disregarding the requirement that the resultant job ordering should be cyclic. If the resultant permutation π on the job set N is a tour, then it is obviously an optimal tour. If we get subtours, then these subtours are patched together to get a tour as follows: a pair $\{i, i + 1\} \subseteq \{1, \dots, n - 1\}$ is chosen such that jobs i and $i + 1$ lie in different subtours of π . (It may be noted the choice of patchings depends on the numbering of the jobs.) The permutation π is then modified to $\pi \circ \beta_i$. This results in the two subtours of π containing jobs i and $i + 1$ being combined into one, while the other subtours of π remain unaffected. This operation, which we call *adjacent patching*, is then repeated until we get a tour. The patching pseudograph of π plays an important role in identifying an optimal set of adjacent patchings. A formal description of the algorithm is given below.

Gilmore-Gomory Scheme for the GG-TSP

- Step 1:** Renumber the jobs such that $b_1 \leq b_2 \leq \dots \leq b_n$. Let π be a permutation on the set N such that $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$. If π is a tour then output π and stop.
- Step 2:** Suppose the digraph G_π has m connected components. Construct the patching pseudograph $G_p^\pi = (N_p^\pi, E_p^\pi)$. Assign to each edge $e_i \in E_p^\pi$ a cost $w_i = d_{ii}$, where D is the density matrix of C^π . Find a minimum cost spanning tree $E_p^\pi[T^*]$ in G_p^π , (that is, the edge set of the optimal spanning tree is $\{e_i : i \in T^*\}$).
- Step 3:** Construct a digraph $G_O = (T^*, E_O)$ where E_O is defined as follows: for all $\{i - 1, i\} \subseteq T^*$,

$$\begin{aligned} (i - 1, i) &\in E_O \text{ if } d_{ij} = 0 \text{ for all } j < i \\ (i, i - 1) &\in E_O \text{ otherwise.} \end{aligned}$$

Find an ordering (i_1, i_2, \dots, i_k) of its node set T^* such that for any $e = (i_u, i_v) \in E_O$, $u < v$. (Since the digraph G_O is acyclic, such an ordering exists.) Then $\gamma^* = \pi \circ \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$ is a tour and is the desired output. Stop.

It is not difficult to verify that the complexity of the above scheme is $O(n \log n)$.

Let us now illustrate the above algorithm with an example. Let $n = 8$, $(a_1, b_1) = (250, 100)$, $(a_2, b_2) = (160, 140)$, $(a_3, b_3) = (350, 200)$, $(a_4, b_4) = (100, 300)$, $(a_5, b_5) = (120, 450)$, $(a_6, b_6) = (550, 500)$, $(a_7, b_7) =$

$(500, 600)$, $(a_8, b_8) = (400, 700)$; $f(x) = 2$ and $g(x) = -1$ for all $x \in \mathbb{R}$. In this case, the cost matrix is

$$C = \begin{bmatrix} 300 & 120 & 500 & 0 & 40 & 900 & 800 & 600 \\ 220 & 40 & 420 & -40 & -20 & 820 & 720 & 520 \\ 100 & -40 & 300 & -100 & -80 & 700 & 600 & 400 \\ -50 & -140 & 100 & -200 & -180 & 500 & 400 & 200 \\ -200 & -290 & -100 & -350 & -330 & 200 & 100 & -50 \\ -250 & -340 & -150 & -400 & -380 & 100 & 0 & -100 \\ -350 & -440 & -250 & -500 & -480 & -50 & -100 & -200 \\ -450 & -540 & -350 & -600 & -580 & -150 & -200 & -300 \end{bmatrix}.$$

Step 1: We already have $b_1 \leq b_2 \leq \dots \leq b_8$. Hence, no renumbering of jobs is necessary. We have $\pi(1) = 4$, $\pi(2) = 5$, $\pi(3) = 2$, $\pi(4) = 1$, $\pi(5) = 3$, $\pi(6) = 8$, $\pi(7) = 7$, $\pi(8) = 6$. The permutation π is not a tour. The graph G_π is shown in Figure 11.2.

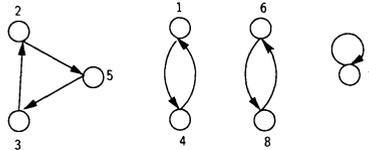


Figure 11.2. The digraph G_π

The matrix C^π and its density matrix D are as given below.

$$C^\pi = \begin{bmatrix} 0 & 40 & 120 & 300 & 500 & 600 & 800 & 900 \\ -40 & -20 & 40 & 220 & 420 & 520 & 720 & 820 \\ -100 & -80 & -40 & 100 & 300 & 400 & 600 & 700 \\ -200 & -180 & -140 & -50 & 100 & 200 & 400 & 500 \\ -350 & -330 & -290 & -200 & -100 & -50 & 100 & 200 \\ -400 & -380 & -340 & -250 & -150 & -100 & 0 & 100 \\ -500 & -480 & -440 & -350 & -250 & -200 & -100 & -50 \\ -600 & -580 & -540 & -450 & -350 & -300 & -200 & -150 \end{bmatrix},$$

and

$$D = \begin{bmatrix} 20 & 20 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 40 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 50 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 50 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Step 2 : $m = 4$; $N_1 = \{2, 3, 5\}$, $N_2 = \{1, 4\}$, $N_3 = \{6, 8\}$, $N_4 = \{7\}$. The patching pseudograph G_p^π is precisely the one shown in Figure 11.1. The weights of the edges of G_p^π are $\{w_1 = d_{11} = 20, w_2 = d_{22} = 20, w_3 =$

$d_{33} = 50, w_4 = d_{44} = 50, w_5 = d_{55} = 0, w_6 = d_{66} = 0, w_7 = d_{77} = 0$. Hence, $\{e_1, e_5, e_6\}$ is the edge set of a minimal cost spanning tree, (that is, $T^* = \{1, 5, 6\}$).

Step 3 : The corresponding Order digraph G_O is shown in Figure 11.3.



Figure 11.3. Order digraph $G_O = (T^*, E_O)$

This gives us ordering $(1, 5, 6)$ of elements of T^* . Hence, $\gamma^* = \pi \circ \beta_1 \circ \beta_5 \circ \beta_6$. Thus, the output is $\gamma^* = (1, 5, 8, 6, 7, 3, 2, 4, 1)$.

The key operation used in the above algorithm is *post-multiplication of the starting permutation π by a sequence of adjacent transpositions*. We now present some basic results that will help us develop a better feel for this operation and therefore the algorithm. This will simplify explanation of the extensions of the scheme that will be discussed in the next section.

3.2. Some basic results

Recall that a circuit is a permutation with only one non-trivial subtour.

Observation 11 [360] For $i \in \{1, 2\}$, let φ_i be a circuit on N with its unique non-trivial subtour \mathfrak{C}_i on node set N_i . If $N_1 \cap N_2 = \emptyset$, then $\varphi_1 \circ \varphi_2 = \varphi_2 \circ \varphi_1$.

For example, for any $\{i, j\} \subseteq \{1, 2, \dots, n\}$ such that $|i - j| > 1$, $\beta_i \circ \beta_j = \beta_j \circ \beta_i$. However, $\beta_i \circ \beta_{i+1} \neq \beta_{i+1} \circ \beta_i$.

Observation 12 [360] Let π be a permutation on N with non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_r$ on node sets N_1, N_2, \dots, N_r , respectively. For each $j \in \{1, \dots, r\}$, let φ_j be the circuit on N with \mathfrak{C}_j as its unique non-trivial subtour. Then $\pi = \varphi_{i_1} \circ \varphi_{i_2} \circ \dots \circ \varphi_{i_r}$ for any ordering (i_1, i_2, \dots, i_r) of the elements of the set $\{1, 2, \dots, r\}$.

Observation 13 [360] Let π be an arbitrary permutation on N and let $\{i, j\} \subseteq N$. (i) If i and j both belong to the same subtour \mathfrak{C} of π then in $\pi \circ \alpha_{ij}$, the subtour \mathfrak{C} is decomposed into two subtours, one containing i and the other containing j , while all the other subtours of $\pi \circ \alpha_{ij}$ are precisely the same as the other subtours of π . (ii) If i and j belong to

two different subtours \mathfrak{C}_1 and \mathfrak{C}_2 of π then in $\pi \circ \alpha_{ij}$, the two subtours \mathfrak{C}_1 and \mathfrak{C}_2 are combined into a single subtour \mathfrak{C} while all the other subtours of $\pi \circ \alpha_{ij}$ are precisely the same as the other subtours of π .

In case (ii), we say that the subtour \mathfrak{C} is obtained by *patching* the subtours \mathfrak{C}_1 and \mathfrak{C}_2 . If $j = i + 1$, then we call it an *adjacent patching*.

For any set $S \subseteq \{1, 2, \dots, n - 1\}$, define $\{S_1, S_2, \dots, S_\ell\}$, the *natural partition* of S , as follows. For any $x \in \{1, \dots, \ell\}$, S_x is of the form $\{i_x, i_x + 1, \dots, j_x\}$ and for all $x \in \{1, \dots, \ell - 1\}$, $j_x + 1 < i_{x+1}$.

Observation 14 [360] *For any set $S \subseteq \{1, 2, \dots, n - 1\}$, with natural partition $\{S_1, S_2, \dots, S_\ell\}$, consider all the permutations $\beta_{u_1} \circ \beta_{u_2} \circ \dots \circ \beta_{u_r}$ produced by using all the possible orderings (u_1, u_2, \dots, u_r) of the elements of S . Then it follows from observations 11, 12 and 13 that each of these permutations will have precisely ℓ non-trivial subtours on the same node sets $\{\{i_x, i_x + 1, \dots, j_x + 1\} : x \in \{1, \dots, \ell\}\}$ and if two different orderings of the elements of S are such that for each $x \in \{1, 2, \dots, \ell\}$ the ordering of the elements of S_x is the same in both of them then they will both produce the same permutation.*

3.2.1 Adjacent patchings and pyramidal permutations.

A path in a digraph $G = (N, E)$ is said to be a *pyramidal path* [8] if and only if it is of the form $(i_1, i_2, \dots, i_u, j_1, j_2, \dots, j_v)$ with $i_1 < i_2 < \dots < i_u$ and $j_1 > j_2 > \dots > j_v$. Note that whether a path is pyramidal depends on the numbering of the nodes. A closed, pyramidal path is called a *pyramidal subtour*. A permutation is said to be *pyramidal* if and only if all its non-trivial subtours are pyramidal and it is said to be *dense* [141] if and only if the node set of each of its non-trivial subtours is of the form $\{i, i + 1, \dots, j\}$. Let π be a dense permutation with its non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_r$ on node sets $\{i_1, i_1 + 1, \dots, j_1\}, \{i_2, i_2 + 1, \dots, j_2\}, \dots, \{i_r, i_r + 1, \dots, j_r\}$, respectively. Then we say that π is dense on the node set $\{i_1, i_1 + 1, \dots, j_1 - 1\} \cup \{i_2, i_2 + 1, \dots, j_2 - 1\} \cup \dots \cup \{i_r, i_r + 1, \dots, j_r - 1\}$ [141].

Our interest in pyramidal and dense permutations is mainly due to their well-known relationship with permutations representable as product of adjacent transpositions. We establish this next. We do not know who was the first to observe this relationship.

Let $S = \{u, u + 1, \dots, v\}$, for some $1 \leq u \leq v \leq n - 1$. For an arbitrary ordering (i_1, i_2, \dots, i_k) of the elements of the set S , define the corresponding *order digraph* $G_O = (S, E_O)$ as follows. For any $j \in \{u, \dots, v - 1\}$, if j precedes $(j + 1)$ in the ordering then $(j, j + 1) \in E_O$; else, $(j + 1, j) \in E_O$. The arc set E_O forms a simple chain on node set S between u and v . For every ordering of the elements of S , there is a

unique order digraph. Conversely, any simple chain between u and v on node set S is the order digraph corresponding to one or more orderings of elements of S .

Lemma 15 *For any set $S = \{u, u + 1, \dots, v\}$, where $1 \leq u \leq v \leq n - 1$, and any ordering (i_1, i_2, \dots, i_k) of elements of S , the permutation $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$ is a pyramidal circuit dense on the set S . Two different orderings of the elements of S with the same order digraph result in the same circuit. Conversely, for any pyramidal circuit, ψ , dense on S , there exists a simple chain on S between u and v such that for any ordering (i_1, i_2, \dots, i_k) of elements of S , having this simple chain as its order digraph, $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$.*

The lemma can be proved by mathematical induction on $(v - u)$ using the following construction schemes.

Consider an arbitrary ordering (i_1, i_2, \dots, i_k) of the elements of the set S . Let $G_O = (S, E_O)$ be its order digraph. Define $u = x_0 \leq x_1 < x_2 < \dots < x_{2r} \leq v$ such that $(x_0, x_0 + 1, \dots, x_1)$, $(x_2, x_2 - 1, \dots, x_1)$, $(x_2, x_2 + 1, \dots, x_3)$, \dots , $(x_{2r}, x_{2r} - 1, \dots, x_{2r-1})$, $(x_{2r}, x_{2r} + 1, \dots, v)$ are all paths in G_O . Then the unique non-trivial subtour of $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$ is $(x_0, x_0 + 1, \dots, x_1, x_2 + 1, x_2 + 2, \dots, x_3, x_4 + 1, \dots, x_{2r-1}, x_{2r} + 1, \dots, v + 1, x_{2r}, x_{2r} - 1, \dots, x_{2r-1} + 1, x_{2r-2}, \dots, x_{2r-3} + 1, \dots, x_1 + 1, x_0)$.

For example, let $S = \{3, 4, 5, 6, 7\}$ and consider the simple chain on S between nodes 3 and 7 as shown in Figure 11.4.

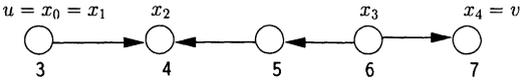


Figure 11.4. A simple chain on $S = \{3, 4, 5, 6, 7\}$

Then any ordering of the elements of S having the digraph in Figure 11.4 as its order digraph will result in the pyramidal, dense circuit with unique subtour $(3, 5, 6, 8, 7, 4, 3)$.

Conversely, for any pyramidal circuit ψ which is dense on set $S = \{u, u + 1, \dots, v\}$, we construct a simple chain G_O on S between u and v such that for any ordering (i_1, i_2, \dots, i_k) of the elements of S with G_O as its order digraph, $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$: Define $T^F = \{i : i \in S; \psi(i) > i + 1\}$ and $T^B = \{i : \psi^{-1}(i) > i + 1\}$. The elements of

$T^F \cup T^B \cup \{v\}$ can be arranged as $u = x_0 \leq x_1 < x_2 < \dots < x_{2r} \leq v$ such that $\psi(v + 1) = x_{2r}$ and for all $1 \leq j \leq r$, $\psi(x_{2j-1}) = x_{2j} + 1$ and $\psi(x_{2j-1} + 1) = x_{2j-2}$. Define digraph $G_O = (S, E_O)$ as follows. For all $0 \leq j < r$ and for all $x_{2j} \leq s < x_{2j+2}$, $(s, s + 1) \in E_O$ if $s < x_{2j+1}$; and $(s + 1, s) \in E_O$ if $s \geq x_{2j+1}$. For all $x_{2r} \leq s < v$, $(s, s + 1) \in E_O$.

Definition 16 For a cost matrix C and any two permutations π and ψ on N , the cost of ψ relative to π is

$$c(\pi \circ \psi) - c(\pi) = c^\pi(\psi) - c^\pi(\xi).$$

For a given cost matrix C and permutation π on N , let \bar{C} and \hat{C} be the matrices obtained from C by subtracting suitable constants from each of its rows and columns such that $\bar{c}_{1,1}^\pi = \bar{c}_{2,1}^\pi = \bar{c}_{2,2}^\pi = \bar{c}_{3,2}^\pi = \dots = \bar{c}_{n,n-1}^\pi = \bar{c}_{n,n}^\pi = 0$ and $\hat{c}_{1,1}^\pi = \hat{c}_{1,2}^\pi = \hat{c}_{2,2}^\pi = \hat{c}_{2,3}^\pi = \dots = \hat{c}_{n-1,n}^\pi = \hat{c}_{n,n}^\pi = 0$. Let D be the density matrix of C^π . Then D is also the density matrix of each of \bar{C}^π and \hat{C}^π . It is readily apparent that for any $i < j$, \bar{c}_{ij}^π is the sum of all the entries of the matrix D (including the diagonal entries) in the triangle formed by the (i, i) th, $(j - 1, j - 1)$ th and $(i, j - 1)$ th entries; \hat{c}_{ji}^π is the sum of all the entries of the matrix D (including the diagonal ones) in the triangle formed by the (i, i) th, $(j - 1, j - 1)$ th and $(j - 1, i)$ th entries; and for all $i < j - 1 < n$, \hat{c}_{ij}^π is the sum of all the non-diagonal entries of the matrix D in the triangle formed by the (i, i) th, $(j - 1, j - 1)$ th and $(i, j - 1)$ th entries and \bar{c}_{ji}^π is the sum of all the non-diagonal entries of the matrix D in the triangle formed by the (i, i) th, $(j - 1, j - 1)$ th and $(j - 1, i)$ th entries.

For any $(n - 1) \times (n - 1)$ matrix A let us define for any $1 \leq i < j \leq n$, $A^{i,j}$ as the $(n - 1) \times (n - 1)$ matrix with

$$a_{uv}^{i,j} = \begin{cases} a_{uv} & \text{if } i \leq u \leq v < j \\ 0 & \text{otherwise.} \end{cases}$$

and for any $1 \leq j \leq i \leq n$, let $A^{i,j}$ be the $(n - 1) \times (n - 1)$ matrix with

$$a_{uv}^{i,j} = \begin{cases} a_{uv} & \text{if } j \leq v < u < i \\ 0 & \text{otherwise.} \end{cases}$$

Then for any $1 \leq i \neq j \leq n$, \bar{c}_{ij}^π is the sum of all the entries of $D^{i,j}$. For any permutation ψ on N , let $D^{(\psi)} = \sum_{i \in N} D^{i,\psi(i)}$. Then, the cost of ψ relative to π is $c^\pi(\psi) - c^\pi(\xi) = \bar{c}^\pi(\psi)$ is the sum of all the entries of $D^{(\psi)}$. Thus, $c^\pi(\psi) - c^\pi(\xi)$ is a function of the entries of the matrix D and we shall denote it by $D(\psi)$. (For example, for any adjacent transposition β_i , the cost of β_i relative to π is $D(\beta_i) = d_{ii}$.) Let the non-trivial subtours

of ψ be $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_r$ on node sets N_1, N_2, \dots, N_r respectively. For each $1 \leq j \leq r$, let φ_j be the circuit on N with \mathfrak{C}_j as its only non-trivial subtour. Then $D(\psi) = \sum_{1 \leq j \leq r} D(\varphi_j)$.

We shall now give a geometric interpretation of the structure of the cost of a pyramidal tour.

Observation 17 [479] *Let π be an arbitrary permutation on N and let D be the density matrix of C^π . For any $1 \leq u \leq v < n$, let (i_1, i_2, \dots, i_k) be a given arbitrary ordering of the elements of the set $S = \{u, u + 1, \dots, v\}$ and let G_O be its order digraph. Let $u = x_0 \leq x_1 < x_2 < \dots < x_{2r} \leq v$ be such that*

$$(x_0, x_0 + 1, \dots, x_1), (x_2, x_2 - 1, \dots, x_1), (x_2, x_2 + 1, \dots, x_3), \dots, \\ (x_{2r}, x_{2r} - 1, \dots, x_{2r-1}), (x_{2r}, x_{2r} + 1, \dots, v)$$

are all paths in G_O . Let $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$. The cost of ψ relative to π is then given by

$$c^\pi(\psi) - c^\pi(\xi) = D(\psi) \\ = \left(\sum_{s=u}^v d_{s,s} \right) + (\text{the sum of all the non-diagonal entries} \\ \text{of } D \text{ in the triangles formed by the triplets of entries} \\ \{[(x_0, x_0), (x_1, x_0), (x_1, x_1)], [(x_1, x_1), (x_1, x_2), (x_2, x_2)], \\ [(x_2, x_2), (x_3, x_2), (x_3, x_3)], \dots, [(x_{2r-1}, x_{2r-1}), \\ (x_{2r-1}, x_{2r}), (x_{2r}, x_{2r})], [(x_{2r}, x_{2r}), (v, x_{2r}), (v, v)]\}.$$

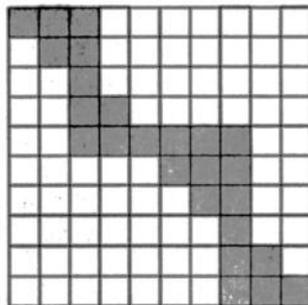


Figure 11.5. Cost structure of a pyramidal tour

(See Figure 11.5.) Thus, the permutation $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$ and its cost relative to π depend only on the order digraph G_O of (i_1, i_2, \dots, i_k)

and the density matrix D of C^π . For any $S \subseteq \{1, 2, \dots, n - 1\}$, let \mathcal{F} be the set of all the orderings of the elements of S . Define,

$$D[S] = \min\{D(\beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}) : (i_1, i_2, \dots, i_k) \in \mathcal{F}\}.$$

It follows from the above that if $S_1 \cup S_2 \cup \dots \cup S_\ell$, is the natural partition of S , then $D[S] = \sum_{i=1}^\ell D[S_i]$.

We are now in a position to understand the Gilmore-Gomory scheme better and we give below a more detailed explanation of its steps. We do not give here a formal proof of validity of this scheme, as this will follow from the proofs of validity of its generalization to larger subclasses of the TSP, which include the class of GG-TSP, which we present in Section 4.

3.3. Explanation of the steps of Gilmore-Gomory scheme for GG-TSP

Let π be the permutation on the set N obtained in Step 1 of the Gilmore-Gomory scheme, (that is, $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$). The density matrix D of C^π satisfies the following properties [360].

Property 1 :
$$d_{ij} = \int_{\max\{b_i, a_{\pi(j)}\}}^{\min\{b_{i+1}, a_{\pi(j+1)}\}} (f(x) + g(x))dx \geq 0.$$

Property 2 : If $b_i \geq a_{\pi(i)}$ then $d_{ij} = 0$ for all $j < i$; while, if $b_i \leq a_{\pi(i)}$ then $d_{i-1,j} = 0$ for all $j \geq i$.

Since D is a non-negative matrix, π is an optimal permutation (optimal solution to the Assignment Problem on C). If π is a tour then it is obviously an optimal tour. Suppose the digraph G_π has m connected components for some $m > 1$. Let $E_p^\pi[T]$ be the edge set of a spanning tree of G_p^π (see definition 10). Then for any $u \in T$, elements u and $(u + 1)$ lie in different connected components (subtours) of π . Let $\pi' = \pi \circ \beta_u$, that is π' is obtained from π by an *adjacent patching*. By Observation 13, this results in an interchange of the successors of u and $(u + 1)$ and the two subtours containing these two elements of N are combined into one, while the other subtours of π remain unaffected. Thus, π' has one less subtour than π and the edge set $E_p^{\pi'}[T - \{u\}]$ forms a spanning tree of $G_p^{\pi'}$.

Recursively, we get that for any ordering $(j_1, j_2, \dots, j_{m-1})$ of the elements of T , $\pi \circ \beta_{j_1} \circ \dots \circ \beta_{j_{m-1}}$ is a tour on N . We call this process of obtaining a tour from a given permutation π as *Gilmore-Gomory patching* or *GG-patching*. Obviously, every tour on N cannot be obtained this way. For example, if $N = \{1, 2, 3, 4\}$, $\pi = \xi$ (the identity permuta-

tion) and $\psi = (1, 3, 2, 4, 1)$, then ψ is not a pyramidal tour and hence, it cannot be obtained from π by GG-patching. Gilmore-Gomory [360] show that there exists an optimal tour for the GG-TSP that is obtained from the permutation π of Step 1 of their algorithm by GG-patching.

Define a digraph $G_O = (N - \{n\}, E_O)$ as follows: For any $1 \leq j < n$, if $b_{j+1} \geq a_{\pi(j+1)}$, then $(j, j+1) \in E_O$; else, $(j+1, j) \in E_O$. It follows by Observation 17 and Property 2 that for any $S \subseteq N$ and any ordering (i_1, i_2, \dots, i_k) of the elements of S with $G_O[S]$ as its order digraph, the cost of $\psi = \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_k}$ relative to π is $D(\psi) = \sum_{j \in S} d_{jj}$. Since D is non-negative, this implies that $D[S] = \sum_{j \in S} d_{jj}$. Thus, the problem of finding an optimal GG-patching reduces to the problem of finding a minimum cost spanning tree $E_p^\pi[T^*]$ in G_p^π where d_{ii} is the cost of each edge e_i . An optimal ordering of the transpositions corresponding to the elements of T^* is given by $G_O[T^*]$.

It may be noted that the Gilmore-Gomory scheme for the GG-TSP does not require full information about the cost matrix C , but it only requires information about the density matrix D of C^π .

3.4. Other applications of GG-TSP

Ball et al [83] consider an interesting special case of GG-TSP that arises in the context of the problem of optimal insertion of chips on a PC board.

Interesting real world applications of GG-TSP are also found in the area of flow shop scheduling with limited storage space between machines. Here, we have k classes of jobs and we are given n jobs, out of which n_i jobs belong to class i for $i = 1, 2, \dots, k$. (Thus, $n = \sum_{i=1}^k n_i$.) These jobs are to be processed on m given machines M_1, \dots, M_m in this order. The processing time of a job of class i on machine j is given by p_{ij} . Each machine can process only one job at a time. The storage space between a pair of consecutive machines M_i and M_{i+1} is limited and can accommodate at most s_i jobs. The problem is to schedule the jobs on the machines so as to minimize the total makespan (time required to process all the jobs). When all s_i 's are zero, we call the problem a "no wait" problem. In this case, for any pair i, j of classes of jobs, the difference in the completion times of two consecutively processed jobs, where the first job belongs to class i and the succeeding job belongs to class j , depends only on i and j and will be denoted by c_{ij} . If we add a dummy job of some $(k+1)^{st}$ class with $p_{k+1,j} = 0$ for all $j \in \{1, \dots, m\}$, the problem reduces to an instance of TSP [700, 824]. The no wait problem with only one job of each class is shown to be NP-hard for fixed $m \geq 4$ [653] and for $m = 3$ [728]. For $m = 2$ with only one job of each class, it is shown

in [700] that the no wait problem can be formulated as a special case of the GG-TSP with

$$\begin{aligned} c_{ij} &= p_{j,2} - p_{i,2} + \max\{p_{j,1}, p_{i,2}\} \\ &= p_{j,2} + \max\{p_{j,1} - p_{i,2}, 0\} \\ &= p_{j,2} + c'_{ij} \end{aligned}$$

where, $c'_{ij} = \max\{p_{j,1} - p_{i,2}, 0\}$. The matrix C' has the same density matrix as C and C' is of the Gilmore-Gomory type with $g(x) = 1$ and $f(x) = 0$ for all x . Thus, the problem can be solved in $O(k \log k)$ time using the Gilmore-Gomory algorithm. The matrix C of this type is called a *pseudo-large matrix* [798]. In [805] a special case of the no wait problem with arbitrary m and $n_i = 1$ for all i is considered in which the processing times are fixed on all but a pair (M_u, M_v) of machines, (that is, $p_{ij} = c_j$ for all i and all $j \in \{1, \dots, m\} - \{u, v\}$). It is shown that this special case can be reduced to TSP on a pseudo-large cost matrix and thus can be solved in $O(k \log k)$. For $m = 2$, $n_i = 1$ for all i , if $0 < s_1 < k - 1$, the problem has been shown to be NP-hard in [653], while if $s_i \geq k - 1$, then the storage space limitation causes no bottleneck and the problem can be solved using the well-known Johnson's rule [467]. The case $m = 2$ and arbitrary $n_i > 1$ is considered in [3]. Here it is shown that for arbitrary $s_1 > 0$ and $n_i \geq s_1 + 1$, the problem is NP-hard, while if $n_i > \lceil s_1 \max\{p_{i,1}, p_{i,2}\} / |p_{i,1} - p_{i,2}| \rceil + 1$, the problem can be reduced to TSP on a pseudo-large cost matrix and hence can be solved in $O(k \log k)$ time.

Kollias et al [511] consider a modification of the GG-TSP where the starting temperature a_0 of the furnace is specified but there is no restriction on the ending temperature b_0 . In this case, the ending temperature will obviously be the ending temperature of the last job processed and if we know that job i will be processed last, then we can fix $b_0 = b_i$ and thereby reduce the problem to an instance of GG-TSP. By trying all the n possible choices of the ending job, the problem can be reduced to n instances of GG-TSP. Through an initial $O(n \log n)$ time pre-processing step involving sorting, the complexity of each of these instances of GG-TSP can be reduced to $O(n)$. This gives us an overall complexity of $O(n^2)$. In [511], an algorithm is proposed for a special case of this problem where $a_i < b_i$ for all $i \in N$, $a_0 < \min\{a_i : i \in N\}$ and $f(x) = g(x) = 1$ for all x . In this case, it is easy to show that there exists an optimal solution in which the job with largest b value will be processed last. Kollias et al [511] also propose an $O(n \log n)$ algorithm for an undirected version of the problem under the condition that $a_0 < a_i < b_i$ for all i and $f(x) = g(x) \geq 0$ for all x . Their algorithm however may not

produce an optimal solution as in the following example.

$$n = 3, a_1 = 1, b_1 = 22, a_2 = 2, b_2 = 24, a_3 = 21, b_3 = 23.$$

An optimal tour in this case is $(0, 1, 3, 2, 1)$ with a total cost of 3. The tour produced by their scheme is however $(0, 1, 3, 2, 0)$ with a total cost of 21. An $O(n^2)$ algorithm for this problem follows from the results in [486].

4. GG Scheme: a generalization of Gilmore-Gomory scheme for GG-TSP

We shall now discuss a generalization of the Gilmore-Gomory scheme for GG-TSP to a patching scheme for a general TSP. This scheme, which we call GG scheme, has been studied by various researchers [51, 139, 142, 361, 479, 481, 483, 484, 486, 485, 487, 775, 799, 801].

Algorithm GG Scheme

Input: An $n \times n$ cost matrix C , a permutation π on N .

Step 1: Check if π is a tour. If yes, then output π and stop. Suppose π has r subtours, for some $r > 1$, on node sets N_1, \dots, N_r .

Step 2: Construct the patching pseudograph $G_p^\pi = (N_p^\pi, E_p^\pi)$ of π .

Step 3: Find a spanning tree $E_p^\pi[T^*]$ in G_p^π such that $D[T^*]$ is minimum, where D is the density matrix of C^π . Let $(i_1, i_2, \dots, i_{r-1})$ be an ordering of the elements of T^* such that $D[T^*] = D(\beta_{i_1} \circ \dots \circ \beta_{i_{r-1}})$. Output the tour $\gamma = \pi \circ \beta_{i_1} \circ \dots \circ \beta_{i_{r-1}}$. Stop.

Two natural questions which arise are:

- (i) What are the necessary and sufficient conditions on the pair (C, π) under which an optimal tour can be obtained by applying the GG scheme with π as the starting permutation?
- (ii) What is the computational complexity of the GG Scheme?

As we shall see in Section 4.1.1, checking, for a given pair (C, π) , if the GG scheme with π as the starting permutation produces an optimal tour is co-NP-complete even for the special case $\pi = \xi$. Hence, the problem of finding an elegant answer to question (i) seems difficult. Also, it follows from the results of [139, 742], (see also [361]), that implementation of the GG scheme for an arbitrary, starting permutation π is NP-hard even when C is a product matrix, (that is, for some $\{a_i, b_i \in \mathbb{R} : i \in N\}$, $c_{ij} = a_i b_j$).

The results reported in the literature fall into three main categories. (i) For a given specific permutation π , sufficient conditions on matrix D under which for any cost matrix C such that D is the density matrix of C^π the GG scheme with π as the starting permutation produces an optimal tour to $TSP(C)$. Most of these results deal with the case $\pi = \xi$. (ii) Some necessary and some sufficient conditions on matrix D under which for any cost matrix C and any permutation π on N such that D is the density matrix of C^π , the GG scheme with π as the starting permutation produces an optimal tour to $TSP(C)$. (iii) Polynomial time algorithms for implementation of the GG scheme for various special cases of the matrix D and/or the starting permutation π .

4.1. Sufficiency conditions for validity of GG scheme for a given starting permutation π

Most of the existing results regarding the validity of the GG scheme for a specific starting permutation deal with the case $\pi = \xi$. The following result by Michalski [594] is one of the few which do not fall in this category.

For any $n \times n$ matrix A with its density matrix D and any $1 \leq i, j < n$ and $1 \leq u, v < n$, we denote

$$M_{i,j,u,v}^D = \sum_{y=u}^v \sum_{x=i}^j d_{xy}. \tag{1}$$

From the definition of density matrix, it follows that,

$$M_{i,j,u,v}^D = \begin{cases} a_{i,v+1} + a_{j+1,u} - a_{i,u} - a_{j+1,v+1}, & \text{if } i \leq j \text{ and } u \leq v \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Theorem 18 [594] *Suppose C is symmetric with density matrix D such that for all $1 \leq i < j < n$ and $1 \leq u < v < n$, $\{i, j\} \cap \{u, v\} = \emptyset$, $M_{i,j,u,v}^D \leq 0$. Then $\psi = (1, n - 1, 3, n - 3, \dots, 4, n - 2, 2, n, 1)$ is an optimal solution to $TSP(C)$.*

Theorem 18 can be proved by induction on n using the following facts. (i) Any principal submatrix of C also satisfies the condition of the theorem. (ii) If we define permutation π as : $\pi(i) = (n - i + 1)$ for all $i \in N$, then the density matrix D' of C^π satisfies the property : $d'_{ii} = d'_{n-i,n-i}$ for all $1 \leq i < n$ and $d'_{ij} \geq 0$ for all $1 \leq i, j < n$ such that $i + j \notin \{n - 1, n, n + 1\}$. We leave the details as an exercise.

It can be shown that if we implement the GG scheme with starting permutation π as defined above, then $\{e_i : i \in S = \{1, n - 2, 3, n -$

$4, \dots\}$ is the arc set of an optimal spanning tree of G_p^π in Step 3 of the GG scheme. Performing the GG-patching corresponding to the elements of S in any order gives us the tour $\psi = (1, (n-1), 3, (n-3), \dots, 4, (n-2), 2, n, 1)$. Thus, the GG scheme produces an optimal solution to $TSP(C)$.

We now discuss the case $\pi = \xi$. It follows from Lemma 15 that in this case any solution produced by the GG scheme is a minimum cost pyramidal tour. We shall therefore refer to an instance of the TSP for which the GG scheme produces an optimal solution with $\pi = \xi$ as *pyramidally solvable*.

4.1.1 Polynomially testable sufficiency conditions for TSP to be pyramidally solvable.

For any digraph on node set N , we denote by \wp_{ij} a path from node i to node j . For a path $\wp_{ij} = ((i =)u_0, u_1, u_2, \dots, u_r, u_{r+1}(=j))$ and for any $0 \leq k \leq r$, and $1 \leq \ell \leq r+1$, we define $\wp_{ij}(u_k) = u_{k+1}$, $\wp_{ij}^{-1}(u_\ell) = u_{\ell-1}$ and we call u_{k+1} ($u_{\ell-1}$) the *successor* (*predecessor*) of u_k (u_ℓ) in \wp_{ij} . A *peak* of the path \wp_{ij} is a node p in \wp_{ij} such that (i) $p > \wp_{ij}(p)$, if $\wp_{ij}(p)$ exists and (ii) $p > \wp_{ij}^{-1}(p)$, if $\wp_{ij}^{-1}(p)$ exists. Similarly, we define a *valley* of the path \wp_{ij} as a node v in \wp_{ij} such that (i) $v < \wp_{ij}(v)$, if $\wp_{ij}(v)$ exists and (ii) $v < \wp_{ij}^{-1}(v)$, if $\wp_{ij}^{-1}(v)$ exists. Thus, a pyramidal path is a path with only one peak and/or only one valley and a tour is pyramidal if and only if it has exactly one peak (node n) and exactly one valley (node 1).

It is shown in [548] that for a symmetric cost matrix C that obeys the triangle inequality, and a tour γ on N , the problem of checking if there exists another tour ψ with $c(\psi) < c(\gamma)$ is *NP-complete*. This problem can be easily reduced to the problem of checking if a given instance $TSP(C)$ is pyramidally solvable, thereby implying that the later problem is co-NP-complete. In view of this fact, it seems highly unlikely that polynomially testable conditions, which are necessary and sufficient for a given instance of TSP to be pyramidally solvable, exist. A considerable amount of literature exists on the subject of polynomially testable sufficiency conditions [8, 9, 34, 51, 54, 52, 53, 56, 58, 57, 60, 144, 142, 148, 250, 251, 252, 361, 491, 507, 506, 690, 776, 777, 778, 801, 803, 804, 819, 820]. The first such sufficiency condition was reported in [776]. We find the first explicit reference to the class of pyramidal tours in [8] where it is shown that if the cost matrix C is an ordered product matrix, (that is, $c_{ij} = (a_i b_j)$ for some $0 \leq a_1 \leq a_2 \leq \dots \leq a_n$ and $b_1 \geq b_2 \geq \dots \geq b_n \geq 0$), then there exists an optimal tour to $TSP(C)$ that is pyramidal. As we shall see later, this result follows from the fact that in this case, the density matrix D of C is non-negative.

We shall now discuss the most general polynomially testable sufficiency conditions known to date. But first we shall present some basic results.

We denote by C^T , the transpose of the matrix C and by C^R , the reverse of C , (that is, $c_{ij}^R = c_{n-i+1, n-j+1}$). For any tour $\gamma = (i_1, \dots, i_n, i_1)$ on N , γ^{-1} is the tour $(i_1, i_n, i_{n-1}, \dots, i_2, i_1)$ and γ^R is the tour $(n - i_1 + 1, n - i_2 + 1, \dots, n - i_n + 1, n - i_1 + 1)$. A tour γ is optimal for $TSP(C)$ if and only if γ^{-1} is an optimal solution to $TSP(C^T)$ and γ^R is an optimal solution to $TSP(C^R)$. Thus, $TSP(C)$ is pyramidally solvable if and only if each of $TSP(C^T)$ and $TSP(C^R)$ is pyramidally solvable.

For any tour γ on N , we recursively define the permutations $\{\gamma^{(i)} : i \in N\}$ as follows. $\gamma^{(n)} = \gamma$. For $i = n - 1, n - 2, \dots, 1$, let $\gamma^{(i+1)}(u_i) = i + 1$. Then, $\gamma^{(i)} = \gamma^{(i+1)} \circ \alpha_{u_i, i+1}$, (that is, $\gamma^{(i)}$ is obtained from $\gamma^{(i+1)}$ by replacing subpath $(u_i, i + 1, v_i)$ of $\gamma^{(i+1)}$ by arc (u, v) and loop $(i + 1, i + 1)$). For a given cost matrix C , let $c^{\gamma, i} = c(\gamma^{(i)}) - c(\gamma^{(i-1)})$.

We make the following observations. For all $2 \leq i \leq n$,

- (i) $\gamma^{(i)}$ is a circuit with $\{1, 2, \dots, i\}$ as the node set of its unique non-trivial subtour;
- (ii) $c^{\gamma, i} = (c_{ui} + c_{iv} - c_{uv} - c_{ii}) = M_{u, i-1, v, i-1}^D$ for some distinct nodes u, v with $1 \leq u, v < i$, where D is the density matrix of C ;
- (iii) $c(\gamma) = c^{\gamma, n} + c(\gamma^{(n-1)})$ and $c(\gamma) - c(\xi) = \sum_{2 \leq i \leq n} c^{\gamma, i}$.

For any $n \times n$ matrix A , $A^{(U, k)}$ is the $k \times k$ upper principal submatrix of A obtained by deleting all the rows and columns indexed by $\{k + 1, k + 2, \dots, n\}$, and $A^{(L, k)}$ is the $(n - k + 1) \times (n - k + 1)$ lower principal submatrix of A obtained by deleting all the rows and columns indexed by $\{1, 2, \dots, k - 1\}$.

For any $k < i, j \leq n, i \neq j$, $A^{(U, i, j, k)}$ is the $(k + 1) \times (k + 1)$ submatrix of A obtained by deleting all the rows and columns indexed by $\{k + 1, k + 2, \dots, n\}$ except row i and column j . We call $A^{(U, i, j, k)}$ an upper minor of A . For any $1 \leq i, j < k, i \neq j$, $A^{(L, i, j, k)}$ is the $(n - k + 2) \times (n - k + 2)$ submatrix obtained by deleting all the rows and columns indexed by $\{1, 2, \dots, k - 1\}$, except row i and column j . We call $A^{(L, i, j, k)}$ a lower minor of C .

We need the following definitions introduced in [60].

A property \mathcal{P} of square matrices is upper hereditary if and only if for any $n \times n$ matrix A satisfying \mathcal{P} and any $3 \leq p < q - 1 \leq n - 1$, each of the upper minors $A^{(U, p+1, q, p)}$ and $A^{(U, q, p+1, p)}$ satisfies \mathcal{P} . It is a lower hereditary property if and only if for any $n \times n$ matrix A satisfying \mathcal{P} and $1 \leq q < p - 1 \leq n - 4$, each of the lower minors $A^{(L, p-1, q, p)}$ and

$A^{(L,q,p-1,p)}$ satisfies \mathcal{P} . Property \mathcal{P} is *hereditary* if and only if for any $n \times n$ matrix A satisfying \mathcal{P} , at least one of the following two statements is true. (a) for all $3 \leq p < q-1 \leq n-1$, each of $A^{(U,p+1,q,p)}$ and $A^{(U,q,p+1,p)}$ satisfies \mathcal{P} . (b) for all $1 \leq q < p-1 \leq n-4$, each of $A^{(L,p-1,q,p)}$ and $A^{(L,q,p-1,p)}$ satisfies \mathcal{P} . For any matrix property \mathcal{P} , $MVI(\mathcal{P})$ (minimal violator index of (\mathcal{P})) is the smallest positive integer such that there exists an $MVI(\mathcal{P}) \times MVI(\mathcal{P})$ matrix C satisfying \mathcal{P} , for which none of the optimal tours to $TSP(C)$ is pyramidal. Obviously, $MVI(\mathcal{P}) \geq 4$.

Lemma 19 [60] *Let \mathcal{P} be a hereditary matrix property with $MVI(\mathcal{P}) < \infty$. Let $n = MVI(\mathcal{P})$. Then, for any $n \times n$ cost matrix C satisfying \mathcal{P} for which none of the optimal tours to $TSP(C)$ is pyramidal, the following are true.*

- (i) *If for all $3 \leq p < q-1 \leq n-1$, each of the upper minors $C^{(U,p+1,q,p)}$ and $C^{(U,q,p+1,p)}$ satisfies \mathcal{P} , then every optimal tour to $TSP(C)$ has node $(n-1)$ as one of its peaks.*
- (ii) *If for all $1 \leq q < p-1 \leq n-4$, each of the lower minors $C^{(L,p-1,q,p)}$ and $C^{(L,q,p-1,p)}$ satisfies \mathcal{P} , then every optimal tour to $TSP(C)$ has node 2 as one of its valleys.*

In particular, if \mathcal{P} is upper hereditary (lower hereditary), then every optimal tour to $TSP(C)$ has node $(n-1)$ as one of its peaks (node 2 as one of its valleys).

The most general polynomially testable sufficiency conditions for pyramidal solvability of TSP, known to date, are (i) BK-I, (ii) BK-II(k) for some integer $4 \leq k$, (iii) BK-III, (iv) BK-IV, (v) Baki-I and (vi) Baki-II. We shall discuss these below.

Definition 20 [51, 60] *A cost matrix C with density matrix D satisfies the property Baki-Kabadi-I (BK-I) if and only if*

- (i) *for any $\{i, j, k\}$ such that $2 < i < j \leq k \leq (n-1)$, and any distinct $1 \leq u_l, v_l < l$ for each $l \in \{i, i+1, \dots, j\}$,*

$$\sum_{l=i}^j (M_{u_l, l-1, v_l, l-1}^D) + \sum_{l=i}^j (M_{l, k, l-1, l-1}^D) \geq 0;$$

- (ii) *for any $\{i, j\}$ such that $2 < i \leq j \leq (n-1)$ and any $u < i, v < (i-1), u \neq v,$*

$$M_{i, j, i-1, i-1}^D + M_{u, i-1, v, i-1}^D \geq 0;$$

- (iii) *C^T satisfies (i) and (ii).*

Theorem 21 [51, 60] *If C satisfies the property BK-I, then $TSP(C)$ is pyramidally solvable.*

Proof. If the result is not true, then $MVI(BK - I) < \infty$. Let $n = MVI(BK - I)$ and let C be a $n \times n$ matrix satisfying the property BK-I such that none of the optimal tours for $TSP(C)$ is pyramidal. Let γ be an optimal solution to $TSP(C)$, $\gamma^{-1}(n) = a$, and $\gamma(n) = b$. We assume that $a < b$. (Otherwise, replace γ by γ^{-1} and C by C^T .) Since BK-I is both upper and lower hereditary, it follows by Lemma 19 that the node $(n - 1)$ is a peak of γ and hence, $b < n - 1$. Define a tour γ' on N as follows. $\gamma'(i) = \gamma^{(b)}(i)$ for $i \in \{1, 2, \dots, b\} - \{a\}$; $\gamma'(a) = n$; $\gamma'(i) = i - 1$ for $i \in \{b + 1, \dots, n\}$, (that is, $G_{\gamma'}$ is obtained from $G_{\gamma^{(b)}}$ by replacing the arc (a, b) by the path $(a, n, n - 1, \dots, b)$). Recall that for any $2 < i \leq n$, $c^{\gamma, i} = M_{u, i-1, v, i-1}^D$, where (u, i, v) is a subpath of $\gamma^{(i)}$.

If $b < n - 2$ then,

$$\begin{aligned} c(\gamma) - c(\gamma') &= \{c(\gamma^{(b)}) + \sum_{j=b+1}^n c^{\gamma, j}\} - \{c(\gamma^{(b)}) + \sum_{j=b}^{n-1} M_{a, j, j, j}^D\} \\ &= \sum_{j=b+1}^{n-1} c^{\gamma, j} + \sum_{j=b+1}^{n-1} M_{j, n-1, j-1, j-1}^D \\ &\geq 0 \quad (\text{by condition (i) of property BK-I}) \end{aligned}$$

If $b = n - 2$, then $c^{\gamma, (n-1)} = M_{u, n-2, v, n-2}^D$ for some $v < (n - 2)$, and

$$\begin{aligned} c(\gamma) - c(\gamma') &= \{c(\gamma^{(b)}) + \sum_{j=b+1}^n c^{\gamma, j}\} - \{c(\gamma^{(b)}) + \sum_{j=b}^{n-1} M_{a, j, j, j}^D\} \\ &= c^{\gamma, n-1} + M_{n-1, n-1, n-2, n-2}^D \\ &\geq 0 \quad (\text{by condition (ii) of property BK-I}). \end{aligned}$$

Thus, γ' is an optimal solution to $TSP(C)$. But $(n - 1)$ is not a peak of γ' and this contradicts Lemma 19. This proves the theorem. ■

We refer the reader to [60] for an interesting generalization of the property BK-I.

Definition 22 [60] *For any integer $k \geq 4$, a cost matrix C satisfies the property Baki-Kabadi-II(k) (BK-II(k)) if and only if the following two conditions hold.*

- (i) *For any $k - 1 \leq p < q \leq n$ and any two sets of $(k - 2)$ distinct numbers $1 \leq i_1, i_2, \dots, i_{(k-2)} < p$ and $1 \leq j_1, j_2, \dots, j_{(k-2)} < p$ such that*

$i_u \neq j_v$, for $u \neq v$ and $|\{i_1, i_2, \dots, i_{(k-2)}\} \cup \{j_1, j_2, \dots, j_{(k-2)}\}| \geq k - 1$, the $k \times k$ submatrix C' of C , corresponding to the rows and columns $\{i_1, i_2, \dots, i_{(k-2)}, p, p + 1\}$ and $\{j_1, j_2, \dots, j_{(k-2)}, p, q\}$ or $\{i_1, i_2, \dots, i_{(k-2)}, p, q\}$ and $\{j_1, j_2, \dots, j_{(k-2)}, p, p + 1\}$ is such that there exists an optimal solution γ to $TSP(C')$ in which the node $(k - 1)$ is adjacent to the node k , (that is, either $\gamma(k) = k - 1$ or $\gamma(k - 1) = k$).

(ii) For any $r \leq k$, $TSP(C^{(U,r)})$ is pyramidally solvable.

Theorem 23 [60] *If cost matrix C satisfies the property BK-II(k) for some integer $4 \leq k \leq n$, then $TSP(C)$ is pyramidally solvable.*

Proof. If the result is not true then there exist an integer $k \geq 4$ such that $MVI(BK - II(k)) < \infty$. Let $n = MVI(BK - II(k))$ and let C be an $n \times n$ cost matrix satisfying the property BK-II(k) for which there does not exist any optimal solution to $TSP(C)$ that is pyramidal. Suppose γ is an optimal solution to $TSP(C)$. Since BK-II(k) is upper hereditary, by Lemma 19, $(n - 1)$ is a peak of γ . Choose a set $\{i_1, i_2, \dots, i_{k-2}\}$ of distinct nodes in $\{1, 2, \dots, n - 2\}$ including the nodes $\gamma^{-1}(n - 1)$ and $\gamma^{-1}(n)$. Remove the arcs $\{(i_j, \gamma(i_j)) : 1 \leq j \leq k - 2\} \cup \{(n, \gamma(n)), (n - 1, \gamma(n - 1))\}$ from G_γ to get $(k - 2)$ node-disjoint, paths $\{P_{u_i, v_i} : 1 \leq i \leq k - 2\}$, in addition to isolated nodes n and $(n - 1)$. If we contract these $k - 2$ paths in G_γ to nodes $\{1', 2', \dots, (k - 2)'\}$, respectively, the resultant digraph defines a tour γ' on the node set $\{1', 2', \dots, (k - 2)', (n - 1), n\}$. Let C' be the submatrix of C corresponding to the rows $\{v_1, v_2, \dots, v_{(k-1)}, (n - 1), n\}$ and columns $\{u_1, u_2, \dots, u_{(k-1)}, (n - 1), n\}$. Let us index the rows/columns of C' by $\{1', 2', \dots, (k - 2)', (n - 1), n\}$, in that order. Then, by property BK-II(k), there exists an optimal solution ψ' for $TSP(C')$ such that node n is adjacent to node $(n - 1)$ in ψ' . In $G_{\psi'}$, replace the nodes $\{1', 2', \dots, (k - 2)'\}$ by their respective paths to get a digraph corresponding to a tour, say ψ , on N . Then $c(\gamma) - c(\psi) = c'(\gamma') - c'(\psi') \leq 0$. Thus, ψ is an optimal solution to $TSP(C)$. But node $(n - 1)$ is not a peak of ψ , a contradiction. This proves the theorem. ■

The following interesting result is proved in [60].

Theorem 24 [60] *For any $k \geq 4$, if a matrix satisfies the property BK-II(k), then it satisfies the property BK-II($k + 1$).*

We shall now give without proofs the remaining sufficiency conditions, BK-III, BK-IV, Baki-I and Baki-II. For short proofs of sufficiency of these conditions the reader is referred to [52, 54, 58].

Definition 25 A cost matrix C with density matrix D satisfies property Baki-Kabadi-III (BK-III) if and only if

(i) for all $1 \leq i < j < k \leq v \leq n - 1$,

$$\begin{aligned} M_{i,k,j,v}^D - M_{k-1,k,k,v}^D &\geq 0, \\ M_{i,v,j,k}^D - M_{k-1,v,k,k}^D &\geq 0, \\ M_{i,k,j,v}^D - M_{k,k,k-1,v}^D &\geq 0 \text{ and} \\ M_{i,v,j,k}^D - M_{k,v,k-1,k}^D &\geq 0; \end{aligned}$$

(ii) C^T satisfies (i).

Property BK-III is a minor generalization of a sufficiency condition for pyramidal solvability given in [58].

Theorem 26 If C satisfies property BK-III, then $TSP(C)$ is pyramidally solvable.

Definition 27 [58] A cost matrix C with density matrix D satisfies property Baki-Kabadi-IV (BK-IV) if and only if

(i) for all $1 \leq i < j < u, v \leq n - 1$,

$$\begin{aligned} M_{i,j-1,j,u}^D + M_{j,v,j,j}^D &\geq 0 \text{ and} \\ M_{j-1,u,j,j}^D &\geq 0; \end{aligned}$$

(ii) C^T satisfies (i).

Definition 28 [54] A cost matrix C satisfies property Baki-I if and only if for any $4 \leq i \leq k \leq n$, $1 \leq a, b < i - 1$, $a \neq b$, $\{u, v\} = \{i - 2, i - 1\}$, and optimal pyramidal tours γ^1 and γ^2 for cost matrices $C^1 = A^{(L,a,b,i)}$ and $C^2 = A^{(L,u,v,i)}$ respectively, where $A = C^{(U,k)}$,

$$c^2(\gamma^2) - c^1(\gamma^1) \leq c_{uv} - c_{ab}.$$

Theorem 29 [54, 58] If C satisfies property Baki-I or BK-IV, then $TSP(C)$ is pyramidally solvable.

For a cost matrix C with density matrix D , we define an $n \times n$ matrix R^C as follows:

$$r_{ij}^C = \begin{cases} 0 & \text{if } |i - j| \leq 1 \\ r_{i,j-1}^C & \text{if } i < j - 1 \text{ and } M_{i,j-1,j-1,j-1}^D = m_{j-1}^* \\ 1 & \text{if } i < j - 1 \text{ and } M_{i,j-1,j-1,j-1}^D > m_{j-1}^* \\ r_{i-1,j}^C & \text{if } j < i - 1 \text{ and } M_{i-1,i-1,j,i-1}^D = m_{i-1}^* \\ 1 & \text{if } j < i - 1 \text{ and } M_{i-1,i-1,j,i-1}^D > m_{i-1}^* \end{cases}$$

where $m_i^* = \min\{M_{u,i,v,i}^D : u \neq v, 1 \leq u, v \leq i\}$.

Definition 30 [52, 53] *A cost matrix C satisfies property Baki-II if and only if there exists a pyramidal tour γ on N with $r^C(\gamma) = 0$, (that is, the cost of γ with respect to the matrix R^C is 0).*

Theorem 31 [53] *For a cost matrix C with density matrix D , and a pyramidal tour γ on N , the following two statements are equivalent.*

- (i) $r^C(\gamma) = 0$ (and therefore C satisfies the property Baki-II).
- (ii) For all $2 < i \leq n$, $c^{\gamma,i} = \min\{M_{u,i-1,v,i-1}^D : u \neq v, 1 \leq u, v < i\}$.

Hence, if these statements hold, then γ is an optimal solution to $TSP(C)$ and therefore, $TSP(C)$ is pyramidally solvable.

To the best of our knowledge, all other known sufficiency conditions are special cases of one or more of the above. We shall mention some of them.

If the density matrix D of C is non-negative, then it satisfies the conditions BK-I, BK-III, BK-IV and Baki-I.

Rana et al [34] give an interesting application of this in the context of m -machine flow shop problem in which there is no storage space between consecutive machines (see Section 3.4). They show that if the processing times $\{p_{ij} : i \in \{1, \dots, n\}; j \in \{1, \dots, m\}\}$ of n given jobs on the m machines can be semi-ordered, (that is, there exists a permutation σ on N such that $p_{\sigma(i),j} \leq p_{\sigma(i+1),j}$ for all i and j), then the problem can be expressed as $TSP(C)$ with the density matrix D of C non-negative.

As interesting special cases of matrices satisfying property Baki-II, we have the classes of Kalmanson matrices [491], Supnick matrices [776], generalized Supnick matrices [144], generalized Kalmanson matrices [144] and the matrices satisfying Demidenko condition-II [252]. We define these below.

A symmetric, cost matrix C , with density matrix D , is called a *Kalmanson matrix* [491] if and only if for all $1 \leq i \leq j < k < u \leq n$,

$$M_{i,u,j+1,k}^D \geq 0 \text{ and } M_{i,j,k+1,u}^D \leq 0.$$

Matrix C is called a *Supnick matrix* [776] if and only if for all $1 \leq i < j < u < n$,

$$M_{i,u,j,j}^D \geq 0 \text{ and } M_{i,j-1,j+1,u}^D \geq 0.$$

Corollary 32 [491] *If C is a Kalmanson matrix, then $\gamma = (1, 2, 3, \dots, n-1, n, 1)$ is an optimal solution to $TSP(C)$.*

Proof. Let D be the density matrix of C . Then, for any $1 \leq u < v < i \leq n$,

$$\begin{aligned}
 c^{\gamma,i} &= M_{i-1,i-1,1,i-1}^D \\
 M_{v,i-1,u,i-1}^D - c^{\gamma,i} &= M_{v,i-1,u,i-1}^D - M_{i-1,i-1,1,i-1}^D \\
 &= \sum_{j=v}^{i-2} M_{j,j,u,i-1}^D - M_{i-1,i-1,1,u-1}^D \\
 &\geq 0,
 \end{aligned}$$

and by symmetry,

$$M_{u,i-1,v,i-1}^D - c^{\gamma,i} = M_{v,i-1,u,i-1}^D - M_{i-1,i-1,1,i-1}^D \geq 0.$$

Thus, C satisfies the property Baki-II with $r^C(\gamma) = 0$. ■

We shall further study the class of Kalmanson matrices in Section 6.

A Supnick matrix also satisfies properties BK-III. Corollary 33 below is proved in [776] and it is shown in [53] that it follows from the fact that a Supnick matrix satisfies the property Baki-II.

Corollary 33 [776] *If C is a Supnick matrix, then $\gamma = (1, 3, 5, 7, \dots, 6, 4, 2, 1)$ is an optimal solution to $TSP(C)$.*

The proof of Corollary 33 follows along the same lines as the proof of Corollary 32.

A cost matrix C with density matrix D is a *generalized Supnick matrix* [144] if and only if

- (i) for all $1 \leq i < i + 1 < j < n$, $d_{ij} \geq 0$ and $d_{ji} \geq 0$;
- (ii) for all $1 \leq i \leq n - 3$, $M_{i,i+1,i+1,i+2}^D \geq 0$ and $M_{i+1,i+2,i,i+1}^D \geq 0$;
- (iii) for all i even with $2 \leq i \leq n - 2$, $d_{i,i+1} - d_{i+1,i} \geq 0$;
- (iv) for all i odd with $1 \leq i \leq n - 2$, $d_{i+1,i} - d_{i,i+1} \geq 0$.

The matrix C is a *generalized Kalmanson matrix* [144] if and only if

- (i) for all $1 \leq i < i + 1 < j < n$, $d_{ij} \geq 0$ and $d_{ji} \geq 0$;
- (ii) for all $1 \leq i \leq n - 3$, $M_{i,i+1,i+1,i+2}^D \geq 0$ and $M_{i+1,i+2,i,i+1}^D \geq 0$;
- (iii) for all i even with $2 \leq i \leq n - 2$, $d_{i,i+1} - d_{i+1,i} \geq 0$;
- (iv) for all i odd with $1 \leq i \leq n - 2$, $d_{i+1,i} - d_{i,i+1} \geq 0$.

We say that the matrix C satisfies *Demidenko condition-II* [252] if and only if

- (i) $d_{ij} \geq 0$ for all $1 \leq i, j < n$, $|i - j| > 1$;
- (ii) $M_{i-1,i+1,i,i}^D \geq 0$ for all $3 \leq i \leq n - 2$, i odd;
- (iii) $M_{i,i,i-1,i+1}^D \geq 0$ for all $2 \leq i \leq n - 2$, i even;

- (iii) $d_{i,i+1} - d_{i+1,i} \geq 0$ for all $2 \leq i \leq n - 2$, i even;
- (iv) $d_{i+1,i} - d_{i,i+1} \geq 0$ for all $1 \leq i \leq n - 2$, i odd.

Corollary 34 below can be proved along the same lines as Corollary 33.

Corollary 34 [144] *If C is a generalized Supnick matrix or if it satisfies the Demidenko condition-II, then $(1, 3, 5, 7, \dots, 6, 4, 2, 1)$ is an optimal solution to $TSP(C)$. If C is a generalized Kalmanson matrix, then $(1, 2, 3, \dots, n - 1, n, 1)$ is an optimal solution to $TSP(C)$.*

In [52], an example is given of a matrix which satisfies the property Baki-II, but none of the special cases mentioned above.

We give below other interesting known sufficiency conditions.

A symmetric matrix C with density matrix D is a *Van der Veen matrix* [803] if and only if for all $1 \leq i < j < k \leq n$,

$$M_{i,j,k}^D \geq 0.$$

Corollary 35 [54] *Every Van der Veen matrix satisfies property Baki-I.*

Proof. Let C be a Van der Veen matrix with density matrix D . For any $4 \leq i \leq k \leq n$ and $1 \leq a < b < i - 1$, let γ^1 be an optimal pyramidal tour for the cost matrix $C^1 = A^{(L,a,b,i)}$, where $A = C^{(U,k)}$ and the first row/column of C^1 is indexed by $(i - 1)$. Let $\gamma^1(i - 1) = x$ and $\gamma^1(y) = i - 1$. Let $u = i - 2$, $v = i - 1$, $C^2 = A^{(L,u,v,i)}$, $C^3 = A^{(L,v,u,i)}$ and let the first row/column of each of C^2 and C^3 also be indexed by $(i - 1)$. Let γ^2 and γ^3 be optimal pyramidal tours for C^2 and C^3 , respectively.

Case 1: $(i - b)$ is odd: In this case,

$$\begin{aligned} c^1(\gamma^1) - c^2(\gamma^2) - c_{ab} + c_{uv} &\leq c^1(\gamma^1) - c^2(\gamma^1) - c_{ab} + c_{uv} \\ &= M_{a,b,b,x-1}^D + M_{b+1,y-1,b,b+1}^D \\ &+ \sum_{j=1}^{(i-b-3)/2} (M_{b+2j-1,b+2j,b+2j,x-1}^D + M_{b+2j+1,y-1,b+2j,b+2j+1}^D) \geq 0. \end{aligned}$$

By symmetry,

$$\begin{aligned} c^1(\gamma^1) - c^3(\gamma^3) - c_{ab} + c_{vu} &\leq c^1(\gamma^1) - c^3((\gamma^1)^{-1}) - c_{ab} + c_{vu} \\ &= c^1(\gamma^1) - c^2(\gamma^1) - c_{ab} + c_{uv} \geq 0. \end{aligned}$$

Case 2: $(i - b)$ is even: In this case,

$$\begin{aligned} c^1(\gamma^1) - c^3(\gamma^3) - c_{ab} + c_{vu} &\leq c^1(\gamma^1) - c^3(\gamma^1) - c_{ab} + c_{vu} \\ &= M_{a,b,b,x-1}^D + M_{b+1,y-1,b,b+1}^D \\ &+ \sum_{j=1}^{(i-b-2)/2} (M_{b+2j-1,y-1,b+2j-2,b+2j-1}^D + M_{b+2j-1,b+2j,b+2j,x-1}^D) \geq 0. \end{aligned}$$

By symmetry,

$$\begin{aligned} c^1(\gamma^1) - c^2(\gamma^2) - c_{ab} + c_{uv} &\leq c^1(\gamma^1) - c^2((\gamma^1)^{-1}) - c_{ab} + c_{uv} \\ &= c^1(\gamma^1) - c^3(\gamma^1) - c_{ab} + c_{vu} \geq 0. \end{aligned}$$

■

It is shown in [54] that the class of Van der Veen matrices is a proper subclass of the matrices satisfying property Baki-II.

A cost matrix C with density matrix D satisfies *Demidenko condition-I* if and only if

(i) for all $1 \leq i, j \leq k < u < n, i \neq j,$

$$M_{i,k,j,k}^D + M_{k+1,u,k,k}^D \geq 0;$$

(ii) C^T satisfies (i).

The class of matrices satisfying Demidenko condition-I can be easily seen to be a proper subclass of the matrices satisfying property BK-I. This resolves the question raised in [361] about a simple, short proof of sufficiency of Demidenko condition-I for pyramidal solvability of TSP.

Matrix C is called an *Aizenshtat-Maksimovich matrix (AM-matrix)* [9] if and only if for all $1 \leq i < n, c_{i,i+1} = c_{i+1,i}$ and for all $1 \leq i < u < v \leq n, c_{iv} \geq \max\{c_{iu}, c_{uv}\}$ and $c_{vi} \geq \max\{c_{vu}, c_{ui}\}$. The matrix C is a *Klyaus matrix* [507] if and only if for all $1 \leq i < j < v \leq, c_{ij} + c_{ji} \geq 0, c_{ij} + c_{jv} \leq c_{iv}$ and $c_{vj} + c_{ji} \leq c_{vi}$. If C is a symmetric matrix, then it is of the type *Suprunenko-I* [777] if and only if for all $1 \leq i, j, u, v \leq n$ such that $|i - j| < |u - v|, c_{ij} < c_{uv}$.

It follows from the definition of Demidenko condition-I that the matrices of each of the types Suprunenko-I, AM and Klyaus satisfy the Demidenko condition-I [251]. Some additional classes of sufficiency conditions on C are reported in [148, 801, 819, 820]. In each of these cases, the cost matrix, its transpose or its reverse satisfies property BK-II(4).

A Sufficiency condition that simultaneously generalizes properties BK-I and BK-II(k) is given in [60]. The same idea can be used to obtain a sufficiency condition that simultaneously generalizes properties BK-I, BK-II(k), BK-III, Baki-I and Baki-II.

4.2. Some necessary and some sufficiency conditions on D for validity of GG scheme with arbitrary starting permutation π

We now discuss known necessary conditions and sufficiency conditions on a matrix D under which, for any cost matrix C and any permutation π such that D is the density matrix of C^π , GG scheme with π as the starting permutation produces an optimal tour.

It seems to be a commonly held belief that it is a necessary condition that π should be an optimal solution to the corresponding assignment problem on C . The following counter-example to this is given in [481].

Example 1. Let $D = \begin{bmatrix} -1 & 5 \\ 5 & 5 \end{bmatrix}$. Then, for any matrix C and permutation π such that D is the density matrix of C^π , $\pi \circ \beta_1$ is the unique optimal solution to the assignment problem on C and the GG scheme with π as the starting permutation produces an optimal solution to $TSP(C)$.

Theorem 36 seems to be the only result known on necessary conditions.

Theorem 36 [481] *Suppose a matrix D is such that for any cost matrix C and any permutation π such that D is the density matrix of C^π , GG scheme with π as the starting permutation produces an optimal tour. Then D satisfies each of the following conditions.*

- (i) For all $1 \leq i, j \leq n - 1$, such that $|i - j| > 1$, $d_{ii} + d_{jj} \geq 0$.
- (ii) For all $1 \leq i \leq n - 2$, $D[\{i, i + 1\}] \geq 0$.
- (iii) For all $1 \leq i < j \leq n - 1$, $M_{i,j,i,j}^D \geq 0$.
- (iv) For any principal submatrix D' of D on a consecutive subset of its rows and columns, and any cost matrix C having D' as its density matrix, $TSP(C)$ is pyramiddally solvable.

Let us now investigate sufficiency conditions on D . We first introduce some more definitions and basic results.

Let ψ be a circuit and $S = \{i_1, i_2, \dots, i_k\}$ the node set of its only non-trivial subtour \mathfrak{C} , where $i_1 < i_2 < \dots < i_k$. Let $\omega_0 = \xi$, the identity permutation, $S_0 = \{i_1, i_2, \dots, i_k\}$, and $\psi(i_1) = i_u$. Define $\omega_1 = \omega_0 \circ \alpha_{i_1, i_u}$, and $S_1 = \{i_2, \dots, i_k\}$. If $k = 2$, then $\omega_1 = \psi$; else, $\omega_1(i_1) = \psi(i_1) = i_u$, and $\omega_1(j) \neq \psi(j)$ for all $j \in S_1$. Recursively, define $\omega_2, \omega_3, \dots, \omega_{k-1}$ and S_2, \dots, S_{k-2} as follows. For any $1 \leq j < k - 1$, $S_j = \{i_{j+1}, \dots, i_k\}$, and $\omega_{j+1} = \omega_j \circ \alpha_{i_{j+1}, i_l}$, where $l > j + 1$ is such that $\psi(i_{j+1}) = \omega_j(i_l)$. Then, $\omega_{k-1} = \psi$.

Let π be an arbitrary permutation on N with m non-trivial subtours of sizes n_1, n_2, \dots, n_m . Let $r = \sum_{i=1}^m (n_i - 1)$. Combining this with observations 11 and 12, we get the following fact.

Fact 37 *There exist $1 \leq i_1 < i_2 < \dots < i_r \leq n$ and $i_u < j_u \in N$ for all $1 \leq u \leq r$ such that $\pi = \alpha_{i_1, j_1} \circ \alpha_{i_2, j_2} \circ \dots \circ \alpha_{i_r, j_r}$.*

The observations below follow from Observation 13 and Fact 37.

Observation 38 Let $\{\alpha_{i_1, j_1}, \alpha_{i_2, j_2}, \dots, \alpha_{i_k, j_k}\}$ be a set of transpositions such that $\gamma = \pi \circ \alpha_{i_1, j_1} \circ \alpha_{i_2, j_2} \circ \dots \circ \alpha_{i_k, j_k}$ is a tour. Then the following are true.

- (i) $k \geq (m - 1)$ and $k \equiv (m - 1) \pmod{2}$.
- (ii) Let $X = \cup\{i_\ell, j_\ell : \ell \in \{1, \dots, k\}\}$. Then every connected component of G_π contains at least one element of X .
- (iii) Define a graph $G' = (M, E')$ as : $M = \{1, 2, \dots, m\}$ and $E' = \{e^\ell = (u, v) : i_\ell \text{ and } j_\ell \text{ belong to the connected components } u \text{ and } v \text{ of } G_\pi, \text{ respectively; } \ell \in \{1, \dots, k\}\}$. Then, G' is a connected graph. Let $G'' = (M, E'')$ be a spanning tree of G' . Suppose $E'' = \{e^{[i]} : i \in \{1, \dots, m - 1\}\}$. Then $\gamma' = \pi \circ \alpha_{i_1, j_{[1]}} \circ \dots \circ \alpha_{i_{[m-1]}, j_{[m-1]}}$ is a tour.

Observation 39 Let ψ_1 and ψ_2 be two permutations on N such that $\pi \circ \psi_1$ and $\pi \circ \psi_2$ form tours on N . Suppose ψ_1 has m_1 nontrivial subtours of sizes u_1, u_2, \dots, u_{m_1} and ψ_2 has m_2 non-trivial subtours of sizes v_1, v_2, \dots, v_{m_2} . Then, $(\sum_{i=1}^{m_1} (u_i - 1)) \equiv (\sum_{i=1}^{m_2} (v_i - 1)) \equiv (m - 1) \pmod{2}$.

For any $N \supseteq X = \{i_1, i_2, \dots, i_k\}$, where $1 \leq i_1 < i_2 < \dots < i_k \leq n$, we call the set $\{i_1, i_1 + 1, \dots, i_k - 1\}$ the *range* of X and denote it by $[i_1, i_k - 1]$. For each $1 \leq u < k$, we call the set $\{i_u, i_u + 1, \dots, i_{u+1} - 1\}$ a *region* of X . If X is the node set of a subtour \mathfrak{C} , then we call the range and the regions of X as, respectively, the *range and the regions of \mathfrak{C}* . For any $S \subseteq \{1, 2, \dots, n - 1\}$, we denote by E_S the edge set $\{(i, i + 1) : i \in S\}$.

Lemma 40 [51, 483] For an arbitrary permutation π on N and an arbitrary set $S^0 \subseteq \{1, 2, \dots, n - 1\}$, let G^0 be a mixed pseudograph on N with arc set E_π and edge set E_{S^0} . Let $X \subseteq N$ be such that each connected component of G^0 contains at least one element of X . Then there exists $S \subseteq N$ such that

- (i) each element of S lies in the range of X ;
- (ii) each region of X contains at the most one element of S ; and
- (iii) the mixed pseudograph $G' = (N, E_\pi \cup E_{S^0 \cup S})$ is connected.

Proof. Suppose the mixed pseudograph G^0 has m connected components. We shall prove the result by induction on m .

For $m = 1$, the result holds trivially. Suppose the result is true for all $m < k$ for some $k \geq 2$. Let us consider the case $m = k$.

Let $Y \subseteq X$ be such that each connected component of G^0 contains precisely one element of Y . Let $Y = \{i_1, i_2, \dots, i_k\}$, where $i_1 < i_2 < \dots < i_k$. Since i_1 and i_2 lie in different connected components of G^0 , there exists $i_1 \leq u < i_2$ such that i_1 and u belong to the same connected component and $u + 1$ belongs to a different connected component. Let $S^1 = S^0 \cup \{u\}$, and $Y' = Y - \{i_1\}$. Then the mixed pseudograph, $\tilde{G} = (N, E_\pi \cup E_{S^1})$ has $k - 1$ connected components and each element of Y' lies in a different connected component of \tilde{G} . Hence, it follows by induction hypothesis that there exists $S' \subseteq N$ such that (i) each element of S' lies in the range of Y' ; (ii) each region of Y' contains at the most one element of S' ; and (iii) for $\bar{S} = S^1 \cup S' = S^0 \cup \{u\} \cup S'$, the mixed pseudograph $G' = (N, E_\pi \cup E_{\bar{S}})$ is connected. Thus, $S = S' \cup \{u\}$ satisfies the required conditions. This proves the lemma. ■

In Lemma 40, if $S^0 = \emptyset$ and the set S is chosen as a minimal set satisfying the conditions of the lemma, then $E_p^\pi[S]$ (see definition 10) is the edge set of a spanning tree of G_p^π and therefore, for any ordering $(j_1, j_2, \dots, j_{m-1})$ of elements of S , $\pi \circ \beta_{j_1} \circ \beta_{j_2} \circ \dots \circ \beta_{j_{m-1}}$ is a tour.

As Corollaries to Lemma 40, we get the following.

Corollary 41 [51, 483] *Let ρ and π be arbitrary permutations on N and let $S^0 \subseteq \{1, 2, \dots, n - 1\}$ be such that the mixed pseudograph $G^0 = (N, E_\rho \cup E_{S^0})$ is connected. Let $\rho = \pi \circ \psi$. Suppose ψ has v non-trivial subtours, $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_v$, on node sets N^1, N^2, \dots, N^v , respectively. Then there exists $S \subseteq N$ and a partition $\{S_1, S_2, \dots, S_v\}$ of elements of S such that:*

- (i) *the mixed pseudograph $G' = (N, E_\pi \cup E_{S^0 \cup S})$ is connected, and*
- (ii) *for all $1 \leq i \leq v$, each element of S_i lies in the range of N^i and each region of N^i contains at the most one element of S_i .*

Corollary 42 *Let π be an arbitrary permutation on N and let γ be an arbitrary tour on N . Let $\gamma = \pi \circ \psi$. Suppose ψ has v non-trivial subtours, $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_v$, on node sets N^1, N^2, \dots, N^v , respectively. For all $1 \leq i \leq v$, let $|N^i| = n_i$. Then there exists $S \subseteq N$ and a partition $\{S_1, S_2, \dots, S_v\}$ of elements of S such that:*

- (i) *$E_p^\pi[S]$ is the edge set of a spanning tree of G_p^π ;*
- (ii) *for all $1 \leq i \leq v$, each element of S_i lies in the range of N^i and each region of N^i contains at the most one element of S_i ;*
- (iii) *$|S| \equiv (\sum_{i=1}^v (n_i - 1)) \pmod{2}$.*

For any permutation, ψ on N with non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_v$ having respective ranges $[i_1, j_1 - 1], [i_2, j_2 - 1], \dots, [i_v, j_v - 1]$, we define

the intersection graph of ψ as $G_\psi^I = (N_\psi, E_\psi^I)$, where $N_\psi = \{1, 2, \dots, v\}$, and $E_\psi^I = \{(i, j) : \text{ranges of the subtours } \mathcal{C}_i \text{ and } \mathcal{C}_j \text{ intersect}\}$.

Corollary 43 *Let π be an arbitrary permutation on N and let γ be an arbitrary tour on N . Let $\gamma = \pi \circ \psi$. Suppose ψ has v non-trivial subtours, and G_ψ^I , the intersection graph of ψ , has r connected components. Let $N_\psi^1, N_\psi^2, \dots, N_\psi^r$ be the node sets of the r connected components of G_ψ^I . For all $1 \leq i \leq r$, let $|N_\psi^i| = v_i$ and let X_ψ^i be the union of the node sets of all the non-trivial subtours of ψ corresponding to the nodes in N_ψ^i . Then there exists $S \subseteq N$ and a partition $\{S_1, S_2, \dots, S_r\}$ of its elements such that:*

- (i) $E_p^\pi[S]$ is the edge set of a spanning tree of G_p^π ;
- (ii) for $1 \leq i \leq r$, every element of S_i lies in the range X_ψ^i , every region of X_ψ^i contains at most one element of S_i and $|S_i| \leq (|X_\psi^i| - v_i)$; and
- (iii) $|S| \equiv ((\sum_{i=1}^r |X_\psi^i|) - v) \pmod{2}$.

We shall now give sets of sufficiency conditions on an $(n - 1) \times (n - 1)$ matrix D under which for any cost matrix C and any permutation π on N such that the density matrix of C^π is D , the GG Scheme with π as the starting permutation produces an optimal tour to $TSP(C)$.

Consider a pair (ψ, S) of a permutation on N and a subset of N satisfying the following condition. Suppose ψ has v non-trivial subtours on node sets N^1, N^2, \dots, N^v . Then, there exists a partition $\{S_1, S_2, \dots, S_v\}$ of S such that,

- (i) For each i , each element of S_i lies in the range of N^i and each region of N^i contains at the most one element of S_i .
- (ii) $|S| \equiv (\sum_{i=1}^v (|N^i| - 1) \pmod{2})$, where $S = \cup_{i=1}^v S_i$.

We say that a matrix D satisfies *property GG1* if and only if for any pair (ψ, S) satisfying the above condition, $D(\psi) \geq D[S]$.

Theorems 44 is a minor modification of results reported in [51, 479, 483, 484].

Theorem 44 *Suppose matrix D satisfies the property GG1. Then, for any cost matrix C and any permutation π such that D is the density matrix of C^π the GG scheme, with π as the starting permutation, produces an optimal solution to $TSP(C)$.*

Proof. Suppose D satisfies property GG1. Let a cost matrix C and a permutation π be such that D is the density matrix of C^π and let γ

be an optimal solution to the $TSP(C)$. Let $\psi = \pi^{-1} \circ \gamma$. Suppose ψ has v non-trivial subtours on node sets N^1, N^2, \dots, N^v . By Corollary 42, there exists a subset S of N with partition $\{S_1, S_2, \dots, S_v\}$ such that : (i) for each i , every element of S_i lies in the range N^i and every region of N^i contains at the most one element of S_i ; (ii) $E_p^\pi[S]$ is the edge set of a spanning tree of the patching pseudograph G_p^π ; and therefore, (iii) $|S| \equiv ((\sum_{i=1}^v (|N^i| - 1) \pmod 2)$. Let (i_1, i_2, \dots, i_r) be an ordering of the elements of S for which $D(\beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_r}) = D[S]$. Then, $\gamma^* = \pi \circ \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_r}$ is a tour on N and $c(\gamma^*) - c(\gamma) = (c(\gamma^*) - c(\pi)) - (c(\gamma) - c(\pi)) = D[S] - D(\psi) \leq 0$. Hence, γ^* is an optimal tour. This proves the theorem. ■

We now give our second sufficiency condition. Consider a pair (ψ, S) of a permutation on N and a subset of N satisfying the following condition. Suppose ψ has ℓ non-trivial subtours and G_ψ^I has r connected components of sizes v_1, v_2, \dots, v_r . Let $X_\psi^1, X_\psi^2, \dots, X_\psi^r$ be the unions of the node sets of the non-trivial subtours of ψ corresponding respectively to nodes of the r connected components of G_ψ^I . Then there exists a partition $\{S_1, S_2, \dots, S_r\}$ of S such that,

- (i) for any $1 \leq i \leq r$, every element of S_i lies in the range X_ψ^i , every region of X_ψ^i contains at the most one element of S_i and $|S_i| \leq (|X_\psi^i| - v_i)$, and
- (ii) $|S| \equiv ((\sum_{i=1}^r |X_\psi^i|) - \ell) \pmod 2$.

We say that a matrix D satisfies *property GG2* if and only if for any pair (ψ, S) satisfying the above condition, $D(\psi) \geq D[S]$.

Theorem 45 below is again a minor modification of results reported in [51, 479, 483, 484].

Theorem 45 *Suppose matrix D satisfies property GG2. Then, for any cost matrix C and any permutation π on N such that D is the density matrix of C^π , the GG scheme with π as the starting permutation produces an optimal solution to $TSP(C)$.*

The theorem can be proved along the same lines as Theorem 44 using Corollary 43.

It can be shown that every non-negative matrix satisfies both properties GG1 and GG2. A cost matrix with a non-negative density matrix is often referred to in literature as a Monge matrix [450, 603].

Properties GG1 and GG2 do not seem to be polynomially testable. The following subclass of matrices satisfying property GG2 is introduced in [51, 483].

For any matrix D and any $1 \leq u, v \leq i \leq w, x \leq n - 1$, let

$$\mathcal{A}^D(u, v, i, x, w) = M_{u,i,i,w}^D + M_{i,x,v,i}^D - d_{ii}.$$

We say that a matrix D satisfies property *GG3* if and only if

- (i) $\mathcal{A}^D(u, v, i, x, w) \geq 0$ for all $1 \leq u, v \leq i \leq w, x \leq n - 1$.
- (ii) For any pair (C, π) of a cost matrix C and a permutation π on N , such that D is the density matrix of C^π , and any principal submatrix C' of C^π on a set of its consecutive rows/columns, $TSP(C')$ is pyramidally solvable.

Theorem 46 [51, 483] *If D satisfies property *GG3*, then it satisfies property *GG2*.*

Outline of Proof Consider an arbitrary permutation ψ on N . Suppose ψ has ℓ non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_\ell$ and G_ψ^I has r connected components of sizes v_1, v_2, \dots, v_r . Let $X_\psi^1, X_\psi^2, \dots, X_\psi^r$ be unions of the node sets of the non-trivial subtours of ψ corresponding, respectively, to nodes of the r connected components of G_ψ^I . Let S be any subset of N with a partition $\{S_1, S_2, \dots, S_r\}$, such that for any $1 \leq i \leq r$, every element of S_i lies in the range of X_ψ^i , every region of X_ψ^i contains at the most one element of S_i and $|S_i| \leq (|X_\psi^i| - v_i)$.

For any $1 \leq i < j \leq \ell$ such i and j belong to the same connected component of G_ψ^I , we can find, using condition (i) of property *GG3*, nodes u, v of \mathfrak{C}_i and \mathfrak{C}_j , respectively, such that the ranges of $\{u, \psi(u)\}$ and $\{v, \psi(v)\}$ intersect and $D(\psi \circ \alpha_{u,v}) \leq D(\psi)$. By repeating this process, we get a permutation ϱ , such that $D(\varrho) \leq D(\psi)$ and the node sets of the non-trivial subtours of ϱ are $X_\psi^1, X_\psi^2, \dots, X_\psi^r$.

For any $1 \leq i \leq r$ and any $u \in S_i - X_\psi^i$, we can obtain, using condition (i) of property *GG3*, a permutation θ on N such that (i) the subtour of ϱ on node set X_ψ^i is replaced in θ by two subtours on node sets $(X_\psi^i \cap \{j : j < u\}) \cup \{u\}$ and $X_\psi^i \cap \{j : j > u\}$, respectively, while all the other subtours of θ are the same as the other subtours of ϱ ; and (ii) $D(\theta) \leq D(\varrho)$. By repeating this process, we get a permutation ω , dense on S , such that $D(\omega) \leq D(\psi)$. By condition (ii) of property *GG3*, it now follows that $D(\omega) \geq D[S]$. This proves the result.

Property *GG4* below is also a special case of property *GG2* and it gives the most general known, polynomially testable sufficiency condition.

For any $n \times n$ matrix A and any $1 \leq i, j \leq q \leq u, v \leq n$,

$$\begin{aligned} \mathfrak{F}_U^A(i, q, u, j, q, v) &= M_{i,u,q,v}^A + M_{q,u,j,q-1}^A \text{ and} \\ \mathfrak{F}_L^A(i, q, u, j, q, v) &= M_{i,u,j,q}^A + M_{i,q,q+1,v}^A. \end{aligned}$$

A matrix D satisfies property GG4 if and only if the following five conditions hold.

- (i) For all $1 \leq i, j \leq q \leq u, v \leq n - 1$, except for the case $\{u = v = q$ and $i = j \neq q - 1\}$, $\mathfrak{F}_U^D(i, q, u, j, q, v) \geq 0$.
- (ii) For all $1 \leq i, j \leq q \leq u, v \leq n - 1$, except for the case $\{i = j = q$, and $u = v \neq q + 1\}$, $\mathfrak{F}_L^D(i, q, u, j, q, v) \geq 0$.
- (iii) For any $1 \leq i \leq n - 2$, $D[\{i, i + 1, i + 2\}] \geq 0$.
- (iv) for any $1 \leq i \leq u \leq j < n$, and any $1 \leq k < i$ or $j < k < n$, $M_{i,j,u,u}^D + d_{kk} \geq 0$ and $M_{u,u,i,j}^D + d_{kk} \geq 0$;
- (v) For any pair (C, π) of a cost matrix C and a permutation π on N , such that D is the density matrix of C^π , and any principal submatrix C' of C^π on a set of its consecutive rows/columns, $TSP(C')$ is pyramidally solvable.

Theorem 47 [481] *If D satisfies property GG4, then it satisfies property GG2.*

For proof of Theorem 47, we refer the reader to [481].

Conditions (i), (ii), (iii), and (iv) of property GG4 can be tested in $O(n^4)$. Testing, in general, whether for a given density matrix D the corresponding instance of the TSP is pyramidally solvable is co-NP-complete. However, we do not know if a polynomial testing scheme exists in case the matrix also satisfies conditions (i)-(iv) of property GG4. We can replace condition (v) of the theorem by any one of the polynomially testable sufficiency conditions of Section 4.1.1 for pyramidal solvability of TSP.

A minor generalization of the non-negative case has been observed in [142]. The 2×2 matrix D in Example 1 above satisfies properties GG1, GG2 and GG4, but not the conditions in [142]. For other interesting sufficiency conditions the reader is referred to [51, 483].

4.3. Subclass of TSP polynomially solvable using GG scheme

We have seen that the GG scheme produces an optimal solution to a fairly large subclass of the TSP. However, implementation of the GG scheme is NP-hard even for the special case where C is a product matrix, (that is, there exist real numbers $\{a_i, b_i : i \in N\}$ such that $c_{ij} = a_i \times b_j$). In this case, if we assume that $a_1 \leq a_2 \leq \dots \leq a_n$ and permutation π is such that $b_{\pi(1)} \geq b_{\pi(2)} \geq \dots \geq b_{\pi(n)}$, then the density matrix of C^π is a non-negative matrix and hence, it satisfies properties GG1 and GG2. But it is shown in [742] (see also [361]) that this subclass of the TSP is

NP-hard, implying that, even for this special case, implementation of the GG scheme is an NP-hard problem. By imposing additional conditions on the matrix D and/or the starting permutation π , (or equivalently, the patching pseudograph G_p^π), we obtain special cases of the problem for which Step 3 of the GG scheme, and therefore the entire GG scheme, can be implemented in polynomial time. We now investigate such special cases.

4.3.1 Class of permutations π for which GG scheme starting with π can be polynomially implemented for arbitrary C^π .

The cases of this type reported in the literature are: (i) the case of pyramidal TSP (i.e. when $\pi = \xi$ (the identity permutation)), (ii) the case when G_p^π is a multi-path, (iii) G_p^π is a multi-tour, and (iv) G_p^π is a multi-tree.

A pseudograph is said to be a Multi-path, a multi-tour or a multi-tree if and only if the underlying simple graph, obtained by removing all the parallel edges and loops, is a path, a tour or a tree, respectively.

(i) The Case of Pyramidal TSP ($\pi = \xi$). Klyaus [506] (see also [361]) showed that for any cost matrix C , the minimum cost pyramidal tour can be obtained in $O(n^2)$ time by using a dynamic programming recursion. This implies that the pyramidally solvable classes of TSP identified in Section 4.1.1 are all polynomially solvable.

The following alternate dynamic programming recursion, based on Observation 17, is given in [479]. It is very similar to the one given by Park [660]. Let us define an acyclic, directed multigraph $GA = (N', E_u \cup E_l)$, where $N' = N - \{n\}$ and for all $1 \leq i < j < n$, we have 2 copies, e_{ij}^u and e_{ij}^l , of arc (i, j) with $e_{ij}^u \in E_u$, and $e_{ij}^l \in E_l$. We assign to e_{ij}^u a weight $w_{ij}^u = \sum_{x=i}^j \sum_{y=x}^j d_{xy}$ (the sum of all the entries of D in the triangle with vertices $((i, i), (i, j), (j, j))$); and to e_{ij}^l a weight $w_{ij}^l = \sum_{x=i}^j \sum_{y=i}^x d_{xy}$ (the sum of all the entries of D in the triangle with vertices $((i, i), (j, i), (j, j))$). We call a path with arcs alternately in E_u and E_l an *alternating path* (*a-path*). It follows from Observation 17 that our problem is equivalent to the problem of finding an optimal *a-path* in GA from node 1 to node $(n - 1)$. Let us call an *a-path* with an element of E_u as its last arc, an *upper a-path* (*ua-path*); and an *a-path* with an element of E_l as its last arc, a *lower a-path* (*la-path*). For all $i \in N'$, let us define s_i^u (s_i^l), the optimal weight of *ua-path* (*la-path*) in GA from

node 1 to node i , as follows. $s_1^u = s_1^l = d_{1,1}$. For $i \in \{2, 3, \dots, n - 1\}$,

$$s_i^l = \min\{s_j^u - d_{jj} + w_{ji}^l : 1 \leq j < i\} \text{ and}$$

$$s_i^u = \min\{s_j^l - d_{jj} + w_{ji}^u : 1 \leq j < i\}.$$

Then, the minimum of the costs of all pyramidal tours is $\min\{s_{n-1}^u, s_{n-1}^l\}$. Given all the arc weights w_{ij}^u, w_{ij}^l , an optimal pyramidal tour can thus be computed in $O(n^2)$ time using the above recursion.

We can perform the following pre-process step, which takes $O(n)$ time, such that after this pre-processing, any arc weight can be calculated in unit time. Let the numbers $\{a_i^u, b_i^u : i \in N\}$ and $\{a_i^l, b_i^l : i \in N\}$ be such that the matrices \hat{C} and \bar{C} , defined as

$$\hat{c}_{ij} = c_{ij} - a_i^u - b_j^u \text{ and } \bar{c}_{ij} = c_{ij} - a_i^l - b_j^l$$

have

$$\hat{c}_{ii} = \hat{c}_{i,i+1} = \hat{c}_{i+1,i+1} = 0 \text{ and } \bar{c}_{ii} = \bar{c}_{i+1,i} = \bar{c}_{i+1,i+1} = 0 \text{ for all } i \in N'.$$

Then, as seen before, matrices C, \hat{C} and \bar{C} have the same density matrix D and $w_{ij}^u = \bar{c}_{i,j+1}$ and $w_{ij}^l = \hat{c}_{j+1,i}$. Thus, an optimal pyramidal tour can be computed in $O(n^2)$ time.

Park [660] showed that if the density matrix of C is non-negative then, using the results of [2, 275], the minimum cost pyramidal tour can be computed in $O(n)$. We now present the relevant results in [2, 275] and we give a proof of the Park's result using the above dynamic programming recursion.

Theorem 48 [2] *Given integers m and n , with $m \geq n$, and an $m \times n$, integer matrix A with a non-negative density matrix, let $g(j) = \min\{a_{ij} : i = 1, \dots, m\}$ for all $j \in N$. Then values $\{g(j) : j \in N\}$ can be computed in $O(m)$ time.*

Theorem 49 [275] *Let W be an $n \times n$ matrix with density matrix D , such that $d_{ij} \geq 0$ for all $j > i + 1$. For any $j \in \{2, \dots, n\}$, let $g(j) = \min\{p(i) + w_{ij} : i < j\}$, where $p(1)$ is known and $p(i) = f(g(i))$ for all $i \in \{2, \dots, n\}$, for some given function $f(\cdot)$ whose value, for any real valued x , can be computed in unit time. Then $\{g(j) : j = 2, \dots, n\}$ can be computed in $O(n)$ time.*

The following generalization of Theorem 49 is proved in [345]. It will be useful in Section 6.

Theorem 50 [345] *Let A and B be matrices of sizes $n \times m$ and $m \times n$, respectively, each having a non-negative density matrix. For any $j \in \{2, \dots, n\}$, let $g(j) = \min\{p(i) + a_{ik} + b_{kj} : i < j, k \in \{1, \dots, m\}\}$, where $p(1)$ is known and for all $j \in \{2, \dots, n\}$, $p(j) = f(g(j))$ for some given function $f(\cdot)$ whose value, for any real value x , can be computed in unit time. Then $\{g(j) : j \in \{2, \dots, n\}\}$ can be computed in $O(n + m)$.*

The proofs of theorems 49 and 50 given in [275] and [345], respectively, are both algorithmic proofs. An interesting feature of both these algorithms is that for all $i \in \{2, \dots, n\}$, all the values $\{E(j) : j < i\}$ are computed before computing the value of $E(i)$. This property enables us to obtain the following generalization of Theorem 49.

Theorem 51 *For arbitrary, given integers m, k and $\{u_i, v_i : 1 \leq u_i < v_i \leq m, i \in \{1, \dots, k\}\}$, let $r_i = v_i - u_i + 1$ and $r = \sum_{i=1}^k r_i$. For $i \in \{1, \dots, k\}$, let W^i be an $r_i \times r_i$ integer matrix with rows/columns indexed by the set $\{u_i, u_i + 1, \dots, v_i\}$ and with density matrix D^i . For any $i \in \{1, \dots, k\}$ and $j \in \{u_i, \dots, v_i\}$, let*

$$g(i, j) = \begin{cases} \min\{p(i, a) + w_{aj}^i : u_i \leq a < j\} & \forall u_i < j \leq v_i \\ \infty & \text{otherwise} \end{cases}$$

where, $p(i, 1)$ is given and $p(i, j) = f_{ij}(g(1, j), g(2, j), \dots, g(k, j))$, for some functions $f_{ij}(x)$ such that for any $j, I \subseteq \{1, \dots, k\}$ and n -vector x , the values $\{f_{ij}(x) : i \in I\}$ can be computed in time linear in the number of finite entries of x , and $|I|$. Then $\{g(i, j) : i \in \{1, \dots, k\}, u_i \leq j \leq v_i\}$ can be computed in $O(mr)$. If, in addition, for each $i \in \{1, 2, \dots, k\}$, $d_{ab}^i \geq 0$ for all $u_i < a + 1 < b \leq v_i - 1$, then the complexity can be reduced to $O(m + r)$.

Proof. The $O(mr)$ complexity can be achieved by straightforward implementation of dynamic programming recursion. Now let us consider the case where for each $i \in \{1, \dots, k\}$, $d_{a,b}^i \geq 0$ for all $u_i < a + 1 < b \leq v_i - 1$. For each $1 \leq t \leq m$, let $I_t = \{i : u_i \leq t < v_i\}$. For each $i \in I_1$, apply the Eppstein-algorithm only partially just until we get the value of $g(i, 2)$. Let $g(i, 2) = \infty$ for all $i \notin I_1$. Compute the values $\{p(i, 2) : i \in I_2\}$. Now for each $i \in I_2$, continue with the Eppstein-algorithm just until we get the value $g(i, 3)$. Let $g(i, 3) = \infty$ for all $i \notin I_2$. Compute the values $\{p(i, 3) : i \in I_3\}$. Continue thus, alternately computing the values $\{g(i, 4) : i \in I_3\}, \{p(i, 4) : i \in I_4\}, \dots, \{g(i, m) : i \in I_{m-1}\}, \{p(i, m) : i \in I_m\}$. The total complexity of the algorithm is (the complexity of applying Eppstein-algorithm to each $i \in \{1, \dots, k\}\}) + (\text{the complexity of computing } \{p(i, j) : i \in I_j\} \text{ for all } j) = O(r) + O(\sum_{t=2}^m (|I_{t-1}| + |I_t|)) = O(m + r)$. This proves the theorem. ■

We shall now prove the Park’s result.

Corollary 52 [660] *If the cost matrix C has a non-negative density matrix, then a minimum cost pyramidal tour can be computed in $O(n)$, where n is the size of the matrix C .*

Proof. Consider the formulation of the problem of finding a minimum cost pyramidal tour as the problem of finding a minimum cost a -path in an acyclic digraph with two types of edges, as discussed above. This problem can be converted to the problem in the statement of Theorem 51, with $k = 2$, by defining W^1 and W^2 as two $(n - 1) \times (n - 1)$ matrices with

$$w_{ij}^1 = \begin{cases} \bar{c}_{i,j+1}, & \forall 1 \leq i < j \leq n - 1 \\ \infty, & \text{otherwise} \end{cases}$$

$$w_{ij}^2 = \begin{cases} \hat{c}_{j+1,i}^u, & \forall 1 \leq i < j \leq n - 1 \\ \infty, & \text{otherwise} \end{cases}$$

and $g(1, j) = s_j^u$, $g(2, j) = s_j^l$, $p(1, j) = s_j^l - d_{jj}$, $p(2, j) = s_j^u - d_{jj}$. Since the matrices C , \hat{C} and \bar{C} have the same density matrix, it follows that the matrices W^1 and W^2 satisfy the requirements of Theorem 51. The result thus follows from Theorem 51. ■

Case (ii) The patching pseudograph G_p^π is a multi-path or a multi-tour. It is reported in [142] that the case when $G_p^\pi = (N_p^\pi, E_p^\pi)$ is a multi-path was first considered in [740], where an $O(n^3)$ algorithm for the implementation of the GG scheme in this case is given; and the complexity was reduced to $O(mn)$ in [344], where m is the number of nodes in N_p^π . The complexity was further reduced to $O(n)$ in [141] for the case when the density matrix D of C^π is non-negative. We shall show that the problem can be reduced to a special case of Theorem 51. Though our reduction is conceptually straightforward, because of the different types of arcs involved the description given may at first seem not so straightforward.

Let $\{e_1, e_2, \dots, e_{n-1}\}$ be the edge set E_p^π as in Definition 10. Let us assume that the nodes in the set $N_p^\pi = \{1, \dots, m\}$ are numbered such that the edges of the underlying simple graph of G_p^π form the path $(1, \dots, m - 1, m)$. The patching pseudograph in Figure 11.1 is a multi-path. Here, if we renumber node 2 as 1, and node 1 as 2, then the underlying simple graph forms the path $(1, 2, 3, 4)$.

Let $\{E_1, E_2, \dots, E_r\}$ be the partition of the non-loop edges in E_p^π , such that for some $0 < a_1 \leq b_1 < a_2 \leq b_2 < \dots < a_r \leq b_r \leq n - 1$,

each $E_i = \{e_{a_i}, e_{a_i+1}, \dots, e_{b_i}\}$ is a maximal set of consecutive edges in E_p^π which form a path. For example, in Figure 11.1, $a_1 = b_1 = 1$, $a_2 = b_2 = 3$, $a_3 = 4$, $b_3 = 6$, $a_4 = b_4 = 7$; and $E_1 = \{e_1\}$, $E_2 = \{e_3\}$, $E_3 = \{e_4, e_5, e_6\}$, $E_4 = \{e_7\}$, and $r = 4$. The edge sets E_1, E_2, E_3 and E_4 form paths (1, 2), (1, 2), (1, 2, 3, 4) and (3, 4), respectively. For edge set $E_p^\pi[T]$ of a spanning tree of G_p^π , and any set $S = \{x, x+1, \dots, y\} \subseteq T$, $E_p^\pi[S] \subseteq E_i$ for some i . Let $R = \{1, \dots, r\}$.

Define an acyclic, directed multigraph $GA = (M, \cup_{i \in R}(E_i^{t,u} \cup E_i^{s,l} \cup E_i^{s,u} \cup E_i^{s,l}))$, where $M = \{0, 1, \dots, m-1\}$ and for each $i \in R$, $p \in \{s, t\}$, $q \in \{u, l\}$

$$E_i^{p,q} = \{e_{jk}^{i,p,q} = (x, y) : a_i \leq j \leq k \leq b_i, x < y, \text{ the edge set } \{e_j, \dots, e_k\} \text{ forms path } (x+1, \dots, y+1)\}.$$

Selection of an arc $e_{jk}^{i,s,u}$ ($e_{jk}^{i,s,l}$) implies that edges $\{e_j, \dots, e_k\}$ are chosen to be in the tree $E_p^\pi[T^*]$, the corresponding interchanges are performed in the order (e_k, \dots, e_j) ((e_j, \dots, e_k)), and if $e_j = (x+1, x+2)$ ($e_j = (y, y+1)$) then edge e_{j-1} (e_{k+1}) is not selected to be in $E_p^\pi[T^*]$.

Selection of an arc $e_{jk}^{i,t,u}$ implies that if $e_j = (x+1, x+2)$ ($e_j = (y, y+1)$) then edges $\{e_{j-1}, \dots, e_k\}$ ($\{e_j, \dots, e_{k+1}\}$) are chosen to be in the tree $E_p^\pi[T^*]$ and the corresponding interchanges are performed in the order (e_k, \dots, e_{j-1}) ((e_{k+1}, \dots, e_j)).

We similarly interpret selection of arcs of types $e_{jk}^{i,s,l}$ and $e_{jk}^{i,t,l}$.

Let us now consider paths in GA from node 0 to node m of the following type: for every $i \in R$, every element of $E_i^{t,u}$ ($E_i^{s,l}$) in the path is immediately preceded by an element of either $E_i^{s,u}$ or $E_i^{s,l}$ (either $E_i^{t,u}$ or $E_i^{s,u}$); and for any element of $(E_i^{s,u} \cup E_i^{s,l})$ in the path, the immediately preceding arc is not an element of $(E_i^{p,q}$ for any $p \in \{s, t\}$ and $q \in \{u, l\}$). Each such path corresponds to a tree in G_p^π with some ordering of the corresponding patchings, and vice versa. We are thus interested in a minimum weight path of this type in GA , where an arc $e_{jk}^{i,p,q}$ is assigned weight $w_{jk}^{i,p,q}$ as defined below:

$w_{jk}^{i,s,u} = \sum_{v=j}^k \sum_{z=v}^k d_{vz}$ (the sum of all the entries of D in the triangle with vertices $((j, j), (j, k), (k, k))$).

$w_{jk}^{i,s,l} = \sum_{v=j}^k \sum_{z=j}^v d_{vz}$ (the sum of all the entries of D in the triangle with vertices $((j, j), (k, j), (k, k))$).

For $q \in \{u, l\}$, if $e_{jk}^{i,t,q} = (x, y)$, then

$$w_{jk}^{i,t,q} = \begin{cases} w_{j-1,k}^{i,s,q} - d_{j-1,j-1} & \text{if } e_j = (x+1, x+2) \\ w_{j,k+1}^{i,s,q} - d_{k+1,k+1} & \text{otherwise} \end{cases}.$$

As in the case of the method for finding an optimal pyramidal tour, we can perform a pre-processing step, which takes $O(n)$ time, such that after the pre-processing, cost of any arc in GA can be calculated in unit time. The required path can now be obtained using a dynamic programming recursion that is a straightforward extension of that in the case of pyramidal TSP and the resultant problem can be reduced to a special case of Theorem 51. (We leave this as an exercise.) This gives us a complexity of $O(nm)$ in general and a complexity of $O(n)$ when the density matrix D of C^π is non-negative.

It is shown in [141] that when the patching pseudograph is a multi-tour and the density matrix D of C^π is a non-negative matrix, the GG scheme can be implemented in $O(mn)$. In this case, there exists an ordering of the nodes in N_p^π such that the optimal spanning tree of step 3 of the GG scheme is a path $(1, 2, \dots, m)$. By deleting all the edges in E_p^π of the form $(1, m)$, the problem reduces to one similar to the case of multi-path. The original problem can thus be solved in $O(m^2n)$ in general and in $O(mn)$ when D is non-negative by considering each of the m such possible orderings of the nodes in N_p^π .

Case (iii) The patching graph G_p^π is a multi-tree. This case was first considered in [344] where an $O(m^2+n)$ algorithm is given for the case of multi-star and $O(m^5n)$ algorithm is given for the case of general multi-tree when D is non-negative. Here, $m = |N_p^\pi|$. The complexity for the general multi-tree case with non-negative D was reduced in [143] to $O(nm + m^3)$.

As in the case of multi-path, let us denote by $\{E_1, \dots, E_r\}$ the partition of the non-loop edges in E_p^π , where each E_i is a maximal set of consecutive edges which form a path in G_p^π . Let $E_i = \{e_j : j \in \{a_i, \dots, b_i\}\}$ and let $R = \{1, \dots, r\}$. Then, for any spanning tree $E_p^\pi[T^*]$ of G_p^π , and any set $S = \{j, j + 1, \dots, k\} \subseteq T^*$, $\{e_v : v \in S\} \subseteq E_i$ for some $i \in R$. Let $G' = (N_p^\pi, E')$ be the underlying simple graph of G_p^π .

Let us first consider the case when G' is a star graph. Then $|E_i| \leq 2$ for all i . Gaikov [344] (see also [143]) showed that in this case, Step 3 of the GG scheme can be formulated as a weighted matching problem [266, 569] as follows. Let us number the nodes in N_p^π as $\{0, 1, \dots, m - 1\}$, such that $E' = \{(0, i) : i \in \{1, \dots, m - 1\}\}$. Let $H = (V, F)$ be a multigraph on node set $V = \{1, 2, \dots, m - 1, 1', 2', \dots, r'\}$ with the edge set F defined as follows: For every $e_k = (0, u) \in E_i$, $(u, i') \in F$ and let us assign it a weight $w_{u, i'} = d_{kk}$. Also, for every $i \in R$ such that $E_i = \{e_k = (0, u), e_{k+1} = (0, v)\}$, $u \neq v$, $e^i = (u, v) \in F$ and let us assign it a weight $w^i = D[\{k, k + 1\}]$. Then any matching in H in which all the

nodes $\{1, \dots, m - 1\}$ are matched corresponds to a spanning tree $E_p^\pi[T]$ in G_p^π and has a total cost of $D[T]$, and vice versa. Thus our problem is equivalent to finding a minimum cost matching in H with all the nodes $\{1, \dots, m - 1\}$ matched and this can be solved in $O(n^3)$ time [266]. Gaikov [344] (see also [143]) exploited the special structure of this matching problem when D is non-negative to reduce the complexity to $O(m^2 + n)$.

The following $O(nm + m^3)$ dynamic programming scheme for the case of general multi-tree with $D \geq 0$ is given in [143]. Let us root the multi-tree G_p^π at an arbitrary node $q \in N_p^\pi$. This defines a father-son relationship between the nodes in N_p^π . For any $i \in N_p^\pi$, let $G(i) = (N(i), E(i))$ denote the maximal sub-multi-tree of G_p^π rooted at i . For any $i \in R$, let $f(i)$ be node in N_p^π such that $G(f(i))$ contains all the edges in E_i and is the smallest such sub-multi-tree of G_p^π . Following [143], we call $f(i)$ the *anchor* of E_i . We need the following functions.

(a) For any $v \in N_p^\pi$, $k \in R$ and $\{e_i, e_j\} \subseteq E_k$ such that (i) $e_i = (v, u)$ for some son u of v and (ii) $e_j \in E(u) \cup \{e_i\}$, let $Y = \{i, \dots, j\}$ if $i \leq j$ and $Y = \{j, \dots, i\}$ otherwise. Then, $X^1(v, e_i, e_j) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } H' = (N(u) \cup \{v\}, E(u) \cup \{e_i\}), S \cap \{a_k, \dots, b_k\} = Y\}$.

(b) For any $v \in N_p^\pi$ and any $e_i \in E_p^\pi$ such that $e_i = (v, u)$ for some son u of v , $X^2(v, e_i) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } H' = (N(u) \cup \{v\}, E(u) \cup \{e_i\}); i \in S\}$.

(c) For any $v \in N_p^\pi$, $k \in R$ such that $f(k) = v$ and edges $e_i = (v, u_1)$ and $e_{i+1} = (v, u_2)$ in E_k , $X^3(v, e_i, e_{i+1}) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } G' = (N(u_1) \cup N(u_2) \cup \{v\}, E(u_1) \cup E(u_2) \cup \{e_i, e_{i+1}\}); \{i, i + 1\} \subseteq S\}$.

(d) For any $v \in N_p^\pi$ and any son u of v , $z^1(v, u) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } H' = G_p^\pi[N(v) - N(u)]\}$.

(e) For any $v \in N_p^\pi$ and any edge $e_i = (v, u)$ for some son u of v , $z^2(v, e_i) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } G(v); i \notin S\}$.

(f) For any $v \in N_p^\pi$, $Opt(v) = \min\{D[S] : \{e_x : x \in S\} \text{ forms a spanning tree of } G(v)\}$.

The optimal solution we seek is $Opt(q)$. The following dynamic programming recursion is given in [143].

For any leaf node v of G , $Opt(v) = 0$. Let us consider the case when v is a non-leaf node.

Let us consider any $k \in R$ and $\{e_i, e_j\} \subseteq E_k$ such that (i) $e_i = (v, u)$ for some son u of v and (ii) $e_j \in E(u) \cup \{e_i\}$. Let $Y = \{i, \dots, j\}$ if $i \leq j$ and $Y = \{j, \dots, i\}$ otherwise. If $E_k \cap E(u) = \emptyset$, then $i = j$ and $X^1(v, e_i, e_i) = w(e_i) + Opt(u)$. Suppose $e_i \neq e_b = (u, x) \in E_k$. If $i = j$

then $X^1(v, e_i, e_i) = w(e_i) + z^2(u, e_b)$; and if $i \neq j$ then, $X^1(v, e_i, e_j) = X^1[u, e_b, e_j] + z^1(u, x) + D[Y] - D[y - \{i\}]$.

For any $k \in R$ and $e_i \in E_k$ such that $e_i = (v, u)$ for some son u of v , $X^2(v, e_i) = \min\{X^1(v, e_i, e_j) : e_j \in E_k \cap (E(u) \cup \{e_i\})\}$.

For any $k \in R$ such that $f(k) = v$ and edges $e_i = (v, u_1)$ and $e_{i+1} = (v, u_2)$ in E_k , $X^3(v, e_i, e_{i+1}) = \min\{X^1(v, e_i, e_j) + X^1(v, e_{i+1}, e_\ell) + D[S_1 \cup S_2] - D[S_1] - D[S_2] : e_j \in E_k \cap (E(u_1) \cup \{e_i\}); e_\ell \in E_k \cap (E(u_2) \cup \{e_{i+1}\}); S_1 = \{i, j\} \cup \{x : e_x \text{ lies on the unique sub-path of } E_k \text{ between } e_i \text{ and } e_j\}; S_2 = \{i + 1, \ell\} \cup \{x : e_x \text{ lies on the unique sub-path of } E_k \text{ between } e_{i+1} \text{ and } e_\ell\}\}$.

Computation of $opt(v)$ can be reduced to the problem on a multi-star as follows. Let $\{u_1, \dots, u_d\}$ be the set of sons of v in G . Let $H' = (V, F')$ be a multi-star with node set $V = \{v, u_1, \dots, u_d\}$ and edge set H' defined as follows: for any $e_x = (v, u_i) \in E_k$, edge $e_{v, u_i}^k \in F'$ and we assign a cost $X^2(v, e_x)$ to this edge. For any pair $\{e_x = (v, u_i), e_{x+1} = (v, u_j)\} \subseteq E_k$, we assign a cost $X^3(v, e_x, e_{x+1})$ to the pair $\{e_{v, u_i}^k, e_{v, u_j}^k\}$.

For any son u of v , $z^1(v, u)$ can be computed analogous to $opt(v)$ after deleting $G(u)$ from G .

For any edge $e_i = (v, u)$ for some son u of v , $z^2(v, e_i)$ can be computed similar to $opt(v)$.

We leave it as an exercise to verify that the complexity of this scheme is $O(nm + m^3)$.

4.3.2 Conditions on D under which GG scheme can be polynomially implemented for arbitrary C and π such that D is the density matrix of C^π .

For a given permutation π on N and a pair (A, B) of disjoint subsets of $N' = \{1, 2, \dots, n - 1\}$, let $F(\pi, A, B) = \{X : A \subseteq X \subseteq N' - B; E_p^\pi[X]$ is the edge set of a spanning tree in $G_p^\pi\}$. For a given matrix D , we denote by $\mathcal{P}(D, \pi, A, B)$ the problem of finding $T^* \in F(\pi, A, B)$ such that $D[T^*] = \min\{D[T] : T \in F(\pi, A, B)\}$. We denote the problem $\mathcal{P}(D, \pi, \emptyset, \emptyset)$ by $\mathcal{P}(D, \pi)$.

Thus, for arbitrary cost matrix C and permutation π , the implementation of the GG scheme with π as the starting permutation is polynomially equivalent to the problem $\mathcal{P}(D, \pi)$, where D is the density matrix of C^π .

For a given matrix D and disjoint sets A, B in $N' = \{1, 2, \dots, n - 1\}$, we say that the triplet (D, A, B) is of the *type GG* if and only if there exists a constant λ such that for all $A \subseteq S \subseteq N' - B$, $D[S] = (\sum_{i \in S} d_{ii}) + \lambda$. If $(D, \emptyset, \emptyset)$ is of the type GG then we say that *the matrix D is of the type GG*.

If a triplet (D, A, B) is of the type GG, then for any permutation π on N , the problem $\mathcal{P}(D, \pi, A, B)$ reduces to (i) the problem of finding a

minimum cost spanning tree $E_p^\pi[T^*]$ such that $A \subseteq T^* \subseteq N' - B$, (which can be solved in the $O(n \log n)$), followed by (ii) the problem of finding an optimal ordering of the edges in $E_p^\pi[T^*]$, (which can be solved in $O(n^2)$ using the dynamic programming recursion for finding an optimal pyramidal tour). This gives us an overall complexity of $O(n^2)$. If, in addition, the matrix D is non-negative, then the overall complexity can be reduced to $O(n \log n)$, using the result of Park [660].

As has been shown in Section 3.3, in the case of the GG-TSP there exists a permutation π such that the density matrix D of C^π is non-negative and is of the type GG. In fact, it follows easily from Observation 17 that a non-negative, matrix D is of the type GG if and only if there exists a pyramidal tour on ψ such that $D(\psi) = \sum_{i=1}^{n-1} d_{ii}$ and in this case, the corresponding value of λ is 0. Since, by theorems 44 and 45, the GG scheme with a starting permutation π is valid for $TSP(C)$ if the density matrix of C^π is non-negative [139], this gives us a polynomially solvable class of the TSP that is a generalization of the GG-TSP. Given a cost matrix C , the following scheme can be used to recognize in $O(n^2)$ if there exists a permutation π such that the density matrix of C^π is of this type. Find a permutation π , if one exists, such that the density matrix D of C^π is non-negative. An $O(n^2)$ algorithm for this has been given by Chandrasekaran [176] and also by Deineko and Filonenko [242]. Now find an optimal pyramidal tour ψ^* on cost matrix $C' = C^\pi$ using the $O(n)$ algorithm of Park [660].

We now present classes of matrices D introduced in [51, 483] for which there exists an integer k , which is either constant or polynomial in n , such that for any π , the problem $\mathcal{P}(D, \pi)$ can be decomposed into k subproblems $\mathcal{P}(D, \pi, A, B)$, with (D, A, B) of the type GG.

A non-negative matrix D is of the *type 2-GG* if and only if there exists $1 \leq u \leq n - 2$ such that the triplet $(D, \{u, u + 1\}, \emptyset)$ is of the type GG.

If $(D, \{u, u + 1\}, \emptyset)$ is of the type GG, then the corresponding value of λ is $\min\{d_{u,u+1}, d_{u+1,u}\}$ and the principal submatrices of D on row/columns $\{1, 2, \dots, u\}$ and $\{u + 1, \dots, n - 1\}$ are each of the type GG.

Theorem 53 [51, 483] *Let a cost matrix C and a permutation π be such that the density matrix D of C^π is of the type 2-GG. Then the GG scheme with π as the starting permutation produces an optimal solution to $TSP(C)$. Furthermore, in this case the GG scheme can be implemented in $O(n \log n)$.*

We leave the proof of this theorem as an exercise.

The matrices of the type 2-GG are further generalized in [51, 483] as follows.

A non-negative matrix D is of the type k -GG for some $k \geq 2$ if and only if there exist $1 < u_1 < \dots < u_k = n - 1$, and v_1, \dots, v_{k-1} such that for all $1 \leq i \leq k - 1$, $u_i < u_i + v_i \leq u_{i+1}$, and for any $A_i \subseteq S_i = \{u_i, u_i + 1, \dots, u_i + v_i\}$ for all $1 \leq i \leq k - 1$, $A = \cup_{i=1}^{(k-1)} A_i$, and $B = \cup_{i=1}^{(k-1)} (S_i - A_i)$, the triplet (D, A, B) is of the type GG. Obviously, the corresponding value of λ is $(D[A] - \sum_{i \in A} d_{ii})$ and if we define $u_0 = 1$ and $v_0 = 0$, then for each $0 \leq i \leq k - 1$, the principal submatrix of D on rows/columns $\{u_i + v_i, u_i + v_i + 1, \dots, u_{i+1}\}$ is of the type GG. It should be noted that a matrix of the type 2-GG is also of the type k -GG with $k = 2$ and $v_1 = 1$. The following result is proved in [51, 483]

Theorem 54 [51, 483] *Let a cost matrix C and a permutation π be such that the density matrix D of C^π is of the type k -GG for some $k \geq 2$, $1 < u_1 < \dots < u_k = n$ and v_1, \dots, v_{k-1} such that for all $1 \leq i \leq k - 1$, $u_i < u_i + v_i \leq u_{i+1}$. Then the GG scheme with π as the starting permutation produces an optimal solution to $TSP(C)$. Furthermore, in this case the GG scheme can be implemented in $O(2^{\sum(v_i+1)} n \log n)$ time. In particular, when $\sum_{i=1}^{(k-1)} v_i$ is bounded by a constant we get a strongly polynomial algorithm.*

When $v_i = 2 \forall i$, the complexity can be improved to $O((5^k)n \log n)$ [51, 483].

We now give some interesting special cases of theorems 53 and 54.

A non-negative matrix D is a *NE-staircase matrix* if and only if there exist $0 = u_0 \leq u_1 < u_2 < \dots < u_k = n$ and $n \geq v_1 > v_2 > \dots > v_{k-1} \geq 1$ such that the non-diagonal entries of D which have non-zero values form a subset of $S = \{(u_1, n - 1), \dots, (u_1, v_1 - 1), (u_1 + 1, v_1 - 1), \dots, (u_2, v_1 - 1), (u_2, v_1 - 2), \dots, (u_2, v_2 - 1), \dots\}$.

It should be noted that the elements of the set S form a staircase which starts at the right most column and/or the top most row and ends in the left most column and/or the bottom most row and therefore at the most one diagonal coefficient of D belongs to S . It follows from the characterization of Gilmore-Gomory cost matrices in [176] that a matrix C is a Gilmore-Gomory matrix if and only if there exists a permutation π such that density matrix of $-C^\pi$ is a NE-staircase matrix. Such matrices were first considered in [245] where it is shown that the GG scheme produces an optimal solution the corresponding TSP.

Theorem 55 [51, 245, 483] *A NE-staircase matrix is of the type GG or 2-GG and for any cost matrix C and permutation π such that the density matrix D of C^π is a NE-staircase matrix, an optimal solution to the $TSP(C)$ can be obtained in $O(n)$ using the GG scheme with π as the starting permutation.*

A matrix C is called a *small matrix* [340] if and only if there exist $\{a_i, b_i : i \in N\}$ such that $c_{ij} = \min(b_i, a_j)$. The $TSP(C)$, where C is a small matrix, was first studied and solved by Gabovich [340] in $O(n)$ time (see also [361]). The following result is given in [799].

Theorem 56 [799] *Let C be a small matrix with $c_{ij} = \min(b_i, a_j)$ for some $\{a_i, b_i : 1 \leq i \leq n\}$. Without loss of generality, let us assume that $b_1 \leq b_2 \leq \dots \leq b_n$. Let π be a permutation on N such that $a_{\pi(1)} \geq a_{\pi(2)} \geq \dots \geq a_{\pi(n)}$. Then the density matrix D of C^π is a NE-staircase matrix with at the most one non-zero diagonal coefficient and an optimal solution to $TSP(C)$ can be obtained in $O(n)$ using the GG scheme with π as the starting permutation.*

Given vectors a and b , obtaining the required node ordering and the permutation π needs $O(n \log n)$ time. However, the complexity can be reduced using the following observation. Suppose the staircase of the density matrix D of C^π contains the diagonal element d_{ii} . For any re-ordering of the elements of the row sets $\{1, \dots, i-2\}$ and $\{i+2, \dots, n\}$ and column sets $\{1, \dots, i-2\}$ and $\{i+2, \dots, n\}$ of C^π , the new matrix, which can be represented as \tilde{C}^ψ , where \tilde{C} is the cost matrix corresponding to the new ordering of the nodes in N , can be shown to satisfy property GG1. Hence, by Theorem 44, for cost matrix \tilde{C} , the GG scheme with ψ as the starting permutation produces an optimal solution. Such a matrix can be obtained in $O(n)$ time and for it, the GG scheme with ψ as the starting permutation can be implemented in $O(n)$.

The following interesting special case of Theorem 54 arises in a scheduling problem without preemption in a two machine flexible robotics cell that repetitively produces parts of different types with no buffer storage between the machines [23]. Here, it is required to produce repetitively a given set of n products. Each product has to be processed first on the machine M1 and then on the machine M2. Neither the robot nor any of the machines can be in possession of more than one product at any time. The processing times of each of the n products on machines M1 and M2 are given. The objective is to minimize the average steady-state cycle time required for the repetitive production of the products. It is assumed that the robot arm takes a constant time to load a new item on machine M1, move from machine M1 to machine M2, and unload an item from machine M2. In [23], (see also Chapter 1) it is shown that the problem can be formulated as TSP with the cost matrix C defined as $c_{ij} = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$ for some suitable positive quantities μ , and $\{a_i, b_i : 1 \leq i \leq n\}$; and an interesting $O(n \log n)$ algorithm is developed for it. We now show that for a suitable choice of permuta-

tion π , the density matrix D of C^π is of the type k -GG and therefore by Theorem 54, the problem $TSP(C)$ can be solved in $O(n \log n)$ time using the GG scheme. Let us assume without loss of generality that the products are ordered such that $b_1 \leq b_2 \leq \dots \leq b_n$. Let π be a permutation on N such that $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$. Let $0 \leq x, y \leq n$ be such that $a_{\pi(i)} \leq \mu$ for all $1 \leq i \leq x$; $a_{\pi(i)} > \mu$ for all $x < i \leq n$; $b_j \leq \mu$ for all $1 \leq j \leq y$; $b_j > \mu$ for all $y < j \leq n$. Let $r = \min\{x, y\}$. Then the principal submatrix of D on rows/columns $\{1, 2, \dots, r-1\}$ is a NE-staircase matrix, which is of the type GG or of the type 2-GG with $u_1 = t$ for some $1 \leq t \leq r-2$, and $v_1 = 1$; the principal submatrix of D on rows/columns $\{r, r+1, \dots, n-1\}$ is of the type GG and all the other entries of D have a value 0, except possibly the elements of the set $\{(i, x) : 1 \leq i < r\} \cup \{(y, j) : 1 \leq j < r\}$. Thus, the entire matrix D is of the type 2-GG with $u_1 = r$, and $v_1 = 1$ or $k = 3, u_1 = t, u_2 = r$, and $v_1 = v_2 = 1$. Hence, by Theorem 54, the corresponding problem $TSP(C)$ can be solved in $O(n \log n)$ time using the GG scheme with π as the starting permutation.

Examples of polynomially solvable classes of non-permuted-distribution matrices. The sufficiency conditions of theorems 44 and 45 do not require the matrix D to be non-negative. Several examples of such matrices D , which are not non-negative and for which the GG scheme can be implemented in polynomial time for any C and π such that D is the density matrix of C^π , are given in [51] and [483]. We give below one of them.

Consider the following class of matrices $D : d_{ii} = 2$ for all i ; $d_{i,i+1} = -1$ for all $i \in \{1, \dots, n-2\}$, i odd; $d_{ij} = 0$ for all other i, j . This class of matrices satisfy property GG4. We show now that in this case, the problem $\mathcal{P}(D, \pi)$ can be solved in polynomial time for any π . For any set $E_p^\pi[T]$ of edge set of a spanning tree in G_p^π , $(\sum_{i \in T} d_{ii}) = 2(|T|) =$ a constant. Therefore, an optimal tree is one for which the entries of D contributing to $D[T]$ include the maximum number of -1 's. This is equivalent to finding a tree in G_p^π with a maximum number of pairs of edges $\{\{e_1, e_2\}, \{e_3, e_4\}, \dots\}$. This is a special case of the graph 2-parity problem and can be solved in strongly polynomial time [568].

4.4. Other subclasses of TSP solvable using GG scheme

In [339, 775] the GG scheme is shown to work for the following subclass of TSP, which is termed *the wallpaper cutting problem* in [348, 361]. Here, we are digraph $H = (V, F)$ on k nodes which is a tour $(1, 2, \dots, k, 1)$,

(that is, $F = \{(1, 2), (2, 3), \dots, (k - 1, k), (k, 1)\}$). Each node i in N is assigned an ordered pair (a_i, b_i) of nodes in V . Each arc e in F is assigned a length $\ell(e)$ such that total length of all the arcs in H is non-negative. The cost matrix C of TSP on N is defined as c_{ij} = the length of the unique path from node b_i to node a_j in H . It is shown in [339] (see also [775]) that there exists an optimal solution π to the assignment problem on C such that the GG scheme starting with π produces an optimal solution to the TSP. Furthermore, an $O(k \log k)$ scheme is given for finding such a permutation π and it is shown that the GG scheme with π as the starting permutation can be implemented in $O(n + k \log k)$. Alternate schemes for the problem are given in [361, 482] and a greedy scheme for a special case is given in [348]. We shall revisit this problem in Section 5.3.3.

Another interesting subclass of TSP for which an optimal solution can be obtained in $O(n \log n)$ using the GG scheme is given in [413]. This arises as a special case of the problem of DNA sequencing by hybridization. Here, an instance $TSP(G, C)$ is considered, where $G = (N, E)$ is an incomplete digraph which is the line digraph of another incomplete digraph H , (that is, nodes in N are in 1-1 correspondence with the arcs in H and for distinct nodes i, j in N , $(i, j) \in E$ if and only if the arcs of H corresponding to i and j are, respectively, of the form $e_i = (u, v)$ and $e_j = (v, x)$ for some distinct nodes u, v, x of H). It is assumed that each node in H has indegree=outdegree=2. This implies that each node in G has indegree=outdegree=2, $|N|$ is even and that there exists a renumbering of the nodes in N and a permutation π on N such that

$$(i) \quad E = \cup_{i=1}^{n/2} \{(2i-1, \pi(2i-1)), (2i-1, \pi(2i)), (2i, \pi(2i-1)), (2i, \pi(2i))\},$$

and

$$(ii) \quad c_{2i-1, \pi(2i-1)} + c_{2i, \pi(2i)} \leq c_{2i-1, \pi(2i)} + c_{2i, \pi(2i-1)}.$$

The permutation π is therefore an optimal solution to the assignment problem on C and the only permutations on N that have finite total cost with respect to the cost matrix C are permutations of the form $\pi \circ \beta_{i_1} \circ \dots \circ \beta_{i_k}$ for some $\{i_1, \dots, i_k\} \subseteq \{1, 3, \dots, n-1\}$. Let $E_p^\pi[T^*]$ be the edge set of a minimum cost spanning tree of G_p^π , where for each $i \in \{1, 3, \dots, n-1\}$, cost of edge e_i is $(c_{2i-1, \pi(2i)} + c_{2i, \pi(2i-1)} - c_{2i-1, \pi(2i-1)} + c_{2i, \pi(2i)}) \geq 0$; and the cost of every other arc in G_p^π is ∞ . Then it follows from Lemma 17 that $\pi \circ \beta_{i_1} \circ \dots \circ \beta_{i_k}$ is an optimal tour for any ordering (i_1, \dots, i_k) of elements of T^* .

If the digraph H is such that every node in H has indegree=outdegree ≥ 4 , then the problem is NP-hard [413], while the complexity of the case when every node in H has indegree=outdegree=3 is open.

4.5. Other cases solvable using a patching-type scheme

In the GG scheme, we start with a suitable permutation π and optimally patch the subtours of π using only adjacent patchings. Conceptually, we could obtain a larger class of solvable cases, if we allowed the use of non-adjacent patchings. Only a couple of solvable cases are known where an optimal solution is obtained this way [546, 801].

Lawler [546] considers the case where the density matrix D of C is an upper triangular matrix, (that is, $d_{ij} = 0$ for all $i > j$) and for this case, the following scheme is given in [546] for transforming any permutation π such that $\pi(n) = 1$ into a tour with the same cost.

- Step 1:** If π is a tour, then output $\pi^* = \pi$ and stop. Else, π contains r subtours, $\mathfrak{C}_1, \dots, \mathfrak{C}_r$ on node sets N_1, \dots, N_r , respectively, for some $r > 1$. Without loss of generality, let us assume that $\{1, n\} \subseteq N_1$. Let $i = 0$, $N_1^i = N_1$, $u^i = n$, $\pi^0 = \pi$ and $\mathfrak{C}_1^i = \mathfrak{C}_1$.
- Step 2:** Let $u^{i+1} = \max\{j : j \in N - N_1^i\}$. Suppose $u^{i+1} \in N_v$. Let $\pi^{i+1} = \pi^i \circ \alpha_{u^{i+1}, u^i}$. In π^{i+1} , the subtours \mathfrak{C}_1^i and \mathfrak{C}_v are combined into a subtour \mathfrak{C}_1^{i+1} on the node set $N_1^{i+1} = N_1^i \cup N_v$. All the other subtours of π^{i+1} and precisely the subtours of π^i .
- Step 3:** If $N_1^{i+1} = N$, then output $\pi^* = \pi^{i+1}$ and stop. Else, let $i = i+1$ and go back to step 2.

In each iteration i , $\pi^i(u^i) = 1$ and $\pi^i(u^{i+1}) = a^i \leq u^{i+1}$. Hence, $c(\pi^{i+1}) = c(\pi^i) + M_{u^{i+1}, u^i-1, 1, a^i-1}^D = c(\pi^i)$; and therefore, $c(\pi^*) = c(\pi)$.

Now, consider any tour γ on N such that $\gamma(n) = a > 1$. Since γ is a tour, there exists $a \leq u^1 < n$ such that $\gamma(u^1) = b < a$. Let $\gamma^1 = \gamma \circ \alpha_{u^1, n}$. Then $c(\gamma^1) = c(\gamma) + M_{u^1, n-1, b, a-1}^D = c(\gamma)$ and $\gamma^1(n) = b$. Again, since γ is a tour, there exists $b \leq u^2 < n$ such that $\gamma^1(u^2) = \gamma(u^2) = c < b$. Let $\gamma^2 = \gamma^1 \circ \alpha_{u^2, n}$. Then $c(\gamma^2) = c(\gamma^1) = c(\gamma)$. By repeating this process, we get a permutation γ^* such that $c(\gamma^*) = c(\gamma)$ and $\gamma^*(n) = 1$. It therefore follows that if we choose π as a minimum cost permutation with $\pi(n) = 1$ and perform the Lawler's algorithm to find a tour with the same cost as π , then this resultant tour will be an optimal tour.

The complexity of $TSP(C)$, when there exists a permutation π on N such that the density matrix D of C^π is an upper triangular matrix, is open. In [801], the following $O(n^3)$ scheme is presented for the case where C is a non-negative, symmetric matrix and C^π is upper triangular, where $\pi(i) = n - i + 1$ for all $i \in N$.

- Step 0:** Let $m = \lfloor 1/2(n - 1) \rfloor$. Let A be an $m \times m$ matrix defined as :
 $a_{i,1} = \min\{c_{ik} : k \in \{i + 1, \dots, n - m\}\}$ and $a_{ij} = c_{i,j+\lfloor (1/2)n \rfloor}$
for all $i = 1, \dots, m$ and $j = 2, \dots, m$.
- Step 1:** Compute an optimal permutation (solution to assignment problem) ψ on $\{1, \dots, m\}$ with A as the cost matrix. Let $P = \emptyset$.
- Step 2:** For $i = 1, \dots, m$, if $i + \psi(i) \leq m + 1$, then add i to set P . If $\psi(i) = 1$, then let $\gamma(i) = k(i)$, else, let $\gamma(i) = \psi(i) + \lfloor (1/2)n \rfloor$. Here, $k(i)$ is such that $a_{i,1} = c_{i,k(i)}$. Let $i = 1$ and $V = \{1\}$.
- Step 3:** If $i \in P$, then let $V = V \cup \{\gamma(i)\}$ and $i = \gamma(i)$. Else, find the smallest integer j such that $j \notin V$ and $i + j \geq n + 1$. Let $\gamma(i) = j$, $V = V \cup \{j\}$ and $i = j$.
- Step 4:** If $V = N$, then stop. Else, go to step 3.

It is easy to see that $c(\gamma) = \sum_{i=1}^m a_{i,\psi(i)}$. It is shown in [801] that for any tour ρ on N , $c(\rho) \geq \sum_{i=1}^m a_{i,\psi(i)}$. From this, validity of the algorithm follows.

5. Minimum cost connected directed pseudograph problem with node deficiency requirements (MCNDP)

We now present an alternative formulation of GG-TSP. Ball et al [83] arrive at this new formulation when dealing with a special case of the GG-TSP that arises in the context of the problem of optimal insertion of chips on a PC board and they give an algorithm for it that is less efficient but more intuitive than the Gilmore-Gomory algorithm. The results of [83] are extended in [482] to obtain polynomial schemes for other non-trivial classes of TSP.

We need the following definitions and basic results. For any directed pseudograph $G = (V, E)$ and any node i in V , we define the *deficiency of node i* as $df(i) = d^+(i) - d^-(i)$. We say that G is *degree balanced* if and only if for each i in V , $df(i) = 0$.

Theorem 57 [84] *A directed pseudograph G is degree balanced if and only if it can be decomposed into arc-disjoint subtours. A connected, directed pseudograph is degree balanced if and only if it has an Eulerian tour.*

For any directed pseudograph $G = (V, E)$ and any vector $x \in \mathbb{Z}_+^E$, G^x is a directed pseudograph (V', E') where E' contains $x(e)$ copies of arc e for all e in E , and

$$V' = \{i : i \in V; \text{ at least one arc in } E' \text{ is incident with } i\}.$$

Thus, G^x is obtained from G by deleting some arcs in E and making multiple copies of some of the other arcs in E as indicated by vector x and then deleting the isolated nodes.

5.1. An alternative formulation of the GG-TSP

Let $X = \cup_{i=1}^n \{a_i, b_i\} = \{u_1, u_2, \dots, u_k\}$, where $u_1 \leq u_2 \leq \dots \leq u_k$. For each job i , let $a_i = u_{s_i}$ and $b_i = u_{f_i}$. Consider the directed pseudograph $G = (V, E \cup F)$, and the digraph $G_B = (V, E)$, where $V = \{1, 2, \dots, k\}$, $E = \cup_{i=1}^{k-1} \{e_i^f = (i, i + 1), e_i^b = (i + 1, i)\}$, and $F = \{g_i = (s_i, f_i) : i \in N\}$. For any job i , processing of job i corresponds to traversing the arc g_i in F . Let us assign to each arc $g_i \in F$ a cost $w(g_i) = 0$ and for any $i \in \{1, \dots, k - 1\}$, let us assign to the arcs e_i^f and e_i^b in E costs $w(e_i^f) = \int_{u_i}^{u_{i+1}} f(x)dx$, and $w(e_i^b) = \int_{u_i}^{u_{i+1}} g(x)dx$. For any $1 \leq i, j \leq k$, let $[i, j]$ be the unique path from node i to node j in G_B . Then the cost of the path $[i, j]$ is $w([i, j]) = \sum_{e \in [i, j]} w(e)$ which is the cost of changing the temperature of the furnace from u_i to u_j . Hence, for any jobs i, j in N , the cost of processing job j immediately after job i is $c_{ij} = w([f_i, s_j])$. The condition that $f(x) + g(x) \geq 0$ for all x , implies that $w(e_i^f) + w(e_i^b) \geq 0$ for all i . Thus, every subtour in G_B has a non-negative cost.

We associate with any cyclic permutation γ of the jobs, a vector $x_\gamma \in \mathbb{Z}_+^{E \cup F}$ defined as follows: Let $\gamma = (1, i_2, i_3, \dots, i_n, 1)$. Starting from the node s_1 traverse the arcs of G and return back to the node s_1 as follows: $g_1 - [f_1, s_{i_2}] - g_{i_2} - \dots - g_{i_n} - [f_{i_n}, s_1]$. For each $e \in E \cup F$, $x_\gamma(e)$ is the number of times arc e is traversed. Then (i) $x_\gamma(g_i) = 1$ for all $i \in N$ and (ii) the directed pseudograph $G^{x_\gamma} = (V, E' \cup F)$ is degree balanced and connected. If for each $e \in E \cup F$ we assign a cost $w(e)$ to each of the copies of the arc e in $E' \cup F$, then the total cost $w(G^{x_\gamma})$ of all the arcs of G^{x_γ} is $c(\gamma)$, the total change-over cost under γ . Conversely, suppose $x \in \mathbb{Z}_+^{E \cup F}$ is such that $x(g_i) = 1$ for all i in N and $G^x = (V, \tilde{E} \cup F)$ is connected and degree balanced. Then by Theorem 57, G^x has an Eulerian tour. We can represent such a tour as $g_1 - \wp_{f_1, s_{i_2}} - g_{i_2} - \wp_{f_{i_2}, s_{i_3}} - \dots - g_{i_n} - \wp_{f_{i_n}, s_1}$ where, for any i, j , $\wp_{i,j}$ is a trail from node i to node j in G_B . Let $\gamma = (1, i_2, i_3, \dots, i_n, 1)$. If we assign a cost of $w(e)$ to each of the copies of the arc e in $\tilde{E} \cup F$, then $w(G^x) = w(\wp_{f_1, s_{i_2}}) + w(\wp_{f_{i_2}, s_{i_3}}) + \dots + w(\wp_{f_{i_n}, s_1}) \geq w([f_1, s_{i_2}]) + \dots + w([f_{i_n}, s_1]) = c(\gamma)$. Thus, the problem GG-TSP is equivalent to the problem of finding $x^* \in \mathbb{Z}_+^{E \cup F}$ with $x^*(g_i) = 1$ for all $i \in N$ such that $G^{x^*} = (V, E^* \cup F)$ is connected and degree balanced and has a minimum total arc cost.

We now present a generalization of this problem which we call the Minimum Cost Steiner directed pseudograph Problem with Node-Deficiency

Requirements (MCNDP) [482]. We then present an intuitive algorithm for a special case of the MCNDP, which we call the G-G problem and which includes the above formulation of the GG-TSP as a special case. This algorithm is a minor modification of the algorithm presented in [83] for the GG-TSP. Finally, we discuss the results in [482] which use the insight gained from the algorithm for the G-G problem to develop polynomial schemes for interesting special cases of MCNDP and, therefore, of the TSP.

5.2. Statement of the problem MCNDP

We are given a directed pseudograph, $G = (V, E \cup F)$, where $V = \{1, 2, \dots, k\}$ is the node set and E and F are disjoint families of arcs on V . We call the set $S = \{i : i \in V; \text{no arc in } F \text{ is incident with } i\}$ as the set of Steiner nodes. We are also given an arc cost function, $w : E \cup F \rightarrow \mathbb{R}$, with $w(e) = 0$ for all $e \in F$; and a node deficiency function $d : V \rightarrow \mathbb{Z}$, such that (i) $\sum_{i \in V} d_i = 0$ and (ii) $d_i = 0$ for all $i \in S$. The problem is to find $x^* \in \mathbb{Z}_+^{E \cup F}$ with $x^*(e) = 1$ for all $e \in F$ such that (i) $G^{x^*} = (V', E^* \cup F)$ is connected; (ii) for each node $i \in V'$, the deficiency of node i in G^{x^*} is d_i ; and (iii) if for each $e \in E \cup F$, we assign a cost $w(e)$ to each of the copies of the arc e in $E^* \cup F$, then the cost $w(G^{x^*})$ is minimum.

We assume throughout that every subtour in $G_B = (V, E)$ has a non-negative cost. (Else, the problem, if feasible, is obviously unbounded.) We allow for the possibility that some arcs in F are loops, (that is, of the form (i, i)).

The special case of the MCNDP problem in which $d_i = 0$ for all $i \in V$ is known as the Rural Postman Problem (RPP) [83, 120]). It is easy to show that the RPP is polynomially equivalent to the TSP and therefore RPP is NP-hard [556]. We call the special case of the MCNDP with $E = \cup_{i=1}^{k-1} \{e_i^f = (i, i + 1), e_i^b = (i + 1, i)\}$, the G-G problem. This includes our formulation of the GG-TSP. In this case, our assumption above implies that $w(e_i^f) + w(e_i^b) \geq 0$ for all i . Also, here we can assume without loss of generality that $S = \emptyset$. (Since every node has less than 3 neighbors in G_B , we can eliminate a Steiner node without changing the structure of G_B .) Let $F = \{g_i : i \in N\}$. The following algorithm for the G-G problem is a minor modification of the algorithm in [83].

5.2.1 An algorithm for G-G Problem. For each $e \in E$, let $\mathcal{E}(e)$ be the largest integer such that for any $x \in \mathbb{Z}_+^{E \cup F}$ with $x(g_i) = 1$ for all $i \in N$ and for which G^x satisfies the node deficiency requirement, $x(e) \geq \mathcal{E}(e)$. We call $\mathcal{E}(e)$ the essential number of the arc e . In steps 0, 1

and 2 we find a vector $x^0 \in \mathbb{Z}_+^{E \cup F}$ such that $x^0(g_i) = 1$ for all i , $x^0(e) = \mathcal{E}(e)$ for all $e \in E$ and the directed pseudograph $G^{x^0} = (V', E^0 \cup F)$ satisfies the node-deficiency requirement. In Step 3, we add a minimum cost set of extra edges to G^{x^0} to make it connected while continuing to satisfy the node-deficiency requirement.

Step 0: Let $x^0(e) = 0$ for all $e \in E$ and $x^0(g_i) = 1$ for all $i \in N$. Let $i = 1$.

Step 1: Calculate $df(i)$, the deficiency of the node i in G^{x^0} . If $df(i) > d_i$ then let $e = e_i^b$; else, let $e = e_i^f$. Increase $x^0(e)$ by $|df(i) - d_i| = \mathcal{E}(e)$.

Step 2: Increase the value of i by 1. If $i < k$, then go to Step 1.

Step 3: Suppose G^{x^0} contains r connected components G^1, G^2, \dots, G^r . If $r = 1$, then output $x^* = x^0$ and stop. Else, construct a pseudograph $G_P = (N_P, E_P)$, which we shall also call a patching pseudograph, with node set $N_P = \{1, 2, \dots, r\}$ and edge set $E_P = \{e_i = (u, v) : i \in \{1, \dots, k-1\}, i \in G^u, i+1 \in G^v\}$. Let the cost of the edge e_i be $c(e_i) = w(e_i^f) + w(e_i^b)$. Find a minimum cost spanning tree \mathfrak{S}^* in G_P . Increase $x^0(e_i^f)$ and $x^0(e_i^b)$ by 1 for each $e_i \in \mathfrak{S}^*$ to get a new vector x^* . Output G^{x^*} and stop.

The following explanation will convince the reader of validity of the algorithm. It is not difficult to verify that G^{x^0} , the input to step 3 of the algorithm, contains precisely $\mathcal{E}(e)$ copies of each arc e and it satisfies the node-deficiency requirement. If G^{x^0} is connected then obviously x^0 is an optimal solution to the problem. Else, for each $e \in E$ we have to increase $x^0(e)$ by some number $y(e)$ so that the total cost of additional copies of the arcs is minimum and the resultant directed pseudograph is connected and still satisfies the node-deficiency requirements. It is obvious that $y(e_i^f) = y(e_i^b) \in \{0, 1\}$. Recursively, we can see that $y(e_i^f) = y(e_i^b) \in \{0, 1\}$ for all i . Thus, our problem reduces to finding $Y \subseteq \{1, 2, \dots, k-1\}$ such that if we increase $x^0(e_i^f)$ and $x^0(e_i^b)$ by 1 for each $i \in Y$, the resultant directed pseudograph is connected and the cost of the additional arcs is minimum. This is precisely the minimum cost spanning tree problem.

It is not difficult to verify that the overall complexity of the algorithm is $O(n + k \log k)$. To solve the GG-TSP using this scheme, we need the following additional operations. (i) Construct the directed pseudograph G , which can be done in $O(n + k \log k)$ and (ii) construct the desired cyclic ordering of the jobs from the output directed pseudograph G^{x^*} ,

which can be done in $O(n^2)$ [627]. (Note that $x^*(e) \leq n$ for all $e \in E$.) Though the current scheme is not as efficient as the Gilmore-Gomory scheme, it is more intuitive and the insight it provides helps in developing polynomial schemes for other interesting special cases of MCNDP.

5.3. Other solvable cases of MCNDP

We now take a closer look at the two main steps of the above algorithm for the G-G problem and the properties of the digraph G_B and the arc set F in the G-G problem which make the algorithm work:

5.3.1 Main step (I): Finding a minimum cost directed pseudograph satisfying node-deficiency requirements.

For any digraph $G_B = (V, E)$, any set $F = \{g_i : i \in N\}$ of additional arcs on V and any vector $\{d_i : i \in V\}$ of node-deficiency requirements, let $\mathcal{F} = \{x : x \in \mathbb{Z}_+^{E \cup F}, x(g_i) = 1 \text{ for all } i \in N, \text{ the directed pseudograph } G^x \text{ satisfies the node-deficiency requirements}\}$. In the case of the G-G problem, the digraph G_B is such that for any choice of the set F and the vector d , the corresponding set \mathcal{F} has a least element x^0 , (that is, there exists $x^0 \in \mathcal{F}$ such that for any $x \in \mathcal{F}, x \geq x^0$). G^{x^0} is thus the minimum cost directed pseudograph satisfying the node-deficiency requirements. The set \mathcal{F} can be expressed as $\mathcal{F} = \{x : Ax = b; x \geq 0, \text{ integer}\}$, where A is the node-arc incidence matrix of G_B [747] and for any node $i \in V, b_i = d_i - (\text{number of arcs of } F \text{ incident out of } i) + (\text{number of arcs of } F \text{ incident into } i)$. Let $X_b = \{x : Ax = b, x \geq 0\}$. It follows from total unimodularity of the matrix A [747] that (i) the set \mathcal{F} has a least element if and only if the set X_b has a least element and the least element of the two sets is the same; (ii) the least element, if it exists, is an extreme point of X_b and can be obtained by solving the linear program $\min \{e^t x : Ax = b, x \geq 0\}$. This linear program can be solved in $O(k|E| \log k)$ using network flow techniques [7, 612]. In [482] the following characterization is given of digraphs G_B for which for any choice of the arc set F and the vector d , the corresponding set X_b , if it is non-empty, has a least element:

We call a connected digraph G_B a *DS-digraph* if and only if (i) every simple closed chain in G_B is a subtour and (ii) the subtours of G_B are pairwise arc-disjoint.

Theorem 58 [482] *Let $G_B = (V, E)$ be a digraph on node set $V = \{1, 2, \dots, k\}$. Let A be the node-arc incidence matrix of G_B . Then the set $X_b = \{x : Ax = b; x \geq 0\}$ has a least element for each integral k -vector b for which it is non-empty if and only if each connected component of G_B is a DS-digraph.*

In the case of the G-G problem, the only simple closed chains of the digraph G_B are $\{(i, i + 1, i) : i \in \{1, \dots, k - 1\}\}$. Thus, G_B is a DS-digraph. It is shown in [482] that given any digraph G_B we can check in $O(k)$ time if it is a DS-graph; and for a DS-digraph, the least element x^0 of \mathcal{F} can be obtained in $O(k|E| + n)$, where $k = |V|$.

5.3.2 Main step (II): Optimally augmenting G^{x^0} to make it connected while satisfying node-deficiency requirements.

In this step, we find a minimum cost set of inessential arcs $\{y(e) : e \in E\}$ which when added to G^{x^0} , result in a directed pseudograph which is connected and still satisfies the node-deficiency requirements. The directed multigraph $G_B^y = (V', E^y)$, having as arcs only the set of *inessential arcs*, is obviously degree balanced and hence by Theorem 57, the set of inessential arcs is a collection of arc-disjoint subtours. Since every subtour has a non-negative cost, there exists a minimum cost set of inessential arcs that contains at the most one copy of each subtour. Thus the problem of finding the minimum cost set of inessential arcs is the one of finding a set of subtours in G_B with minimum total arc cost which when added to G^{x^0} , containing only the essential arcs, results in a connected directed pseudograph. In the case of the G-G problem the digraph G_B satisfies the G-G property, defined below, and which makes this step easy.

A digraph G_B satisfies *G-G property* if and only if (i) all the subtours in G_B are of size 2 and (ii) in any instance of the MCNDP problem on G_B , we can assume that the set S of Steiner nodes is empty.

Suppose the digraph G_B in an instance of the MCNDP satisfies condition (i) of G-G property. Suppose the directed pseudograph $G^{x^0} = (V', E' \cup F)$, containing only the set of *essential arcs*, has r connected components G^1, G^2, \dots, G^r . Let $Y = V - V'$. Construct the patching pseudograph $G_P = (N_P, E_P)$, where $N_P = \{1, 2, \dots, r\} \cup Y$ and $E_P = \{e_{ij} = (u, v) : (i, j, i) \text{ is a subtour in } G_B \text{ and } i \in G^u \text{ or } i = u \in Y; j \in G^v \text{ or } j = v \in Y\}$. Let the cost of the edge $e_{i,j}$ be $c(e_{i,j}) = w((i, j)) + w((j, i))$. Then, the problem of augmenting G^{x^0} to make it connected while still satisfying the node-deficiency requirements is equivalent to the problem of finding a minimum cost Steiner tree in G_P with Y as the set of Steiner nodes. This is in general NP-hard [347]. If in addition $S = \emptyset$, then $Y = \emptyset$; and the problem reduces to the minimum cost spanning tree problem.

5.3.3 Other polynomially solvable cases of MCNDP.

We shall now use the insight gained above to develop polynomial schemes for other special classes of the MCNDP. The following lemma will be useful.

Lemma 59 [482] *Consider an arbitrary instance of MCNDP problem. Let $\Delta = 1/2 \sum_{i \in V} |d_i|$. If the problem has an optimal solution, then there exists an optimal solution G^{x^*} such that*

(i) *for any arc $e \in E$, $x^*(e) \leq \Delta + n$;*

(ii) *for any pair of arcs in E of the type $\{(i, j), (j, i)\}$,*

$$(x^*((i, j)), x^*((j, i))) \in \{(1, 1)\} \cup \{(u, 0), (0, u) : 0 \leq u \leq \Delta + n\}.$$

We leave the proof of Lemma 59 as an exercise.

Class (i) The Tree-TSP and its extension[83, 482]. Here, the digraph $G_B = (V, E)$ is such that its underlying undirected, simple graph is a tree T on the node set V . Our assumption of non-existence of a negative cost subtour in G_B implies that for all $\{(i, j), (j, i)\} \subseteq E$, $w((i, j)) + w((j, i)) \geq 0$. In this case, G_B is a DS-digraph and satisfies condition (i) of the G-G property. We can also assume, without loss of generality, that every node $i \in V$ such that in the tree T , $\deg(i) \leq 2$, is a non-Steiner node. However, some nodes with degree more than 2 in the tree T may be Steiner nodes. If there are no Steiner nodes then, from the arguments given above, it follows that the corresponding MCNDP problem can be solved in $O(n + k \log k)$ using a minor modification of the scheme for the G-G problem. If there are Steiner nodes, then the main step (ii) of the algorithm is a special case of the minimum cost Steiner tree problem and it is shown in [480] that this special case of the MCNDP is NP-hard even when $d_i = 0$ for all i . However, if the number of Steiner nodes is bounded by a fixed constant r , then the problem can be obviously solved in $O(2^r k \log k + n)$, by solving 2^r spanning tree problems, one for each choice of the subset of the Steiner nodes to be included in the tree. The corresponding undirected case is discussed in [321]. In [485, 487], an undirected version of the tree TSP is considered and it is shown that a modification of the GG scheme solves the problem.

Class (ii) An extension of the wallpaper cutting problem[482]. Here, the digraph G_B is a tour $(1, 2, \dots, k, 1)$. Our assumption that every subtour in G_B has a non-negative cost implies that $\sum_{e \in E} w(e) \geq 0$. Since every node has at most two neighbors, we can assume that $S = \emptyset$. In this case, G_B is a DS-digraph and for arbitrary d_i 's, the set of *essential arcs* can be found in $O(n + k \log k)$ [482]. Since G_B has only one subtour passing through all the arcs, the main step (II) is trivial - if G^{x^0} is connected then it is obviously an optimal solution; else, add to x^0 one copy of each arc in E . The complexity of this algorithm is $O(n + k \log k)$. When $d_i = 0$ for all i , this is equivalent to the *Wallpaper cutting problem* [339, 348, 361, 775] discussed in Section 4.4. However, obtaining an optimal tour for the *Wallpaper cutting problem* using this

approach takes $O(nk)$ time. Though this approach is computationally not as efficient as the ones in [339, 361, 775], it is more intuitive.

Class (iii) Let $E = \{(1, 2), \dots, (k - 1, k), (k, 1)\} \cup \{(1, k), (k, k - 1), \dots, (2, 1)\}$. Again, since every node here has at the most two neighbors, we can assume that $S = \emptyset$. By Lemma 59, there exists an optimal solution, G^{x^*} such that $(x^*((1, k)), x^*((k, 1))) \in \{(1, 1)\} \cup \{(u, 0), (0, u) : u \in 0, 1, \dots, \Delta + n\}$. If we fix the values of $x^*((1, k))$ and $x^*((k, 1))$, the problem reduces to the G-G problem which can be solved in polynomial time as shown above. This gives us a scheme of $O((\Delta + n)(n + k \log k))$. In [482] properties of the scheme for the G-G problem are exploited to develop an algorithm for this case with complexity $O(k(n + k \log k))$.

Class (iv) Warehouse Order-picking problem and its extensions [699, 482]. In this case, the digraph $G_B = (V, E)$ represents a rectangular warehouse and is defined as follows: $V = \{1, 2, \dots, k\}$. For some positive integers m and v_1, v_2, \dots, v_m , such that $0 = v_0 < (v_1 - 1) < (v_2 - 1) < (v_3 - 1) < \dots < (v_m - 1) = (k - 1)$,

$$E_1 = \cup_{j=1}^m \{\cup\{(i, i + 1), (i + 1, i)\} : v_{j-1} < i < v_j\}$$

$$E_2 = \cup_{j=1}^{m-1} \{(v_{j-1} + 1, v_j + 1), (v_j + 1, v_{j-1} + 1), (v_j, v_{j+1}), (v_{j+1}, v_j)\}$$

and $E = E_1 \cup E_2$ (see figure 11.6).

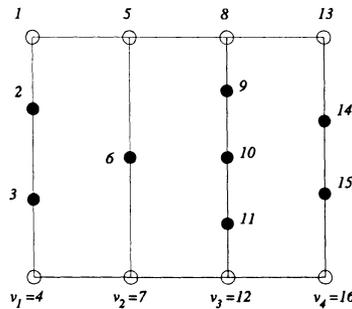


Figure 11.6. Digraph G_B corresponding to warehouse order-picking problem

Ratliff and Rosenthal [699] consider an undirected version of the problem, (that is, having $w((i, j)) = w((j, i)) \geq 0$ for all $(i, j) \in E$) with all the arcs in F as loops, $S = \{1, v_1, v_1 + 1, v_2, v_2 + 1, v_{m-1}, v_{m-1} + 1, v_m\}$ as the set of Steiner nodes, and $d_i = 0$ for all i . In this case, the *essential number* of each arc in E is 0 and the problem reduces to that of finding a vector $y \in \mathbb{Z}_+^E$, such that $G^y = (V', E^y)$ is degree balanced, all the non-Steiner nodes belong to the same connected component of G^y , and $\sum_{e \in E} w(e)y(e)$ is minimum. Ratliff and Rosenthal [699] give an $O(k)$

dynamic programming scheme for the problem. Our line of reasoning above points to an alternate dynamic programming scheme as follows.

As pointed out above, the problem of finding the optimal set of *inessential arcs* is equivalent to finding the optimal set of subtours in G_B such that in the resultant directed pseudograph all the non-Steiner nodes belong to the same connected component. The digraph G_B has two types of subtours : (i) two-arc subtours and (ii) subtours of the form

$$(v_i + 1, v_i + 2, \dots, v_{i+1}, v_{i+2}, \dots, v_j, v_j - 1, \dots, v_{j-1} + 1, \\ (v_{j-2} + 1), \dots, v_{i+1} + 1, v_i + 1)$$

and its reverse.

Lemma 60 *There exists an optimal set of subtours in which the subtours of size more than 2 are pairwise arc-disjoint.*

Lemma 60 leads to an obvious dynamic programming scheme with a complexity of $O(k)$. We leave the details as an exercise.

The following generalizations of the Ratliff-Rosenthal result are given in [482].

Subclass (iv(a)): *m is a fixed constant; the set F is arbitrary; and the node deficiency function $\{d_i : i \in k\}$ is arbitrary:*

In this case, if we delete from G_B , $(m - 1)$ pairs of arcs

$$\{(1, v_1 + 1), (v_1 + 1, 1)\}, \{(v_2, v_3), (v_3, v_2)\}, \{(v_2 + 1, v_3 + 1), (v_3 + 1, \\ v_2 + 1)\}, \{(v_4, v_5), (v_5, v_4)\}, \{(v_4 + 1, v_5 + 1), (v_5 + 1, v_4 + 1)\}, \dots\},$$

we get a digraph of the type in the G-G problem. Using this fact and extending the ideas of case (iii) above, an algorithm for this case has been developed in [482] which has a complexity $O(k^{m-1}(n + k \log k))$.

Subclass (iv(b)): *For a constant, ℓ , there exist $0 = i_0 < i_1 < \dots < i_r = v_m = k$, such that (i) for all $1 \leq j < r$, $i_j = v_p$ for some $p \in \{1, 2, \dots, m - 1\}$; (ii) for all $0 \leq j < r$, let $i_j = v_p$ and $i_{j+1} = v_q$. Then $q - p \leq \ell$; and (iii) for every arc $(u, v) \in F$, $\{u, v\} \subseteq \{i_j + 1, \dots, i_{j+1}\}$ for some $0 \leq j < r$: Again, by using ideas similar to the ones used in cases (iii) and (iva), an algorithm of complexity $O((k^{\ell+1})(k \log k + n))$ has been developed for this case in [482].*

6. Solvable cases of geometric TSP

An instance of symmetric TSP is called *geometric TSP* if every element of the set N is represented by a point in \mathbb{R}^d for some integer $d > 1$ and for any $\{i, j\} \subseteq N$, c_{ij} is the distance between the points

corresponding to i and j , computed according to some geometric norm. The norms usually considered are L^p -norm for some positive integer p , where for any $x, y \in \mathbb{R}^d$, $\text{dist}(x, y) = (\sum_{i=1}^d |x_i - y_i|^p)^{1/p}$ and in case of L^∞ -norm, $\text{dist}(x, y) = \max\{|x_i - y_i| : i \in \{1, \dots, d\}\}$. Another class of norms considered in the literature is the class of polyhedral norms. In this case, we are given a centrally symmetric polyhedron $P \subseteq \mathbb{R}^d$ having the origin as its center; and for any $x, y \in \mathbb{R}^d$, $\text{dist}(x, y)$ is the minimum value u such that if we center the polyhedron P at x by translation and scale it by u , the point y is on the boundary of the resultant polyhedron. The subclass of geometric TSP, in which $d = 2$ and the norm used is the L^2 -norm, is called *Euclidean TSP*.

It is shown in [456] that the geometric TSP is NP-hard for any fixed dimension d and any L^p or polyhedral norm. However, specially in the case of Euclidean TSP, the nice, geometric representation of the problem has led to identification of some interesting polynomially solvable cases. It should be noted that if $TSP(C)$ is an instance of Euclidean TSP and C' is a symmetric matrix having the same density matrix as C , then $TSP(C')$ is not necessarily Euclidean. As we shall see below, most of the polynomial solution schemes that have been developed for special cases of Euclidean TSP actually require for their validity and polynomial complexity not the Euclideanness but certain properties of the density matrix of C .

The first and perhaps the only known non-trivial property of Euclidean TSP was pointed out by Flood [314] who observed that in this case an optimal tour defines a simple (i.e. non-self-intersecting) polygon in the plane. Another interesting conjecture made in [760] was that every optimal tour is a subgraph of the Delaunay triangulation (see [673] for the definition). However, a counter-example to this conjecture was given in [493] and another simple counter-example is given in [264].

Throughout this section, we denote by S the set of n points in \mathbb{R}^d corresponding to the elements of the set N . Let $CH(S)$ be the convex hull of S and let $S_H \subseteq S$ be the subset of points that lie on the boundary of $CH(S)$. An instance of Euclidean TSP is said to be a *convex-hull TSP* if and only if $S_H = S$.

Lemma 61 (*Folklore*) *In the case of Euclidean TSP, the nodes corresponding to the point set S_H appear in every optimal tour in the same order in which these points lie on the boundary of $CH(S)$. In particular, in the case of convex-hull TSP if the elements of N are arranged in the cyclic order in which the corresponding points S lie along the boundary of the convex polygon, then $(1, 2, \dots, n, 1)$ is an optimal tour.*

Lemma 61 follows from Flood’s observation and the fact that any other tour that does not satisfy the conditions of the lemma defines a self-intersecting polygon in the plane. Quintas and Supnick [690] proved that these results hold for the points defined on a 2-sphere, when the point set does not contain any pair of antipodal points and all edges are minor geodesic arcs. We shall now discuss the generalization of these results to general, symmetric TSP given in [59] and [491]. Throughout the rest of this section, we denote by G a complete undirected graph on node set N .

For a symmetric cost matrix C , two edges (u, v) and (x, y) in G are said to *intersect* [59] if and only if they are nonadjacent (i.e. $\{u, v\} \cap \{x, y\} = \emptyset$) and

$$(c_{uv} + c_{xy}) \geq \max\{(c_{uy} + c_{vx}), (c_{ux} + c_{vy})\}.$$

They are said to *strictly intersect* if and only if the inequality is strict. *Two paths in G are said to (strictly) intersect* if and only if an edge in one path (strictly) intersects an edge in the other path. A path in G is *(strictly) self-intersecting* if and only if a pair of edges in the path (strictly) intersect.

Obviously, every optimal solution to a symmetric TSP is non-strictly-self-intersecting. The following stronger result is proved in [59].

Lemma 62 [59] *Let N_p, N_a , and N_b be disjoint subsets of N and let φ be a path in G such that the node set of φ is N_p . Let C be a symmetric cost matrix such that for any $u \in N_a$, $v \in N_b$ and any path φ' between u and v such that the node set of φ' is a subset of $N - N_p$, φ' strictly intersects φ . Then φ cannot be a subpath of an optimal solution to $TSP(C)$.*

Proof. If a tour in G contains φ as a subpath, then it also contains a subpath φ' on a subset of $N - N_p$ connecting some node u in N_a to some node v in N_b . Such a tour is therefore strictly self-intersecting and hence it cannot be an optimal tour. ■

Kalmanson [491] abstracted a property of the cost matrix of a convex-hull TSP which ensures optimality of the tour $(1, 2, \dots, n, 1)$. A symmetric cost matrix C is called a *Kalmanson matrix* [491] if and only if for all $1 \leq u < v < w < x \leq n$, the edges (u, w) and (v, x) intersect. If the intersection is strict in each case, then we call C a *non-degenerate Kalmanson matrix* [59]. If C is a (non-degenerate) Kalmanson matrix then we shall call $TSP(C)$ a *(non-degenerate) Kalmanson TSP*.

It should be noted that the Kalmanson conditions are defined on the density matrix of the cost matrix C and thus the class of Kalmanson TSP includes non-Euclidean instances of TSP.

Let C be a symmetric cost matrix such that for some $k < n$, its principal submatrix C_P on rows (and columns) $1, 2, \dots, k$ is a Kalmanson matrix. The node set $Y = \{k + 1, k + 2, \dots, n\}$ is interior to C_P [59] if and only if for all $1 \leq u < v < w < x \leq k$, any two node disjoint paths (u, \dots, w) and (v, \dots, x) in G , such that at least one of these two paths contains a node in Y , strictly intersect.

Theorem 63 [59] *Let C be a cost matrix and for some integer $k < n$, let C_P be the principal submatrix of C on rows/columns $1, 2, \dots, k$. If C_P is a Kalmanson matrix and the node set $\{k + 1, k + 2, \dots, n\}$ is interior to C_P then there exists an optimal tour to $TSP(C)$ in which the nodes $1, 2, \dots, k$ appear either in ascending order or in descending order. If, in addition, C_P is a non-degenerate Kalmanson matrix then the nodes $1, 2, \dots, k$ appear in every optimal tour to $TSP(C)$ either in ascending order or in descending order.*

Proof. Let us first consider the case when C_P is a non-degenerate Kalmanson matrix. Let γ be a tour in G in which the nodes $1, 2, \dots, k$ appear in neither ascending nor descending order. Then for some $1 < u + 1 < w < k$, γ has a subpath φ of the type (u, \dots, w) which does not contain any node in $N_a = \{u + 1, u + 2, \dots, w - 1\}$ and $N_b = \{w + 1, w + 2, \dots, k, 1, 2, \dots, u - 1\}$. Let N_p be the node set of φ . The sets N_p, N_a and N_b and the path φ satisfy the condition of Lemma 62 and therefore, γ cannot be an optimal solution to $TSP(C)$. If C_P is a Kalmanson matrix which is not non-degenerate, then we can perturb C to a symmetric matrix C' such that the corresponding principal submatrix C'_P of C' on rows/columns $1, 2, \dots, k$ is a non-degenerate Kalmanson matrix, the node set $\{k + 1, k + 2, \dots, n\}$ is interior to C'_P and every optimal solution to $TSP(C')$ is an optimal solution to $TSP(C)$. The result now follows from the non-degenerate case. This proves the theorem. ■

Since, by definition, an empty set is interior to any Kalmanson matrix we have the following corollary.

Corollary 64 [491] *If C is a Kalmanson matrix then the tour $\gamma = (1, 2, \dots, n, 1)$ is an optimal solution to $TSP(C)$. If C is a non-degenerate Kalmanson matrix then γ is the unique optimal solutions to $TSP(C)$.*

A symmetric matrix C is a *non-adjacent Kalmanson matrix (NK-matrix)* [55] if and only if for every $1 \leq u < v < w < x \leq n$, such that u, v, w and x are not consecutive nodes (in the modulo sense) and edges (u, w) and (v, x) intersect.

Theorem 65 [55, 59] *If C is a NK-matrix, then an optimal solution to $TSP(C)$ can be obtained in $O(n)$.*

One of the most general polynomially solvable cases of Euclidean TSP known so far is the *Convex-hull-and- k -line TSP*, where k is a fixed positive integer [247]. In this case, for some $m < n$, the points corresponding to some $n - m$ elements of N lie on the boundary of a convex polygon, while the points corresponding to the remaining m elements of N lie in the interior of the convex polygon on a set of k line segments which are quasi-parallel, a term introduced in [732] which generalizes the notion of parallelism. It is assumed in [247] that the carrying lines of the k segments intersect the boundary of the convex polygon in the same two edges and that every one of the $n - m$ points on the boundary of the convex polygon either lies above everyone of the k carrying lines of the k segments or lies below everyone of the k lines (see Figure 11.7).

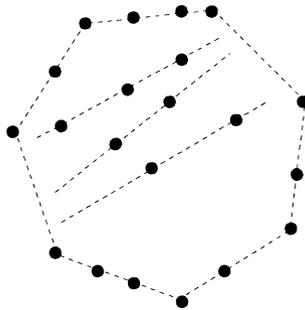


Figure 11.7. The convex-hull-and- k -line TSP

Deineko and Woeginger [247] produced an $O(n^{k+2})$ algorithm for this general case, extending the results on the 3-(parallel) line TSP in [235] and the k -(quasi-parallel) line TSP, for a fixed k , in [732]. A special case of this problem with $k = 1$ (called convex-hull-and-line TSP) is considered in [246] and their $O(n^2)$ algorithm for it is improved in [345] to an $O(n)$ scheme. As shown in [59], the validity and computational efficiency of the algorithms in [246] and [345] can be proved under some Kalmanson-like conditions on the density matrix of the cost matrix C and hence, these algorithms work for a larger subclass of TSP that includes some non-Euclidean cases. A similar generalization can be obtained for the convex-hull-and- k -line TSP.

6.1. The convex-hull-and- k -line TSP

For any integer k and numbers $0 = n_0 < n_1 < \dots < n_k = n$, let $N^i = \{n_{i-1} + 1, \dots, n_i\}$ for $i = 1, 2, \dots, k$. We call (N^1, N^2, \dots, N^k)

an ordered partition of N . An edge (a, b) in G is called a *long chord* with respect to an ordered partition (N^1, \dots, N^k) of N if and only if a and b both belong to same set N^u and $|a - b| > 1$. A collection $\{(a_i, b_i) : i \in \{1, \dots, f\}\}$ of edges in G is called a *fence* with respect to this ordered partition if and only if for each $i \in \{1, \dots, f\}$,

- (i) a_i and b_i belong to two different sets N_u and N_v and
- (ii) there exists some $x \in \cup_{j=1}^f \{a_j, b_j\}$ such that either $n_{u-1} < a_i < x \leq n_u$ or $n_{v-1} < b_i < x \leq n_v$.

A collection of edges in G is said to be *long-chord-and-fence-free (LCF-free)* with respect to an ordered partition if and only if it does not contain any long chord or fence as a subset.

The main results of [247] are that (i) for any partition of the node set N into k linearly ordered subsets, a minimum cost tour which is LCF-free with respect to this partition can be computed in $O(n^k)$; and (ii) in the case of the convex-hull-and- k -line TSP there exists a natural partition of the node set into $(k+2)$ linearly ordered sets such that there exists an optimal tour which is LCF-free with respect to this partition. We shall now elaborate on these below.

Let (N^1, \dots, N^k) be an ordered partition of N . Let a vector $l \in \mathbb{Z}_+^k$ be such that $0 \leq l_i \leq n_i - n_{i-1}$ for all i and let $N^l = \cup_{i=1}^k \{n_{i-1} + 1, \dots, n_{i-1} + l_i\}$. Let $M = \{(a_1, b_1), \dots, (a_r, b_r)\}$ be a non-empty partial matching of elements of $\{i : l_i \geq 1\}$. Then we denote by $R(M, l)$ the collection of sets of node-disjoint paths $R = \{R_1, \dots, R_r\}$ on N^l such that

- (i) every node in N^l belongs to some path R_i ;
- (ii) for $i \in \{1, \dots, r\}$, path R_i connects node $n_{a_i-1} + l_{a_i}$ to node $n_{b_i-1} + l_{b_i}$; and
- (iii) the set of all the edges in all the paths in R is LCF-free with respect to the given ordered partition.

Lemma 66 [247] *Let (N^1, \dots, N^k) be a given ordered partition of N . Every LCF-free tour on N contains either (i) an edge (n_i, n_j) or (ii) an edge $(n_i - 1, n_i)$ for some $i \in \{1, 2, \dots, k\}$ such that $n_i - n_{i-1} \geq 2$. For any $l \in \mathbb{Z}_+^k$ such that $0 \leq l_i \leq n_i - n_{i-1}$ for all i and any set $M = \{(a_1, b_1), \dots, (a_r, b_r)\}$ of non-empty partial matching of elements of $\{i : l_i \geq 1\}$, any set R in $R(M, l)$, as defined above, contains either (i) an edge $(n_{i-1} + l_i, n_{j-1} + l_j)$, where l_i and l_j have value at least 1, or (ii) an edge $(n_{i-1} + l_i - 1, n_{i-1} + l_i)$ for some $i \in \cup_{q=1}^r \{a_q, b_q\}$ such that $l_i \geq 2$.*

By using Lemma 66 and applying straightforward dynamic programming recursion, we get the following lemma.

Lemma 67 [247] *Let (N^1, \dots, N^k) be a given ordered partition of N . For any symmetric cost matrix C , vector $l \in Z_+^k$ such that $0 \leq l_i \leq n_i - n_{i-1}$ for all i , and partial matching $M = \{(a_1, b_1), \dots, (a_r, b_r)\}$ of elements of $\{i : l_i \geq 1\}$, the element of $R(M, l)$ with minimum total edge cost can be computed in $O(n^k)$. Hence, minimum cost LCF-free tour can be computed in $O(n^k)$.*

Lemma 68 below establishes that in the case of the convex-hull-and- k -line TSP there exists an optimal tour that is LCF-free.

A set Y of line segments in the Euclidean plane is said to be *quasi-parallel* [247, 732] if and only if we can assign an orientation to each of the line segments such that (i) for every pair y_1, y_2 of the line segments, the intersection of the carrying lines of y_1 and y_2 lies either to the left of both y_1 and y_2 on their respective carrying lines or to the right of both y_1 and y_2 on their respective carrying lines, where the terms left and right are as per the orientations of y_1 and y_2 ; and (ii) for any $Y' \subseteq Y$ with $|Y'| \geq 3$, there exist two segments y_1, y_2 in Y' such that out of the four closed regions into which the carrying lines of y_1, y_2 divide the plane, all the segments of Y' lie in the same region. We shall assume in what follows that such an orientation of the line segments in Y is known.

If a set Y of line segments is quasi-parallel, then any subset of Y is also quasi-parallel.

Lemma 68 [247] *Let $TSP(C)$ be an instance of Euclidean TSP on node set N for which for some set $Y = \{y_1, \dots, y_p\}$ of quasi-parallel line segments and some $0 = n_0 < n_1 < \dots < n_p = n$, the points in the plane corresponding to each node set $N^i (= \{n_{i-1} + 1, \dots, n_i\})$ lie on the oriented line segment y_i such that for $n_{i-1} + 1 \leq u < v \leq n_i$, the point u is to the left of the point v (where the term left is as per the orientation of y_i). Then for any fence F (with respect to the ordered partition (N^1, \dots, N^k) of N), two of the edges in F intersect.*

Theorem 69 [247] *Consider an instance of Convex-hull-and- k -line TSP. Let us number the nodes in N corresponding to the points on the convex hull above (below) the carrying lines of the quasi-parallel line segments as $\{1, \dots, n_1\}$ ($\{n_{k+1} + 1, \dots, n\}$) in the order in which they occur when we traverse the convex hull starting from its point of intersection with the carrying line of any segment to the left of the segment. Also let us number the nodes in N corresponding to the points on the line segment y_i as $\{n_i + 1, \dots, n_{i+1}\}$ such that for $n_i + 1 \leq u < v \leq n_{i+1}$, the point*

corresponding to node u is to the left of the point corresponding to the node v (where the term left is as per the orientation of y_i). Then every optimal tour is LCF-free with respect to this ordered partition and hence an optimal tour can be obtained in $O(n^{k+2})$.

Proof. Obviously, an optimal tour cannot contain a long chord. If possible, let γ be an optimal tour that contains a fence with respect to the ordered partition. Let F be a minimal fence in γ . Then F contains at the most two points corresponding to each line segment. If F does not contain any node in $\{1, \dots, n_1\} \cup \{n_{k+1} + 1, \dots, n\}$ then by Lemma 68, two of the edges in F intersect and we have a contradiction to the optimality of γ . Else, if F contains 2 nodes in $\{1, \dots, n_1\}$ ($\{n_{k+1} + 1, \dots, n\}$) then let y_0 (y_{k+1}) be the line segment joining the points corresponding to these nodes and let Y' be the set obtained by adding these additional segments to Y . Then Y' is quasi-parallel and again, by Lemma 68, we have contradiction to optimality of γ . This proves the theorem. ■

6.2. A generalization of the convex-hull-and-line TSP

We shall now discuss the generalization of the convex-hull-and-line TSP considered in [59]. This generalization is arrived at by looking at the convex-hull-and-line TSP as follows: Let N_3 be the set of m points on the line segment in the interior of the convex polygon. The carrying line of this line segment divides the boundary of the convex polygon into two regions : upper and lower. Let N_1 be the set of points on the upper boundary of the convex polygon let N_2 be the set of points on the lower boundary. Each of the three subsets of points $N_1 \cup N_2$, $N_1 \cup N_3$, and $N_2 \cup N_3$ lie on the boundary of some convex polygon. Thus the Convex-hull-and-line TSP is a specific composition of three convex-hull TSP's. In [59], this is generalized to a similar composition of three Kalmanson matrices as follows.

An instance of symmetric TSP, $TSP(C)$, is a *generalized convex-hull-and-line TSP* [59] if and only if there exist $0 < n_1 < n_2 < n$, such that

- (i) each of the matrices $C^{1,2}$, $C^{2,3}$ and $C^{1,3}$, obtained, respectively, from the principal submatrices of C on rows/columns $N^{1,2} = \{1, 2, \dots, n_1, n_1 + 1, n_1 + 2, \dots, n_2\}$, $N^{2,3} = \{n_1 + 1, \dots, n_2, n_2 + 1, n_2 + 2, \dots, n\}$ and $N^{1,3} = \{1, 2, \dots, n_1, n, n - 1, \dots, n_2 + 1\}$ by rearranging the rows and columns in the stated orders, is a Kalmanson matrix and
- (ii) node set $N_3 = \{n_2 + 1, n_2 + 2, \dots, n\}$ is interior to $C^{1,2}$.

It may be noted that this property of the matrix C can be defined purely in terms of its density matrix. Also, if C satisfies the generalized convex-hull-and-line condition, then we can perturb C to obtain a symmetric matrix \bar{C} satisfying the generalized convex-hull-and-line condition with the same values of n_1 and n_2 such that its corresponding principal submatrices $\bar{C}^{1,2}$, $\bar{C}^{2,3}$ and $\bar{C}^{1,3}$ are non-degenerate Kalman-son matrices and every optimal tour to $TSP(\bar{C})$ is also an optimal tour to $TSP(C)$. Hence, by Theorem 63 and Lemma 62, we get the following.

Theorem 70 [59] *If C satisfies the generalized convex-hull-and-line condition, then there exists an optimal tour γ to $TSP(C)$ such that*

- (i) *the nodes in the set $N^{1,2}$ appear in γ in the same order in which they appear in the definition of $N^{1,2}$;*
- (ii) *γ does not contain any edge (u, v) where u and v are non-consecutive nodes in N_3 ; and*
- (iii) *for any $(1 \leq u < v \leq n_1$ or $n_1 < v < u \leq n_2)$ and $n_2 < x, y \leq n$, if edges (u, x) and (v, y) lie in γ then $x \leq y$.*

Let ψ be the subtour $(1, 2, \dots, n_1, n_1 + 1, \dots, n_2, 1)$. From Theorem 70, it follows that there exists an optimal solution γ to $TSP(C)$ which is obtained by suitably partitioning the node set N_3 into simple chains on consecutively numbered nodes, $(i_0, \dots, i_1 - 1)$, $(i_1, \dots, i_2 - 1)$, \dots , $(i_k, \dots, i_{k+1} - 1)$ for some $n_2 + 1 = i_0 < i_1 < i_2 < \dots < i_k < i_{k+1} = n + 1$ and inserting each chain between some adjacent pair of nodes in the subtour ψ such that

- (i) *for any $(1 \leq u < v < n_1$ or $n_1 < v < u < n_2)$ and $0 \leq x, y \leq k$, if chain $(i_x, \dots, i_{x+1} - 1)$ is inserted between nodes u and $u + 1$ and chain $(i_y, \dots, i_{y+1} - 1)$ is inserted between nodes v and $v + 1$ then $x < y$ and if $1 \leq u < v < n_1$ then edges $(u, i_x), (u + 1, i_{x+1} - 1), (v, i_y), (v + 1, i_{y+1} - 1)$ are in γ while if $n_1 < v < u < n_2$ then edges $(u + 1, i_x), (u, i_{x+1} - 1), (v + 1, i_y), (v, i_{y+1} - 1)$ are in γ ;*
- (ii) *the only chain that can be inserted between the nodes n_1 and $n_1 + 1$ is $(i_k, \dots, i_{k+1} - 1)$; and*
- (iii) *the only chain that can be inserted between the nodes 1 and n_2 is $(i_0, \dots, i_1 - 1)$.*

As shown in [246] for the case of the convex-hull-and-line TSP, the problem of identifying an optimal tour of the above type can be reduced to the problem of finding a minimum arc-weight directed path in an acyclic digraph as follows.

For any ordered quadruplet (x, y, u, v) such that $n_2 < x \leq y \leq n$ and (u, v) is an edge in ψ , we say that the chain $(x, x + 1, \dots, y)$ is a candi-

date for insertion between the nodes u and v if insertion of this chain between the nodes u and v , such that the resultant subtour contains edges $(u, x), (v, y)$, does not violate the conditions (i), (ii) and (iii) above. In this case we call the ordered quadruplet (x, y, u, v) *admissible* and we associate with it a cost $w((x, y, u, v)) = c_{ux} + c_{vy} + (\sum_{i=x}^{y-1} c_{i,i+1}) - c_{uv}$. Now, let GA be an acyclic digraph with node set $V = \{n_2, n_2 + 1, \dots, n\}$ and arc set $E = \{e_{ij}^{uv} = (i, j) : n_2 \leq i < j \leq n; (i + 1, j, u, v) \text{ is admissible}\}$. We associate with every arc e_{ij}^{uv} in E a cost $w(e_{ij}^{uv}) = w((i + 1, j, u, v))$. For any tour γ on N obtained by partitioning the node set N_3 into simple chains on consecutive nodes and inserting each chain between some adjacent pair of nodes in ψ without violating the conditions (i), (ii) and (iii) above, there exists a directed path \wp in GA from node n_2 to node n such that

$$c(\gamma) = c(\psi) + \sum_{e \in \wp} w(e).$$

Conversely, consider any directed path $\wp = (e_{i_0, i_1}^{u_1, v_1} - \dots - e_{i_k, i_{k+1}}^{u_{k+1}, v_{k+1}})$ in GA , where $i_0 = n_2$ and $i_{k+1} = n$. If $(u_1, v_1) = (n_2, 1)$ or $(1, n_2)$, then insert the chain $(n_2 + 1, \dots, i_1)$ between nodes u_1 and v_1 as stated above. If $(u_{k+1}, v_{k+1}) = (n_1, n_1 + 1)$ or $(n_1 + 1, n_1)$, then insert the chain $(i_k + 1, \dots, n)$ between nodes u_{k+1} and v_{k+1} . For any other edge (u, v) in ψ , let $0 \leq [1] < \dots < [q] \leq k$ be such that for all $1 \leq j \leq q$, $(u_{[j]+1}, v_{[j]+1}) = (u, v)$. Insert the chain $(i_{[1]} + 1, \dots, i_{[1]+1}) - (i_{[1]+1}, i_{[2]} + 1) - (i_{[2]} + 1, \dots, i_{[2]+1}) - \dots - (i_{[q]} + 1, \dots, i_{[q]+1})$ between nodes u and v such that the edges $(u, i_{[1]} + 1)$ and $(i_{[q]+1}, v)$ are in the resultant subtour. It follows from the fact that C satisfies the generalized convex-hull-and-line condition, that for the resultant tour γ' , $c(\gamma') - c(\psi) \leq \sum_{e \in \wp} w(e)$. Thus an optimal tour can be obtained by finding a minimum arc-weight directed path in GA from n_2 to n and constructing the corresponding tour γ as just described. We now show that this minimum arc-weight directed path problem can be solved efficiently, using theorems 50 and 51.

Let $m = n - n_2 + 1$, $Y_1 = \{1, \dots, n_1 - 1\}$ and $Y_2 = \{n_1 + 1, \dots, n_2 - 1\}$. For every $u \in Y_1$, define an $m \times m$ matrix W^u as

$$w_{ij}^u = \begin{cases} w((n_2 + i, n_2 + j - 1, u, u + 1)), & \text{if } i < j \\ \infty, & \text{otherwise} \end{cases}.$$

Similarly, for every $v \in Y_2$, define an $m \times m$ matrix W^v as

$$w_{ij}^v = \begin{cases} w((n_2 + i, n_2 + j - 1, v + 1, v)), & \text{if } i < j \\ \infty, & \text{otherwise} \end{cases}.$$

Also, define for all $u \in Y_1 \cup Y_2$, and $j \in \{2, \dots, m\}$, $p(u, 1) = 0$ and

$$p(u, j) = p^j = \min\{\min\{g(x, j) : x \in Y_1 \cup Y_2\}, w((n_2 + 1, n_2 + j - 1, 1, n_2)), w((n_2 + 1, n_2 + j - 1, n_2, 1))\}.$$

These matrices and functions satisfy the conditions of Theorem 51 and therefore, the values $\{g(u, m) : u \in S_1 \cup S_2\}$ can be calculated in $O(mn_2)$. The final solution is p^* equal to

$$\min\{\min\{g(u, r) : u \in S_1 \cup S_2\}, \min\{p^j + w((n_2 + j, n, n_1, n_1 + 1)), p^j + w((n_2 + j, n, n_1 + 1, n_1)) : j \in \{1, \dots, r - 1\}\}\}.$$

This gives us an $O(n_2(n - n_2))$ scheme for the generalized convex-hull-and-line TSP, which is same as the complexity achieved in [246].

In [345], the complexity is reduced to $O(n)$ using Theorem 50 as follows. Let $r_1 = n_1 + 1$, $r_2 = n_2 - n_1 + 1$, $q = n - n_2$, $T_1 = \{(u_1 =)n_2, (u_2 =)1, \dots, (u_{r_1} =)n_1\}$ and $T_2 = \{(v_1 =)n_1, (v_2 =)n_1 + 1, \dots, (v_{r_2} =)n_2\}$. Let us define a $q \times r_1$ matrix A^h , a $r_1 \times q$ matrix B^h , a $q \times r_2$ matrix A^ℓ and a $r_2 \times q$ matrix B^ℓ as follows: for any $1 \leq i \leq q$ and $1 < j \leq r_1$,

$$a_{ij}^h = c_{n_2+i, u_j} - \sum_{x=n_2+1}^{n_2+i-1} c_{x, x+1};$$

$$b_{ji}^h = c_{n_2+i, u_{j+1}} + \left(\sum_{x=n_2+1}^{n_2+i-1} c_{x, x+1} \right) - c_{u_j, u_{j+1}}.$$

Similarly, for any $1 \leq i \leq q$ and $1 \leq j < r_2$,

$$a_{ij}^\ell = c_{n_2+i, v_{j+1}} - \sum_{x=n_2+1}^{n_2+i-1} c_{x, x+1};$$

$$b_{ji}^\ell = c_{n_2+i, v_j} + \left(\sum_{x=n_2+1}^{n_2+i-1} c_{x, x+1} \right) - c_{v_j, v_{j+1}}.$$

Then, for any $1 \leq i < j \leq q$ and $u_x \in T_1$ and $v_y \in T_2$,

$$w(n_2 + i, n_2 + j, u_x, u_{x+1}) = a_{ix}^h + b_{xj}^h \text{ and}$$

$$w(n_2 + i, n_2 + j, v_{y+1}, v_y) = a_{iy}^\ell + b_{yj}^\ell.$$

Each of the matrices A^h , B^h , A^ℓ and B^ℓ has a non-negative density matrix. Let $p(0) = 0$; and for all $1 \leq i \leq q$, let

$$g_1(i) = \min\{p(k) + a_{k+1, j}^h + b_{ji}^h : 1 \leq j \leq r_1; 0 \leq k < i\};$$

$$g_2(i) = \min\{p(k) + a_{k+1,j}^\ell + b_{ji}^\ell : 1 \leq j \leq r_2; 0 \leq k < i\};$$

and $p(i) = \min\{g_1(i), g_2(i)\}$. Then $p(q)$ is the optimal solution we seek. From Theorem 50 and the fact that in the algorithmic proof of Theorem 50 given in [345], the values of $g(1), \dots, g(q)$ are calculated sequentially, it follows that our problem can be solved in $O(n)$ time. We thus have the following theorem.

Theorem 71 *The generalized convex-hull-and-line TSP can be solved in $O(n)$ time.*

Another problem considered in [345] is the convex-hull-and-additional-point TSP. Here, there exists a point v such that if we remove that point, the remaining points lie on the boundary of a convex polygon. Let the neighbors of v in an optimal tour be points x and y . Then the problem gets reduced to that of finding a minimal cost Hamiltonian path between points x and y , given that all the points lie on the boundary of a convex polygon. Removal of points x and y divides the boundary of the polygon into two chains and it follows easily from Lemma 62 that the points in each chain occur in any optimal Hamiltonian path in the same order as in the chain. The problem thus reduces to optimally dividing one of the chains into subchains and inserting these subchains optimally between pairs of consecutive points in the other chain. As shown in [345], by following similar arguments as in the case of the convex-hull-and-line TSP, this problem can be posed as a minimum cost directed path problem in an acyclic graph and this can be solved in $O(n)$ using Theorem 50. For previous results on this case, the reader is referred to [441, 581]. We can generalize this problem to the case where the cost matrix C is such that by deleting row and column i from C , for some $i \in N$, we get a Kalmanson matrix. The same method works for this generalized case.

6.3. The tunneling TSP

Recently Barvinok et al [89] have shown that for any fixed dimension d , if $\text{dist}(x, y)$ is some polyhedral norm between points x and y in \mathbb{R}^d and we define $c_{ij} = -\text{dist}(x_i, x_j)$, where x_i and x_j are the points in \mathbb{R}^d corresponding to i and j , respectively, then the corresponding instance of TSP can be solved in polynomial time. In fact, they consider a larger subclass of symmetric TSP which they call Tunneling TSP. In this case, the symmetric cost matrix C is determined by a set $\{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\}$ of k ordered pairs called tunnels, for some constant, positive integer k . For each $i \in N$ and $u \in \{1, \dots, k\}$, the costs $c((i, s_u))$, $c((i, t_u))$ are specified. For any pair of distinct nodes i, j in N , we can travel from i

to j along any of the paths $\{(i, s_u, t_u, j), (i, t_u, s_u, j) : u \in \{1, \dots, k\}\}$ and $c_{ij} = \min\{\min\{c((i, s_u)) + c((j, t_u)), c((i, t_u)) + c((j, s_u))\} : u \in \{1, \dots, k\}\}$. Let

$$T = \cup_{u=1}^k \{s_u, t_u\}, \quad F = \{(s_u, t_u) : u \in \{1, 2, \dots, k\}\},$$

$$\bar{E} = \{(i, s_u), (i, t_u) : i \in N; u \in \{1, \dots, k\}\}.$$

Let us define the undirected graph $\bar{G} = (N \cup T, \bar{E} \cup F)$. Using arguments similar to ones used in Section 5.1, it can be shown that for any tour γ on N there corresponds a vector $x^\gamma \in \mathbb{Z}_+^{\bar{E} \cup F}$ such that

- i) $x^\gamma(e) = 0$ or 1 for all $e \in \bar{E}$ and $x^\gamma((s_u, t_u)) = \sum_{i=1}^n x^\gamma(i, s_u) = \sum_{i=1}^n x^\gamma((i, t_u))$ for all u ;
- ii) $\sum_{u=1}^k \{x^\gamma((i, s_u)) + x^\gamma((i, t_u))\} = 2$ for all i ;
- iii) the multigraph \bar{G}^{x^γ} is connected; and
- iv) the total edge cost of \bar{G}^{x^γ} is $c(\gamma)$, where each copy of edge (s_u, t_u) is assigned a cost 0 .

We call a vector x satisfying conditions (i), (ii) and (iii) a feasible vector. Conversely, any feasible vector x defines at least one tour γ such that $x = x^\gamma$. Our problem is thus equivalent to finding a feasible vector $x \in \mathbb{Z}_+^{\bar{E} \cup F}$ for which \bar{G}^x has a minimum total edge cost. An algorithm for this problem is presented in [89] which is based on the following facts.

- a) For any feasible vector x , $\sum_{u=1}^k x((s_u, t_u)) = 2n$ and, hence, the set $Y = \{y \in \mathbb{Z}_+^F : \text{there exists a feasible vector } x \text{ such that } y((s_u, t_u)) = x((s_u, t_u)) \text{ for all } u\}$ has cardinality $O(n^{k-1})$.
- b) For any $y \in Y$, let $T^y = \cup\{\{s_u, t_u\} : y((s_u, t_u)) > 0\}$. Let us represent T^y as $\{s_{u_1}, t_{u_1}, s_{u_2}, t_{u_2}, \dots, s_{u_r}, t_{u_r}\}$, where $1 \leq u_1 < \dots < u_r \leq k$ and for any $1 \leq j < r$, let $T_j^y = \cup\{\{s_{u_i}, t_{u_i}\} : i \in \{1, \dots, j\}\}$. Since for any feasible vector x , \bar{G}^x is connected, there exist, for the corresponding vector y and any $j \in \{1, \dots, r\}$, some $\alpha_j \in T_j^y$, $i_j \in N$ and $\beta_j \in T^y - T_j^y$ such that $x((\alpha_j, i_j)) = x((\beta_j, i_j)) = 1$. Let $J^x = \{(\alpha_1, i_1, \beta_1), (\alpha_2, i_2, \beta_2), \dots, (\alpha_r, i_r, \beta_r)\}$. There are $O(n^{k-1})$ possible choices of such a set J . Conversely, for any $x \in \mathbb{Z}_+^{\bar{E} \cup F}$ satisfying conditions (i) and (ii) for which such a set J^x exists, \bar{G}^x is connected.
- c) It follows from (b) that given such a pair of sets y and J , finding the corresponding feasible set x such that \bar{G}^x has minimum total edge cost is a weighted b -matching problem which can be solved in $O(n^3)$ time [270]. This gives us a scheme of overall time complexity $O(n^{2k+1})$ [89].

It is shown in [89] that for any fixed dimension d and any polyhedral norm determined by k vectors in R^d , the problem $TSP(C)$, where $c_{ij} = -dist(x_i, x_j)$ can be reduced to an instance of tunnel TSP with k tunnels in $O(dkn)$ time.

7. Generalized graphical TSP

We call the following generalization of TSP *generalized graphical TSP*. We are given a multigraph $G = (N, E)$, an edge cost function $c : E \times \{0, 1, 2\} \rightarrow \mathbb{R}_+$, and a parity function $par : N \rightarrow \{even, odd, neutral\}$. We call the node set $S = \{i : par(i) = neutral\}$ the set of Steiner nodes. The problem is to find $x \in \{0, 1, 2\}^E$ such that (i) $G^x = (N^x, E^x)$ is connected; (ii) for each node i , $deg(i)$ (the number of edges incident with i) is even if $par(i) = even$, odd if $par(i) = odd$ and 0 or even if $par(i) = neutral$; and (iii) the total cost $c(x) = \sum_{e \in E} c(e, x(e))$ is minimum. (Here, as in Section 5, G^x is defined as a multigraph where E^x containing $x(e)$ copies of each edge e and N^x is the set of nodes in N each one of which has at least one edge in E^x incident with it.)

A special case of this problem with $par(i) = even$ for all i is considered in [316]. We shall call it *even generalized graphical TSP (EGGTSP)*. The term *graphical TSP* was first introduced in [219], who considered a special case of EGGTSP with $c(e, i) = ic(e)$ for all $e \in E$ and $i \in \{0, 1, 2\}$. Their motivation for considering the problem was as follows. Suppose we define a symmetric cost matrix C as $c_{ij} = \text{length of the shortest path between nodes } i \text{ and } j \text{ in } G$. Then the problem $TSP(C)$ is equivalent to the graphical TSP on G .

We say that a class \mathcal{C} of multigraphs satisfies property \mathfrak{GP}_1 [316] if and only if for any multigraph G in \mathcal{C} the following three conditions hold.

- i) If e_1 and e_2 are parallel edges in G , (that is, they have the same end-nodes $\{u, v\}$), then the multigraph, obtained from G by replacing e_1 and e_2 by a new edge e (having the same end-nodes $\{u, v\}$) is in \mathcal{C} .
- ii) If e_1 and e_2 are series edges in G , (that is, for some distinct nodes u, v, w in N , $\delta(v) = \{e_1 = (v, u), e_2 = (v, w)\}$), then the multigraph obtained from G by replacing e_1 and e_2 by a new edge $e = (u, w)$ and deleting the isolated node v is in \mathcal{C} .
- iii) If G has neither parallel nor series edges, then any instance of EGGTSP on G can be solved in polynomial time.

Theorem 72 [316] *For any graph G in a class \mathcal{C} satisfying property \mathfrak{GP}_1 , any instance of EGGTSP on G can be solved in polynomial time.*

Proof. If the graph G does not contain any series or parallel edges, then the result follows from the definition of property \mathfrak{P}_1 .

If G contains parallel edges e_1 and e_2 having end-nodes u and v , then replace e_1 and e_2 by a new edge e having the same end-nodes u and v to obtain a smaller graph G' in \mathfrak{C} . Any instance of EGGTSP on G can be reduced to an instance of EGGTSP on G' by defining a cost function on e as follows :

$$\begin{aligned} c(e, 0) &= c(e_1, 0) + c(e_2, 0); \\ c(e, 1) &= \min\{c(e_1, 0) + c(e_2, 1), c(e_1, 1) + c(e_2, 0), \\ &\quad c(e_1, 2) + c(e_2, 1), c(e_1, 1) + c(e_2, 2)\}; \\ c(e, 2) &= \min\{c(e_1, 0) + c(e_2, 2), c(e_1, 2) + c(e_2, 0), c(e_1, 1) + c(e_2, 1), \\ &\quad c(e_1, 2) + c(e_2, 2)\}. \end{aligned}$$

If G contains series edges $e_1 = (u, v)$ and $e_2 = (v, w)$ for distinct nodes u, v and w , then replace e_1 and e_2 by a new edge $e = (u, w)$ and delete the isolated node v to obtain a smaller graph G' in \mathfrak{C} . Again, any instance of EGGTSP on G can be reduced to an instance of EGGTSP on G' by defining a cost function on e as follows :

$$\begin{aligned} c(e, 0) &= \min\{c(e_1, 0) + c(e_2, 2), c(e_1, 2) + c(e_2, 0)\}; \\ c(e, 1) &= c(e_1, 1) + c(e_2, 1); \\ c(e, 2) &= c(e_1, 2) + c(e_2, 2). \end{aligned}$$

Thus, in either case, the original problem can be reduced in polynomial time to an instance of EGGTSP on a smaller graph. The result follows by induction. ■

We now generalize this result. We say that a class \mathfrak{C} of multigraphs satisfies property $\mathfrak{SP}(k)$ for some integer k if and only if for any multigraph G in \mathfrak{C} the following three conditions hold.

- (i) If e_1 and e_2 are parallel edges in G , then the multigraph obtained from G by replacing e_1 and e_2 by a new edge e having the same end-nodes is in \mathfrak{C} .
- (ii) If $e_1 = (u, v)$ and $e_2 = (v, w)$ are series edges in G , then the multigraph obtained from G by replacing e_1 and e_2 by a new edge $e = (u, w)$ and deleting the isolated node v is in \mathfrak{C} .
- (iii) If G has neither parallel nor series edges, then the number of nodes in G is no more than k .

Theorem 73 *For any constant integer k and any graph G in a class \mathfrak{C} satisfying property $\mathfrak{SP}(k)$, any instance of generalized graphical TSP on G can be solved in polynomial time.*

Theorem 73 can be proved using similar arguments as in the proof of Theorem 72 by defining cost functions $c(e, u, p, q)$ for any edge $e = (i, j)$, any parities $p, q \in \{\text{even}, \text{odd}, \text{neutral}\}$ assigned to nodes i, j , respectively, and any $u \in \{0, 1, 2\}$. We leave the details as an exercise.

The class \mathfrak{C} of connected multigraphs satisfying property $\mathfrak{S}\mathfrak{P}(2)$ is precisely the class of *series-parallel multigraphs*. Theorem 74 below thus follows from Theorem 73.

Theorem 74 *The generalized graphical TSP on a series parallel multigraph with edge set E can be solved in $O(|E|^2)$.*

An $O(|E|)$ scheme for generalized graphical TSP on a series parallel graph with no nodes of odd parity and cost function $c(e, i) = ic(e)$ is given in [219].

A graph $G = (N, E)$, is called a *wheel* [220, 221] if and only if there exists an ordering of the node set N and a partition $E = E_S \cup E_R$, such that $E_S = \{(1, i) : i \in \{2, \dots, n\}\}$ and the edge set E_R forms subtour $(2, 3, \dots, n, 2)$. The node 1 is called the *hub* of the wheel, the edges in E_S and E_R are called the *spoke-edges* and the *rim-edges* of the wheel, respectively.

The following is a minor generalization of a result in [316].

Theorem 75 *Let x^* be an optimal solution to an instance of the generalized graphical TSP on a wheel $G = (N, E_S \cup E_R)$. Then, either $x^*(e) = 1$ for all $e \in E_R$ or there exists $f \in E_R$ such that the vector x' , defined as $x'(f) = 0$ and $x'(e) = x^*(e)$ for all $e \in E - \{f\}$, is a feasible solution to the problem.*

An $O(n^3)$ algorithm for the generalized graphical TSP on a wheel follows from this theorem and the fact that if we delete any rim-edge from a wheel, we get a series-parallel graph.

Let $G_1 = (V_1 \cup S, E_1 \cup F_1)$ and $G_2 = (V_2 \cup S, E_2 \cup F_2)$ be two multigraphs such that (S, F_1) and (S, F_2) are complete graphs on node set S . Let $|S| = r$. Then the multigraph $G = (V_1 \cup V_2 \cup S, E_1 \cup E_2)$ is called the *r-sum* of G_1 and G_2 by (S, F_1) and (S, F_2) ; and G_1 and G_2 are called the *S-components* of G [316].

We say that a class \mathfrak{C} of multigraphs satisfies property \mathfrak{P}_2 [316] if and only if for any G in \mathfrak{C} , the following two conditions hold.

- i) If G is 1-sum or 2-sum of G_1 and G_2 , then G_1 and G_2 belong to the class \mathfrak{C} .
- ii) If G cannot be expressed as a 1-sum or 2-sum of two graphs, then the EGGTSP on G can be solved in polynomial time.

Theorem 76 [316] *For any graph G in a class \mathfrak{C} satisfying property \mathfrak{P}_2 , any instance of EGGTSP on G can be solved in polynomial time.*

Proof. If the graph G cannot be expressed as a 1-sum or a 2-sum of two other graphs then the result follows by definition of property \mathfrak{P}_2 .

Suppose G is a 1-sum of two graphs G_1 and G_2 . In this case, any instance of EGGTSP on G decomposes into instances of EGGTSP on smaller graphs G_1 and G_2 .

Suppose G is a 2-sum of two graphs $G_1 = (V_1 \cup S, E_1 \cup F_1)$ and $G_2 = (V_2 \cup S, E_2 \cup F_2)$, where $S = \{u, v\}$, $F_1 = \{f_1 = (u, v)\}$ and $F_2 = \{f_2 = (u, v)\}$. In this case, a solution to an instance of EGGTSP on G can be obtained by solving three instances of the problem on G_1 and one on G_2 as follows.

Solve three corresponding instances of EGGTSP on G_1 , one for each of the three cost functions, c_0 , c_1 and c_2 assigned to the edge f_1 , where

$$\begin{aligned} c_i(f_1, i) &= 0 \quad \forall i = 1, 2, 3; \\ c_i(f_1, j) &= M \text{ (a large enough number)} \quad \forall i, j \in \{1, 2, 3\}, i \neq j. \end{aligned}$$

Let the corresponding optimal solutions be y_0, y_1 and y_2 , with optimal objective function values z_0, z_1 and z_2 , respectively. Now assign to edge f_2 a cost function $c(f_2, i) = z_{2-i}$ for $i = 1, 2, 3$ and solve the corresponding instance of EGGTSP on G_2 . Let \bar{y} be an optimal solution to this problem. Let $\bar{y}(f_2) = i$. Then the following vector x is an optimal solution to the instance of EGGTSP on G :

$$x(e) = \bar{y}(e) \quad \forall e \in E_2; \quad x(e) = y_{2-i}(e) \quad \forall e \in E_1.$$

It follows from decomposition results on graphs, (see e.g. [816]), that we can always choose $G_1 \in \mathfrak{C}$ such that G_1 cannot be expressed as a 1-sum or a 2-sum of two other graphs and that such a decomposition can be obtained in polynomial time [453]. The result now follows. ■

Theorem 76 has been extended in [316] to some special cases of 3-sums leading to polynomial schemes for EGGTSP on classes of graphs such as the propellers.

For any class \mathfrak{C} of multigraphs let us call the subclass of \mathfrak{C} , containing all the multigraphs in \mathfrak{C} which are not decomposable by 1-sum or 2-sum, as the basic set of \mathfrak{C} .

As a corollary of Theorem 76 we have the following :

Corollary 77 *Let \mathfrak{C} be a class of multigraphs such that for any multigraph G in \mathfrak{C} , if G is the 1-sum or 2-sum of G_1 and G_2 , then G_1 and G_2 belong to the class \mathfrak{C} . Suppose every multigraph in the basic set of \mathfrak{C}*

is either (i) the complete graph on 5 nodes, or (ii) the complete (3, 3)-bipartite graph, or (iii) the graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 6), (3, 5), (4, 5), (4, 6), (5, 6)\}$, or (iv) a wheel or (v) a propeller. Then \mathfrak{C} satisfies property \mathfrak{P}_2 and hence, for any G in \mathfrak{C} , EGGTSP on G can be solved in polynomial time.

Theorem 77 generalizes the results in [85, 317].

8. Solvable classes of TSP on specially structured graphs

So far, almost all the results on solvable classes discussed involved a cost matrix C having finite, non-diagonal entries. (Note that all the classes discussed in Section 7 involved cost matrices with finite non-diagonal entries, though these entries were defined using graphs with special structures.) Not many results exist on classes of graphs G for which $TSP(G, C)$ is solvable for all cost matrices C compatible with G . The only such non-trivial results that we are aware of are those reported in [220, 221]. We discuss these next.

Given a graph $G = (N, E)$ and any $\emptyset \neq S \subset N$, we call $\delta(S) = \{(i, j) : i \in S, j \in N - S\}$ a cutset of G . If $|S| = 1$ or $n - 1$ then we call $\delta(S)$ a trivial cutset of G . Else, we call it a non-trivial cutset. Set S is said to define a k -edge cutset on G if and only if $|\delta(S)| = k$. For any $S \subset N$ with $|S| > 1$, we denote by $G \times S$ the graph obtained from G by *shrinking* the node set S to a *new node* μ_S , (that is, the node set of $G \times S$ is $(N \cup \{\mu_S\}) - S$ and its edge set is $\{e : e = (i, j) \in E, \{i, j\} \subseteq N - S\} \cup \{(i, \mu_S) : (i, j) \in E, i \in N - S, j \in S\}$. Graph $G = (N, E)$ is said to be *3-connected* if and only if at least 3 nodes must be deleted in order to disconnect it.

Lemma 78 [221] *Suppose G is a 3-connected graph. Then it cannot have a k -edge cutset for $k < 3$. The set $X = \{S : |\delta(S)| = 3\}$ is nested, (that is, for any $S_1, S_2 \in X$, one of S_1 and $N - S_1$ is contained in one of S_2 and $N - S_2$. Suppose S is a minimal set such that $\delta(S)$ is a non-trivial 3-edge cutset. Then $G \times S$ does not contain any non-trivial 3-edge cutset. Such a minima set S can be found in strongly polynomial time by considering all sets of 3 edges of G .*

We say that a class \mathfrak{C} of 3-connected graphs defined on node set N satisfies property \mathfrak{P}_3 if and only if any graph G in \mathfrak{C} satisfies the following two conditions.

- (i) For every set $S \subset N$ defining a non-trivial 3-edge cutset on G , both $G \times S$ and $G \times (N - S)$ are in \mathfrak{C} .

- (ii) If G does not have non-trivial 3-edge cutsets then $TSP(G, C)$ can be solved in polynomial time for every cost matrix C compatible with G .

Theorem 79 [221] *For any graph G in a class \mathcal{C} satisfying property \mathfrak{P}_3 , any instance of TSP on G can be solved in polynomial time.*

Proof. If the graph G does not have a non-trivial 3-edge cutset then the result follows by definition of property \mathfrak{P}_3 . Else, let S be a minimal non-trivial 3-edge cutset in G and let $\delta(S) = \{e_1, e_2, e_3\}$. Every tour in G contains exactly two of the edges $\{e_1, e_2, e_3\}$. Recursively, find an optimal tour in $G \times (N - S)$ containing the edges e_1 and e_2 . (This can be done by changing the cost of the edge e_3 to some large number M .) Let the optimal tour and the corresponding optimal objective function value be $\gamma_{1,2}$ and $v_{1,2}$ respectively. If no such tour exists, then let $v_{1,2} = M$. Similarly, find $\gamma_{1,3}$, $v_{1,3}$, $\gamma_{2,3}$ and $v_{2,3}$. Now consider $G \times S$ and in this graph, assign to the edges e_1, e_2 and e_3 costs c'_1, c'_2 and c'_3 , respectively, such that

$$c'_i + c'_j = v_{i,j} \quad \forall i, j \in \{1, 2, 3\}, i \neq j.$$

This system of equations has a unique solution

$$c'_1 = \frac{1}{2}(v_{1,2} + v_{1,3} - v_{2,3}), \quad c'_2 = \frac{1}{2}(v_{1,2} + v_{2,3} - v_{1,3}), \quad c'_3 = \frac{1}{2}(v_{1,3} + v_{2,3} - v_{1,2}).$$

Solve the instance of TSP on $G \times S$ recursively to find an optimal tour ψ . Suppose ψ contains edges e_i and e_j for some $\{i, j\} \subset \{1, 2, 3\}$. Then an optimal solution to TSP on G can be obtained by combining the tour $\gamma_{i,j}$ on $G \times (N - S)$ and the tour ψ . ■

A planar graph $G = (N, E)$ is called a Halin graph if and only if we can partition the edge E into disjoint sets E_1 and E_2 such that $T = (N, E_1)$ is a tree with each non-leaf node of degree at least 3 and the edge set E_2 forms a cycle containing all the leaf nodes of T .

If T is a star, (that is, $E_1 = \{(1, i) : i \in \{2, 3, \dots, n\}\}$), then the corresponding Halin graph is a wheel. As discussed above, generalized graphical TSP on a wheel can be solved in $O(n^3)$ time. In the case of TSP on a wheel, the complexity can be reduced to $O(n)$ using the fact that the total number of tours in a wheel is $O(n)$ [220]. From Theorem 79 and the facts that (i) a Halin graph does not have a 3-edge cutset if and only if it is a wheel, and (ii) a minimal three edge cutset (called a fan, see Appendix A) in a Halin graph can be found in linear time [220], we get the following.

Corollary 80 [220] *TSP(G, C) can be solved in $O(n)$ time when G is a Halin graph.*

The results of [221] on TSP on graphs with 3-edge cutset have been extended in [490] to larger classes of graphs, including Halin graph + an additional node and in [437] to outer-facial graphs.

9. Classes of TSP with known compact polyhedral representation

Since the classical work of Edmonds on the matching problem on a general graph [266] and the matroid intersection problem [268], polyhedral combinatorics has developed as a significant area of research [747]. Here, we are interested in the study of the polytope corresponding to a combinatorial optimization problem that can be obtained by representing each solution to the problem as a point in the vector space of reals of appropriate dimension and taking the convex hull of this finite point set. A sufficient knowledge of the polytope associated with a combinatorial optimization problem leads to a polynomial and in some cases even strongly polynomial algorithm for the problem using the results in linear programming [320, 400, 494, 747, 786]. We investigate in this section subclasses of TSP for which polynomial schemes have been devised using this approach.

For a given digraph $G = (N, E)$, if we associate with every arc (i, j) in E a binary variable x_{ij} , then it is well-known [548] that any instance $TSP(G, C)$ can be formulated as an integer linear program (ILP) as follows.

$$\min \sum_{(i,j) \in E} c_{ij} x_{ij} \quad (3)$$

subject to

$$\sum_{\{(i,j) \in E\}} x_{ij} = 1 \quad \forall i \in N \quad (4)$$

$$\sum_{\{(i,j) \in E\}} x_{ij} = 1 \quad \forall j \in N \quad (5)$$

$$\sum_{\{(i,j) \in E; \{i,j\} \subseteq S\}} x_{ij} \leq |S| - 1 \quad \forall \emptyset \neq S \subseteq N \quad (6)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E \quad (7)$$

If G is an undirected graph (and therefore, the cost matrix C is symmetric), then we can associate a binary variable x_{ij} with every $1 \leq i < j \leq n$ such that $(i, j) \in E$ to get the following ILP formulation.

$$\min \sum_{\{i < j; (i,j) \in E\}} c_{ij} x_{ij} \tag{8}$$

subject to

$$\sum_{\{i < j; (i,j) \in E\}} x_{ij} + \sum_{\{j < i; (j,i) \in E\}} x_{ji} = 2 \quad \forall i \in N \tag{9}$$

$$\sum_{\{\{i,j\} \subseteq S; i < j; (i,j) \in E\}} x_{ij} \leq |S| - 1 \quad \forall \emptyset \neq S \subseteq N \tag{10}$$

$$x_{ij} \in \{0, 1\} \quad \forall i < j; (i, j) \in E \tag{11}$$

In the directed (asymmetric) case, if we replace the constraints (6) and (7) by the constraints (12) below,

$$0 \leq x_{ij} \leq 1 \quad \forall (i, j) \in E \tag{12}$$

then the polytope defined by constraints (4, 5, 12) is known as the *Assignment polytope* [7, 612]. In the undirected (symmetric) case, if we replace the constraints (10, 11) by the constraints (13) below,

$$0 \leq x_{ij} \leq 1 \quad \forall i < j; (i, j) \in E \tag{13}$$

then the polytope defined by constraints (9 and 13) is called the *Fractional 2-Matching polytope* [7, 612]. The polytopes defined by constraints (4, 5, 6, 12) in the asymmetric case and constraints (9, 10, 13) in the symmetric case are called *Subtour Elimination polytopes*. The assignment polytope has integer extreme points. However, it has an extreme point corresponding to every permutation on the N that has only non-trivial subtours. The fractional 2-matching polytope may have some fractional extreme points and its integer extreme points correspond to subgraphs of G which decompose the node set N into node-disjoint non-trivial subtours. In both the symmetric and asymmetric cases, the subtour elimination polytope may have some fractional extreme points. However, in this case every integer extreme point corresponds to a tour on N .

Since each of these polytopes is a relaxation of the traveling salesman polytope, minimizing the corresponding objective function (3) or (8) over each of these polytopes gives us a lower bound on the optimal objective function value to the corresponding instance of the TSP. The problem of optimizing objective function (3) over the assignment polytope is the well-known assignment problem for which efficient, strongly polynomial algorithms exist [7, 612]. The fractional 2-matching problem (8, 9, 13) can be solved in strongly polynomial time using the results in [400, 494,

786]. It can also be converted to a special case of minimum cost flow problem on a bipartite graph as follows. Let $N' = \{1', 2', \dots, n'\}$ and let $G' = (N \cup N', F)$ be a bipartite graph where $F = \{(i, j'), (j, i') : (i, j) \in E\}$. Consider the following minimum cost flow problem.

$$\min \sum_{(i,j') \in F} 1/2 c_{ij} y_{ij'} \quad (14)$$

subject to

$$\sum_{\{(i,j') \in F\}} y_{ij'} = 2 \quad \forall i \in N \quad (15)$$

$$\sum_{\{(i,j') \in F\}} y_{ij'} = 2 \quad \forall j' \in N' \quad (16)$$

$$0 \leq y_{ij'} \leq 1 \quad \forall (i, j') \in F \quad (17)$$

For any solution y to the problem (14, 15, 16, 17), the vector x defined as $x_{ij} = (y_{ij'} + y_{ji'})/2$ is a solution to the corresponding fractional 2-matching problem with the same objective function value. Conversely, for any solution x to an instance of the fractional 2-matching problem, if we define a vector y as $y_{ij'} = y_{ji'} = x_{ij} \quad \forall i < j, (i, j) \in E$, then y is a solution to the corresponding instance of the minimum cost flow problem with the same objective function value. The problem (14, 15, 16, 17) can be solved in $O(|F|(|F| + n \log n)) = O(|E|(|E| + n \log n))$ time [7]. Thus, the fractional 2-matching problem can be solved in $O(|E|(|E| + n \log n))$ time.

Though the total number of constraints of each of the types (6) and (10) is exponential in n , in each of these cases the problem of checking if a non-negative solution x satisfies all these constraints and finding a violated constraint if one exists is the well-solved minimum cut problem on a (di)graph. The problem of optimizing (3) or (8) over the asymmetric or symmetric subtour elimination polytope, respectively, which we call the subtour elimination LP, can thus be solved in strongly polynomial time using the Ellipsoid method [400] combined with the scheme of Frank and Tardos [320]. However, practical experience indicates that using simplex method together with cutting plane approach works faster [209]. Thus, if for a subclass of (symmetric) cost matrices C , the corresponding (fractional 2-matching) assignment problem or the subtour elimination LP can be shown to have an optimal solution that is a tour and that can be obtained in polynomial time, then we get a polynomially solvable case of the TSP. In this section we discuss existing results on subclasses of the pairs (G, C) for which one of these relaxed problems has an optimal solution which is a tour and which can be polynomially computed.

A graph G is an *elementary graph* [219] if and only if its fractional 2-matching polytope is its tour polytope, (that is, the extreme points of the associated fractional 2-matching polytope correspond to the tours in G). The following basic result in polyhedral combinatorics allows us to identify some classes of elementary graphs.

Lemma 81 [625] *If x is an extreme point of the fractional 2-matching polytope, then $x \in \{0, 1/2, 1\}^{|E|}$ and the edge set $F = \{e : e \in E; x_e = 1/2\}$ partitions into edge sets of an even number of node-disjoint odd subtours.*

As corollary of Lemma 81, we get the following.

Corollary 82 [219] *If a graph G does not have two node-disjoint odd subtours, then G is an elementary graph.*

It follows from Corollary 82 that (i) any graph on five or fewer nodes is an elementary graph and (ii) a wheel (for definition, see Section 7) is an elementary graph. The following operation is defined in [219] as a tool of generating new classes of graphs, for which the TSP polytope can be represented as a system of a polynomial (in n) number of linear inequalities, from known such graphs.

Let $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ be two graphs on disjoint node sets N_1 and N_2 and let $u \in N_1$ and $v \in N_2$ be nodes of degree 3. Let the edges of G_1 and G_2 incident with nodes u and v be $\{(x_1, u), (x_2, u), (x_3, u)\}$ and $\{(y_1, v), (y_2, v), (y_3, v)\}$, respectively. We say that a graph $G = (N, E)$ is obtained from G_1 and G_2 by *3-splicing* if and only if $N = N_1 \cup N_2 - \{u, v\}$ and $E = E_1 \cup E_2 \cup \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} - \{(x_1, u), (x_2, u), (x_3, u), (y_1, v), (y_2, v), (y_3, v)\}$. We denote this by $G = G_1^u * G_2^v$.

Theorem 83 [219] *Suppose $G = G_1^u * G_2^v$, and N_1 and N_2 are node sets of G_1 and G_2 , respectively. Then a linear system of inequalities defining the tour polytope (convex hull of tours) of G is obtained by taking the union of linear systems defining the tour polytopes of $G \setminus (N_2 - \{v\})$ and $G \setminus (N_1 - \{u\})$.*

The proof of Theorem 83 is straightforward and is left to the reader.

For any family \mathcal{G} of graphs, let us define *closure*(\mathcal{G}) as the class of graphs obtained by repeatedly applying the 3-splicing operation to pairs of graphs in \mathcal{G} . Theorem 83 gives compact representation of the tour polytope of any graph in *closure*(\mathcal{G}) when \mathcal{G} is a set of elementary graphs. It may be noted that when \mathcal{G} is the set of wheel graphs, *closure*(\mathcal{G}) is precisely the set of Halin graphs [219]. Cornuejols et al [219] give a compact representation of the tour polytope of another class of graphs

which they call prisms. For $n = 2p$ for some integer $p \geq 3$, a graph $G = (N, E)$ is called a *prism* if and only if $E = E^1 \cup E^2 \cup \{(i, p+i) : i \in \{1, \dots, p\}\}$, where E^1 and E^2 are edge sets of subtours $(1, 2, \dots, p, 1)$ and $(p+1, p+2, \dots, 2p, p+1)$, respectively. Theorem 83 thus gives us the tour polytope of prismatic graphs which are the graphs in $\text{closure}(\mathcal{G})$ when \mathcal{G} is the class of prisms.

There are instances $TSP(G, C)$ in which we do not have total knowledge of the tour polytope corresponding to the given graph G but the given cost matrix C has a nice structure which enables us to solve the problem $TSP(G, C)$. The simplest such case is when the assignment problem, or in the symmetric case the fractional 2-matching problem, has a unique optimal solution which is a tour. This is then obviously the unique optimal solution to the corresponding instance of TSP and can be computed in strongly polynomial time by solving the corresponding assignment or fractional 2-matching problem. The following is a basic result in linear programming for which a simple and short proof is available [611].

Theorem 84 *For a given pair (G, C) of a digraph $G = (N, E)$ and a matrix C compatible with G , a tour γ in G is unique optimal solution to the assignment problem (3,4,5,12) if and only if there exist vectors $\mathbf{r}, \mathbf{s} \in \mathbb{R}^n$ such that :*

$$c'_{ij} = c_{ij} - r_i - s_j \begin{cases} \leq 0 & \forall (i, j) \in \gamma \\ > 0 & \forall (i, j) \in E; (i, j) \notin \gamma \end{cases} \quad (18)$$

If the cost matrix C is symmetric, (that is, G is an undirected graph), then a tour γ is unique optimal solution to the fractional 2-matching problem (8,9,13) if and only if there exists a vector $\mathbf{r} \in \mathbb{R}^n$ such that :

$$c'_{ij} = c_{ij} - r_i - r_j \begin{cases} \leq 0 & \forall (i, j) \in \gamma \\ > 0 & \forall (i, j) \in E; (i, j) \notin \gamma \end{cases} \quad (19)$$

In the directed case, the problem of checking whether a triplet (G, C, γ) satisfies the condition of Theorem 84 is equivalent to the problem of checking if there exists a non-positive directed cycle in an auxiliary digraph and this can be solved in $O(n|E|)$ [7, 612]. If there is a non-positive directed cycle, then γ is not unique optimal solution to the corresponding assignment problem. Else most of the efficient schemes for checking existence of non-positive directed cycle provide us with the required vectors \mathbf{r} and \mathbf{s} . In the undirected case, we can construct the corresponding minimum cost flow problem on node set $N \cup N'$ as stated before and use the standard approach in network flows [7, 612] to check if there is a

non-positive directed cycle in the corresponding auxiliary graph. If our scheme outputs vectors \mathbf{r}' and \mathbf{s}' , then the required vector \mathbf{r} is given by $\mathbf{r} = (\mathbf{r}' + \mathbf{s}')/2$.

The special case of this corresponding to Euclidean TSP is considered in [264, 712, 731, 737]. In [264], independent proofs have been given for the results mentioned above and in addition the following nice observation has been made.

Lemma 85 [264] *Let C be a symmetric cost matrix with finite non-diagonal entries, satisfying triangle inequality. Suppose a tour γ on N is the unique optimal solution to the fractional 2-matching problem on C . Then for $n > 3$, every vector $\mathbf{r} \in \mathbb{R}^n$ satisfying condition (19) is a non-negative vector.*

Proof. Suppose $\mathbf{r} \in \mathbb{R}^n$ satisfies condition (19) and $\mathbf{r}_1 < 0$. Let $i, j \in N$ be such that $(1, i), (1, j) \in \gamma$. Since \mathbf{r} satisfies condition (19), $\{\mathbf{r}_1 + \mathbf{r}_i \geq c_{1,i}\}$ and $\{\mathbf{r}_1 + \mathbf{r}_j \geq c_{1,j}\}$. This implies that $\mathbf{r}_i \geq c_{1,i}$ and $\mathbf{r}_j \geq c_{1,j}$ and by triangle inequality it follows that $\{\mathbf{r}_i + \mathbf{r}_j \geq c_{ij}\}$. But this contradicts the condition (19). This proves the result. ■

In the Euclidean case, the above vector \mathbf{r} has an interesting interpretation. For each $i \in N$, let P_i be the corresponding point in the plane and let R_i be a disk with centre P_i and radius \mathbf{r}_i . Then the disk R_i intersects precisely 2 disks R_j and R_k , where $(i, j), (i, k) \in \gamma$. The tour γ is thus the intersection graph of the n disks and the disks resemble a necklace. Hence this condition is called “Necklace Condition” [264, 712, 731, 737]. The following interesting results in [264] reduce to $O(n^2)$ the complexity of the problem of checking if a given cost matrix C of an instance of Euclidean TSP and a given tour γ on N satisfy the condition of Theorem 84.

For a non-negative cost matrix C , let s_{ij} be the total cost of the least cost path between nodes i and j . For any $1 \leq m \leq n - 1$ and any $i \in N$, let $d^m(i) = \min\{x : |\{j : j \in N - \{i\}; s_{ij} \leq x\}| \geq m\}$. Let $G^m = (N, E^m)$, where $E^m = \{(i, j) : s_{ij} \leq d^m(i) + d^m(j)\}$. Lemma 86 below follows from Theorem 84 and Lemma 85.

Lemma 86 [264] *Let C be the cost matrix of an instance of Euclidean TSP. Let C' be the matrix obtained from C by replacing the (i, j) th entry by ∞ for all $(i, j) \notin E^2$. Let γ be a tour on N . If the triplet (G, C, γ) satisfies the condition of Theorem 84, then γ is a tour in G^2 and the triplet (G^2, C', γ) also satisfies the condition. Conversely, if γ is a tour in G^2 and (G^2, C', γ) satisfies the condition of Theorem 84, then (G, C, γ) also satisfies the condition.*

Theorem 87 [264] *For an instance of Euclidean TSP and any $1 \leq m \leq n - 1$, the total number of edges in the corresponding graph G^m is at most $(31m - 1)n$. Furthermore, the graph G^m can be constructed in $O(m^2n)$ using the m th-order Voronoi diagram in the plane [673].*

For a proof of Theorem 87, we refer the reader to [264].

Chvatal [197] proved that for an undirected graph G on N and an arbitrary cost matrix C compatible with G , the problem of checking if the corresponding subtour elimination LP has an integer optimal solution is NP-hard. This diminishes considerably the chances of existence of a polynomially testable characterization of the class of pairs (G, C) for which the subtour elimination LP has a tour as an optimal solution. We shall have a look at two interesting cases reported in literature for which the corresponding subtour elimination LP has a unique optimal solution which is a tour. For this, we need an interesting property of the subtour elimination LP which we discuss first.

For any node $v \in N$, an undirected graph $T_v = (N, F)$ is called a v -tree [444] if and only if the graph $T_v[N - \{v\}]$, obtained by deleting the node v from T_v , is a spanning tree on $N - \{v\}$ and there are precisely two edges in F incident to the node v . Every tour on N is a v -tree for any node $v \in N$.

For any undirected graph G on N and any $v \in N$,

(i) the constraint set (10) can be replaced by the constraints only corresponding to subsets S of $N - \{v\}$ without changing the corresponding subtour elimination polytope; and

(ii) the constraint (20) below is a redundant constraint for the subtour elimination LP and hence can be added to the set of constraints of the subtour elimination LP without changing the problem.

$$\sum_{\{v \notin \{i,j\}; i < j; (i,j) \in E\}} x_{ij} = |N| - 2 \tag{20}$$

If we assign a potential variable λ_i to the constraint corresponding to node i in (9) for each i and dualize these constraints, the resultant problem

$$\max\{2 \sum_{i \in N} \lambda_i + z(C, \lambda) : \lambda \in \mathbb{R}^n\} \tag{21}$$

has the same optimal objective function value as the subtour elimination LP. Here,

$$z(C, \lambda) = \min\left\{ \sum_{\{i < j; (i,j) \in E\}} (c_{ij} - \lambda_i - \lambda_j)x_{ij} : x \text{ satisfies (10) for } S \subseteq N - \{v\}, (20), (9) \text{ for } i = v, \text{ and (13)} \right\}. \tag{22}$$

The polytope defined by constraints (10) (only for $S \subseteq N - \{v\}$), (20), (9) (only for $i = v$) and (13) is the convex hull of all the v -trees (see [209]). Thus, the optimal objective function value of the subtour elimination LP is the same as that of:

$$\max\{2 \sum_{i \in N} \lambda_i + \min\{ \sum_{(i,j) \in F} (c_{ij} - \lambda_i - \lambda_j) : (N, F) \text{ is a } v\text{-tree}\} : \lambda \in \mathbb{R}^n\} \tag{23}$$

This is precisely the Held-Karp lower bound on the optimal objective function value of $TSP(G, C)$ [209, 444]. From this and standard results in linear programming, we get the following.

Theorem 88 [209] *Let $TSP(C)$ be an instance of symmetric TSP. If for some vector λ^* of node potentials, a tour γ on N is an optimal solution to the problem (23) for some $v \in N$, then*

- (i) γ is an optimal solution to $TSP(C)$;
- ii) if x is an optimal solution to the corresponding subtour elimination LP, then x is an optimal solution to the problem (22) with $\lambda = \lambda^*$ and therefore for any $1 \leq i < j \leq n$ such that $x_{ij} > 0$ there exists a v -tree T on N containing the edge (i, j) such that T is optimal to the problem (22) with $\lambda = \lambda^*$.
- (iii) If γ is the only solution in the subtour elimination polytope that is an optimal solution to (22) with $\lambda = \lambda^*$, then γ is a unique optimal solution to $TSP(C)$.

Case I : The following subclass of symmetric TSP was introduced by Papadimitriou and Steigliz [654] as the one for which most of the known exchange-type solution improvement heuristic algorithms require exponential number of iterations in the worst case and they appropriately termed it “traps”.

Here $n = 8k$ for some integer $k \geq 4$. The node set N is divided into k sets $\{N^i = \{8i + 1, 8i + 2, \dots, 8(i + 1)\} : 0 \leq i < k\}$. The edge set E of complete graph on N is partitioned into four sets as follows: Let $NS = \{9, 17, \dots, 8(k - 1) + 1\} \cup \{5, 13, \dots, 8(k - 1) + 5\}$ and let F be the edge set of complete graph on node set NS . Then, E^0 is the edge set of the tour $(3, 2, 1, 8, 4, 5, 6, 7, 11, 10, \dots, 8(k - 1) + 3, 8(k - 1) + 2, 8(k - 1) + 1, 8k, 8(k - 1) + 4, 8(k - 1) + 5, 8(k - 1) + 6, 8(k - 1) + 7, 3)$; $E^1 = F - \{(8i + 1, 8i + 5) : i = 1, \dots, k - 1\} \cup \{(8i + 3, 8i + 4), (8i + 7, 8i + 8) : i = 0, \dots, k - 1\}$; $E^2 = \{(1, v) : v \in NS\}$; and $E^3 = E - (E^0 \cup E^1 \cup E^2)$. The symmetric cost matrix C is defined as

$c_{ij} = 1$ for $(i, j) \in E^0$; $c_{ij} = 0$ for $(i, j) \in E^1$; $c_{ij} = M$ for $(i, j) \in E^2$; and $c_{ij} = 2M$ for $(i, j) \in E^3$,

where M is an integer larger than $(3k + 1)$. (See Figure 11.8. Here, the arcs in E^0 are denoted by full lines.)

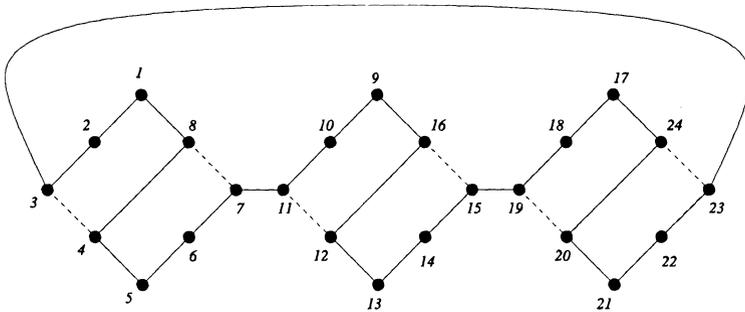


Figure 11.8. Papadimitriou-Steiglitz trap

Papadimitriou and Steiglitz [654] showed that for this problem, γ is the unique optimal tour with total cost $8k$ and there are $2^{k-1}(k - 1)!$ next best tours, each with a cost of $(M + 2k)$ and each of which differs from the optimal tour by exactly $3k$ edges. Padberg and Sung [649] proved the following interesting result.

Theorem 89 [649] For the “Papadimitriou-Steiglitz trap” with $k \geq 4$, $\gamma = (N, E^0)$ is the unique optimal solution to corresponding subtour elimination LP and thus the problem can be solved in strongly polynomial time.

Proof. We shall give a modified version of the proof of [649]. Thus, consider the following vector of node potentials:

$$\lambda_1 = \lambda_4 = \lambda_6 = 3k, \lambda_2 = 3k + 1, \lambda_3 = -1 + 3(k - \lceil (k + 1)/2 \rceil),$$

$$\lambda_7 = 3 \lceil (k + 1)/2 \rceil - 1, \lambda_8 = \lambda_3 + 1;$$

$$\forall 1 \leq i \leq \lceil (k + 1)/2 \rceil - 1,$$

$$\lambda_{8i+4} = \lambda_{8i+6} = 3(\lceil (k + 1)/2 \rceil - i), \lambda_{8i+2} = 2 + \lambda_{8i+4}, \lambda_{8i+3} = -1 - \lambda_{8i+4},$$

$$\lambda_{8i+7} = -1 + \lambda_{8i+4}, \lambda_{8i+8} = 1;$$

$$\text{for } i = \lceil (k + 1)/2 \rceil, \lambda_{8i+2} = \lambda_{8i+6} = 2, \lambda_{8i+3} = \lambda_{8i+7} = -1, \lambda_{8i+4} = \lambda_{8i+8} = 1;$$

$$\forall \lceil (k + 1)/2 \rceil + 1 \leq i < k,$$

$$\lambda_{8i+2} = \lambda_{8i+8} = 1 + 3(i - \lceil (k + 1)/2 \rceil); \lambda_{8i+3} = \lambda_{8i+2} - 2; \lambda_{8i+4} = 1;$$

$$\lambda_{8i+6} = \lambda_{8i+2} + 1; \lambda_{8i+7} = -1 - 3(i - \lceil (k + 1)/2 \rceil).$$

All other $\lambda(i)$'s have value zero.

It can be verified, (using Prim’s algorithm for the minimum cost spanning tree problem starting from node 3) that for this vector of node potentials, γ is an optimal v -tree for $v = (8k - 1)$. Thus by Theorem 88, it follows that γ is an optimal solution to the subtour elimination LP.

To show that γ is a unique optimal solution to the subtour elimination LP, we observe (using the matroidal properties of the set of spanning trees of a graph [821]), that for this vector of node potentials, every optimal v -tree contains only edges in $E^0 \cup E^1$. Hence by Theorem 88 it follows that for any other optimal solution x to the subtour elimination LP, $x_{ij} > 0$ implies that $(i, j) \in E^0 \cup E^1$. Since $c_{ij} = 0$ for $(i, j) \in E^1$ and $c_{ij} = 1$ for $(i, j) \in E^0$, for any vector x in the subtour elimination polytope having positive entries only corresponding to some elements of $E^0 \cup E^1$ and having a positive entry corresponding to at least one element of E^1 , the total objective function value is less than $8k$. But this violates the optimality of γ (which has objective function value $8k$). Hence, x can be optimal only if the edges corresponding to the positive entries of x belong to E^0 . ■

Case II : Althaus and Mehlhorn [20] consider the following problem in Computational geometry which is known as the *curve reconstruction problem*. We are given a finite sample V of n points on an unknown, non-self-intersecting closed curve γ in \mathbb{R}^2 . The problem is to connect the points in V in the order in which they lie on γ .

Giesen [359] showed that if the curve γ is benign and semi-regular then there exists an $\epsilon > 0$ such that if for every $x \in \gamma$ there exists a $y \in V$ with $\|(x, y)\| \leq \epsilon$ then the optimal solution to the Euclidean TSP corresponding to the point set V solves the curve reconstruction problem. Here, a curve is said to be semi-regular if a left and a right tangent exists at every point of the curve; a semi-regular curve is said to be benign if the turning angle (angle between the left and right tangents) at every point of the curve is less than π ; and for any two points x and y in the plane $\|(x, y)\|$ is the Euclidean distance between the points x and y . Giesen did not give any scheme for finding the value of ϵ for a given curve nor did he give any scheme to find the optimal solution to the corresponding instance of TSP. Althaus and Melhorn [20] give a constructive scheme to associate a value $\lambda^*(x)$ with every point x on a given closed, benign, semi-regular curve γ and show that for any sample $V = \{v_1, \dots, v_n\}$ of points on the curve such that the following two assumptions are satisfied, the problem

$$\min\left\{ \sum_{(i,j) \in F} (c_{ij} - \lambda^*(v_i) - \lambda^*(v_j)) : (N, F) \text{ an } i\text{-tree of } G \text{ for some } i \in N \right\}$$

has a unique optimal solution which is a tour, where $G = (N, E)$ is a complete graph on N and for $i, j \in N$, $c_{ij} = \|(v_i, v_j)\|$.

Assumption 1: For any two adjacent (on γ) sample points v_i and v_j , $c_{ij} - \lambda^*(v_i) - \lambda^*(v_j) \leq 0$.

Assumption 2: For any two adjacent sample points v_i and v_j , $\gamma[v_i, v_j]$ turns by less than π , where $\gamma[v_i, v_j]$ is the subcurve of γ with end points v_i and v_j and containing no other point in V .

10. Other solvable cases and related results

We now discuss some other interesting solvable cases. In some of these cases, the problem is transformed in to one of finding a minimum cost Eulerian graph.

10.1. TSP with multiple copies of few types of nodes

Here, every one of the n nodes belongs to one of the given k types, for a small, fixed integer k and we have n_i nodes of type i for each $i \in \{1, \dots, k\}$. We are given a $k \times k$ matrix A and for any pair i, j of nodes, $c_{ij} = a_{uv}$ where nodes i and j are of types u and v , respectively. An application of this case, given in [675], is that of n aircrafts waiting to land on a single runway. Every aircraft belong to one of a few categories and the time-gap between the landing of two aircrafts depends on the types of the two aircrafts. Other applications are reported in [807]. An algorithm for this problem is given in [226], which can be implemented in time $O(f(k))$ or $O(g(k)\log(n))$ where $f(\cdot)$ and $g(\cdot)$ are exponential functions. This algorithm is based on the following five facts.

(i) With every tour, we can associate a connected, degree balanced, directed pseudograph on node set $K = \{1, \dots, k\}$ with $d^-(i) = d^+(i) = n_i$ for all $i \in K$ and the directed pseudograph $G^* = (K, E^*)$ so associated with an optimal tour has a minimum total arc cost, where for any $\{i, j\} \subseteq K$ (including the case $i = j$), cost of every copy of the arc (i, j) is a_{ij} .

(ii) For such an optimal directed pseudograph G^* , the arc multiset E^* can be partitioned into two multisets $\{E_0^*, E_1^*\}$ such that $G_0^* = (K, E_0^*)$ is a minimal, connected, degree balanced, directed multigraph and $G_1^* = (K, E_1^*)$ is a minimum cost directed pseudograph with $d^-(i) = d^+(i) = n_i - \sigma_i$ for all $i \in K$, where σ_i is the indegree of node i in G_0^* .

(iii) Let σ_i be the indegree of node i in G_0^* for each i . If in G^* , we replace E_0^* by any multiset E_0^σ such that (K, E_0^σ) is a connected, degree balanced, directed multigraph with $d^-(i) = \sigma_i$ for all $i \in K$, we get an alternate feasible solution. Hence, G_0^* is a minimum cost connected, directed, degree balanced multigraph with $d^-(i) = \sigma_i$ for all $i \in K$.

(iv) Given the vector $\sigma = (\sigma_1, \dots, \sigma_k)$ of node indegrees, a connected, degree balanced, directed multigraph $G_0^\sigma = (K, E_0^\sigma)$ with minimum total arc cost can be obtained in $O(k^2\Pi(\sigma_i + 1))$ time using the following

dynamic programming recursion. For any vector $b = (b_1, \dots, b_k)$ and any $i \in K$, let us define a vector b^i as $b^i_i = b_i - 1$, $b^i_j = b_j$ for every other $j \in K$. Let $E^{(b,i)}$ be the arc multiset with minimum total cost $a(E^{(b,i)}) = \sum \{a_{ij} : (i, j) \in E_0^{(b,i)}\}$ such that $G^{(b,i)} = (K, E^{(b,i)})$ has an Eulerian trail from node 1 to node i and in $G^{(b,i)}$, $d^-(j) = b_j$ for all $j \in \{2, \dots, k\}$ and $d^-(1) = b_1 - 1$. Then $a(E^{(b,i)}) = \min\{a(E^{(b^i,j)}) + a_{ji} : j \in K\}$ with the initial condition that for any vector b such that for some $i \in K$, $b_1 = b_i = 1$ and $b_u = 0$ for every other $u \in K$, $a(E^{(b,i)}) = a_{1,i}$. The required solution (minimum cost of connected, degree balanced, directed multigraph with indegree vector σ) is then $\min\{a(E^{(\sigma,i)}) + a_{1,i} : i \in K\}$.

(v) Given a vector $(\sigma_1, \dots, \sigma_k)$, the problem of obtaining a minimum cost directed multigraph $G_1^\sigma = (K, E_1^\sigma)$ with $d^-(i) = d^+(i) = n_i - \sigma_i$ for all $i \in K$, is the transportation problem which can be solved in $O(k^4 \log k)$ or $O(k^3 \log n)$ time [7, 612].

The following theorem is crucial for achieving the required computational complexity.

Theorem 90 [226] *There exists a minimal degree balanced, connected, directed multigraph on node set K with node indegree vector $\sigma > 0$ if and only if at least $\max\{\sigma_i : i \in K\}$ entries of vector σ have value 1.*

Proof. The necessity follows from Theorem 57 and the facts that (i) in any decomposition of the arc set of a minimal, connected, degree balanced, directed multigraph into arc-disjoint, simple, directed cycles (subtours), every subtour in the decomposition has at least one node that does not belong to any other subtour in the decomposition; and (ii) there are at least $\max\{\sigma_i : i \in K\}$ subtours in any such decomposition.

The sufficiency can be proved constructively as follows. Given any such vector σ , if $\max\{\sigma_i : i \in K\} > 1$, then form a subtour passing through all the nodes $\{i : \sigma_i > 1\}$ and exactly one node j with $\sigma_j = 1$; subtract 1 from σ -values corresponding to all the nodes in the subtour and repeat the process. If there is no σ_i value more than 1, then form a subtour passing through all the nodes having $\sigma_i = 1$. ■

Theorem 90 implies that $\max\{\sigma_i : i \in K\} < k$ and it reduces the number of possible vectors $\sigma = \{\sigma_i : i \in K\}$ to be considered to a moderately growing exponential function $f(k)$ of k . The algorithm of [226] for computing an optimal directed pseudograph $G^* = (K, E^*)$ is thus as follows. For each choice of vector $\sigma > 0$ such that at least $\max\{\sigma_i : i \in K\}$ of the σ_i 's have value 1, find a minimum cost, connected, degree balanced, directed multigraph $G_0^\sigma = (K, E_0^\sigma)$ and a minimum cost directed pseudograph $G_1^\sigma = (K, E_1^\sigma)$. Let $G^\sigma = (K, E_0^\sigma \cup E_1^\sigma)$. Amongst these directed pseudographs G^σ , one with minimum total arc cost is the

required optimal solution. The overall complexity of this scheme is then $O(f(k))$ or $O(g(k) \log n)$, depending on the algorithm used to solve the transportation problems.

It is shown in [807] that a straightforward integer programming formulation of the problem of finding the optimal, connected, degree balanced, directed pseudograph $G^* = (K, E^*)$ with $d^-(i) = d^+(i) = n_i$ for all $i \in K$ can be solved in $O(p(\log n))$ using the algorithm of [554], where $p(\cdot)$ is a polynomial function. It may be noted that explicitly obtaining an optimal tour γ^* on N from the optimal directed pseudograph $G^* = (K, E)$ will require $O(n)$ effort. The following more compact way to representing the tour is given in [807]. Find a subtour in G^* using any method in network flows. Assign this subtour a capacity equal to the smallest multiplicity of the arcs in the subtour and delete those many copies of each of the arcs in the subtour from G^* . Repeat the process. We leave the details to the reader.

10.2. Sequencing jobs requiring few types of common resources

In this case too, each node belongs to one of the given k types, for some small, fixed integer k . For each $i \in \{1, \dots, k\}$, let N_i be the set of nodes of type i and let $|N_i| = n_i$. With each node i is associated an ordered pair of numbers (a_i, b_i) . For any pair i, j of distinct nodes, $c_{ij} = a_i$ if the nodes i and j are of the same type and $c_{ij} = b_j$ if the two nodes are of two different types. An example given in [806] is that of sequencing n jobs on a single machine, where processing jobs of each type i requires a particular template. If job i is immediately succeeded by job j , then if the two jobs are of the same type, the same template has to be re-used and the job j has to wait until the job i is done with the usage of the template. Thus the change-over time is a_i . If the two jobs are of different types, then the template of the second type has to be set up for job j , which requires time b_j .

For any $i \in \{1, \dots, k\}$, let $X_i = \{a_j : j \in N_i\}$, $Y_i = \{b_j : j \in N_i\}$, and let $Z = X_1 \cup \dots \cup X_k \cup Y_1 \cup \dots \cup Y_k$. For any set $S \subseteq Z$, let $X(S, i) = S \cap X_i$, $Y(S, i) = S \cap Y_i$, $x(S, i) = |X(S, i)|$ and $y(S, i) = |Y(S, i)|$. An $O(n \log n)$ time algorithm for this problem is given in [806], which is based on the following results.

(i) Costs of the n arcs in any tour on N are n distinct numbers from the set Z . We shall call the set of n arc costs corresponding to any tour γ on N , a tour set and we shall denote it by Z^γ .

(ii) Van der Veen et al [806] prove that a set $S \subset Z$, with $|S| = n$ is a tour set if and only if for every $i \in \{1, \dots, k\}$, (a) $x(S, i) + y(S, i) = n_i$;

(b) $y(S, i) \leq \sum_{j \neq i} y(S, j)$ and (c) either $y(S, i) = n_i$ or there exists $j \in N_i$ such that $\{a_j, b_j\} \subseteq S$. Given a tour set S , the corresponding tour γ can be found in $O(n)$.

(iii) Given any integer $y \in \mathbb{Z}_+^k$ with $y_i > 0$ for all i and satisfying conditions (a) and (b) above, define for each $i \mathcal{F}(y, i)$ as

$$\{S : S \subseteq X_i \cup Y_i, |S_i| = n_i, |S_i \cap Y_i| = y_i, \text{ condition (c) is satisfied}\}.$$

It is easy to find for every i a set $S^{y,i} \in \mathcal{F}(y, i)$ such that

$$\sum \{u : u \in S^{y,i}\} = \min\{\sum \{u : u \in S\} : S \in \mathcal{F}(y, i)\} = c_i(y_i).$$

If we define $S^y = \cup_{i=1}^k S^{y,i}$, then S^y is a tour set and

$$\begin{aligned} & \min\{\sum \{u : u \in S\} : y(S, i) = y_i \forall i\} \\ &= \sum \{u : u \in S^y\} = \sum_{i=1}^k c_i(y_i) = c(y). \end{aligned}$$

(iv) Each function $c_i(\cdot)$ can be extended to a convex function $f_i(\cdot)$ such that for any integer $1 \leq y_i \leq n_i$, $c_i(y_i) = f_i(y_i)$. The problem is thus equivalent to the integer program:

$$\min\{\sum_{i=1}^k f_i(y_i) : 2y_i \leq \sum_{i=1}^k y_i, 1 \leq y_i \leq n_i, y_i \text{ integer } \forall i\}.$$

In [806], an $O(n \log n)$ algorithm is presented for this integer program.

10.3. The Burkov-Rubinshtein Case

We call a cost matrix C with finite non-diagonal coefficients, a *Burkov-Rubinshtein matrix (BR-matrix)* if and only if for any distinct $u, v \in N$, $\min\{c_{uv}, c_{vu}\} = \min\{c_{ij} : i \neq j\}$. In [151] a polynomial scheme is given for $TSP(C)$ when C is a BR-matrix. Theorem 91 below is crucial for this algorithm.

A digraph such that its underlying undirected graph is a complete graph is called a *semicomplete digraph* [84]. A digraph is said to be *strongly connected* if and only if for any pair of distinct nodes $i, j \in N$, there exist a path from i to j and a path from j to i .

Theorem 91 [156] *A semicomplete digraph $G = (N, E)$ has a tour if and only if it is .*

Proof. If G has a tour then it is obviously strongly connected. To prove the converse, suppose G is strongly connected. The following

$O(n^3)$ scheme will produce a tour in G . Obviously, G contains some subtour. Choose any subtour $(i_1, i_2, \dots, i_k, i_{k+1}(=i_1))$ in G and let N^0 be its node set. If $N^0 = N$, then we have a tour in G . Else, let $N^1 = N - N^0$. If there exist $u \in N^1$ and $v \in \{1, \dots, k\}$ such that $\{(i_v, u), (u, i_{v+1})\} \subset E$, then replace arc (i_v, i_{v+1}) in the subtour by arcs (i_v, u) and (u, i_{v+1}) to get a larger subtour $(i_1, i_2, \dots, i_v, u, i_{v+1}, \dots, i_{k+1}(=i_1))$ and repeat the process. Else, there exists a partition $\{N^{1,1}, N^{1,2}\}$ of the node set N^1 such that for all $u \in N^{1,1}$, $v \in N^{1,2}$ and $j \in \{1, \dots, k\}$, $\{(i_j, u), (v, i_j)\} \subset E$. Since G is strongly connected, there exist $u \in N^{1,1}$ and $v \in N^{1,2}$ such that $(u, v) \in E$. Replace the arc (i_1, i_2) in the subtour by arcs $\{(i_1, u), (u, v), (v, i_2)\}$ to get a larger subtour and repeat the process. This proves the theorem. ■

An $O(n^2)$ scheme for producing a tour in a strongly connected semi-complete digraph is given in [579].

Consider an instance $TSP(C)$ where C is a BR-matrix. Let $G = (N, E)$ be the complete digraph on N . Let $\min\{c_{ij} : i \neq j\} = a$ $E^0 = \{(i, j) : i \neq j, c_{ij} = a\}$ and let $G^0 = (N, E^0)$. Then, the digraph G^0 is semicomplete. If G^0 is strongly connected, then a tour in G^0 can be obtained in $O(n^2)$ time and it is obviously an optimal solution to $TSP(C)$. Else, it is observed in [151] that for any minimal set $S \subseteq E - E^0$ such that $G^S = (N, E^0 \cup S)$ is strongly connected, every tour in G^S must contain all the arcs in S and hence, if we denote $\sum_{(i,j) \in S} (c_{ij} - a)$ by $d[S]$, then the cost of every tour in G^S must be $na + \sum_{(i,j) \in S} (c_{ij} - a) = na + d[S]$. Thus, $TSP(C)$ reduces to finding arc set $S^* \subseteq E - E^0$ such that G^{S^*} is strongly connected and $d[S^*]$ is minimum. An $O(n^2)$ algorithm for this is given in [151] and it is based on the following idea.

Let $\{G_i^0 = (N^i, E_i^0) : i \in \{1, \dots, k\}\}$ be the strongly connected components of G^0 . (These can be obtained in $O(n^2)$ using depth-first search [218].) Then each G_i^0 is a semicomplete, strongly connected digraph and hence has a tour γ_i . For any $S \subseteq E - E^0$, let $H = (K, F)$, $H^0 = (K, F^0)$ and $H^S = (K, F^0 \cup S')$ be obtained from G , G^0 and G^S , respectively, by shrinking each node set N^i to a supernode i and deleting loops. (Thus, $K = \{1, 2, \dots, k\}$.) We can renumber the nodes in K such that $F^0 = \{(i, j) : 1 \leq i < j \leq k\}$. If S is a minimal set such that G^S is strongly connected, then in H^S , the arcs in S' lie on a path from node k to node 1. We associate with every arc (i, j) in F , an arc (u, v) in E such that $u \in N^i$, $v \in N^j$ and $(c_{uv} - a) = \min\{(c_{xy} - a) : x \in N^i, y \in N^j\}$, and we assign arc (i, j) cost $c_{uv} - a$. Then the cost of every arc in F^0 is 0 and the problem of finding S^* reduces to the problem of finding a minimum cost path from the node k to the node 1 in H . The arcs in E associated with the arcs in the

optimal path can be extended to a tour in G by adding arcs associated with some arcs in F^0 and the arcs in subtours $\{\gamma_i : i \in \{1, \dots, k\}\}$.

It may be noted that the algorithm for TSP on BR-matrices can be extended to any cost matrix for which the digraph corresponding to its minimum non-diagonal entries belongs to a class of digraphs satisfying the following two conditions. (i) Every strongly connected digraph in the class has a tour. (ii) Every node-induced directed subgraph of a digraph in the class also belongs to the class. Some such classes of digraphs are discussed in [84].

10.4. Step-Back pyramidal tours

Let γ be a tour on N . A peak i of γ is called a *step-back peak* [274] if and only if exactly one of the following two conditions holds. (i) $\gamma^{-1}(i) < i$, $\gamma(i) = i - 1$ and $\gamma(\gamma(i)) > i$. (ii) $\gamma^{-1}(\gamma^{-1}(i)) > i$, $\gamma^{-1}(i) = i - 1$ and $\gamma(i) < i$. We call a peak of γ that is not a step-back peak a *proper peak*. A tour γ is called a *step-back pyramidal tour* [274] if and only if the only proper peak of γ is node n . The following results are proved in [274].

Theorem 92 [274] *Let D be the density matrix of a cost matrix C with finite non-diagonal elements. Suppose D satisfies the following conditions. For all $1 \leq i < j < x \leq n - 1$,*

- (i) $M_{i,j,j,x}^D \geq 0$;
- (ii) $M_{j,x,i,j}^D \geq 0$;
- (iii) $M_{i,x,j,j}^D \geq 0$; and
- (iv) $M_{j,j,i,x}^D \geq 0$.

Then, there exists an optimal tour to $TSP(C)$ that is a step-back pyramidal tour.

Theorem 93 [274] *For any cost matrix with finite non-diagonal elements, a minimum cost step-back pyramidal tour can be obtained in $O(n^2)$ time.*

10.5. Other solvable cases

A cost matrix C is called a Brownian matrix [801] if and only if there are two vectors $a, b \in \mathbb{R}^n$ such that $c_{ij} = a_i$ if $i < j$ and $c_{ij} = b_j$ otherwise. In [801], an $O(n \log n)$ time algorithm is given for $TSP(C)$ when C is a Brownian matrix.

Glover and Punnen [382] consider TSP on the following undirected graph $G^{GP} = (N, E)$. For some $u \in N$, the node set $N - \{u\}$ is parti-

tioned into k disjoint sets $\{N_1, N_2, \dots, N_k\}$ where each set N_i contains at least 3 nodes. The edge set E consists of (i) edge sets E^i forming a subtour on node set N_i ; (ii) edge sets $E^{i,i+1}$ containing all the edges connecting nodes in N_i to nodes in N_{i+1} ; (iii) edge sets $E^{u,1}$ and $E^{u,k}$ connecting the node u to all the nodes in N_1 and N_k , respectively. They show that TSP on such a graph G^{GP} is NP-hard and provide the following result on solvable cases of TSP for special symmetric cost matrices on G^{GP} .

Theorem 94 [382] *Let C be a symmetric cost matrix compatible with $G^{GP} = (N, E)$. If C satisfies conditions (i)-(iii) or conditions (i) and (iv) below, then $TSP(G^{GP}, C)$ can be solved in $O(n^2)$ time.*

- (i) For all $i \in \{1, \dots, k-1\}$ and $(a, b) \in E^{i,i+1}$ $c_{ab} \geq \max\{c_{xy} : (x, y) \in E^i \cup E^{i+1}\}$.
- (ii) For all $(a, b) \in E^{u,1}$, $c_{ab} \geq \max\{c_{xy} : (x, y) \in E^1\}$.
- (iii) For all $(a, b) \in E^{u,k}$, $c_{ab} \geq \max\{c_{xy} : (x, y) \in E^k\}$.
- (iv) For all $(a, b) \in E^{u,k}$, $c_{ab} \geq n(\max\{c_{xy} : (x, y) \in E - E^{u,k}\})$

For arbitrary integers n, a_1, a_2, \dots, a_m with $0 < a_1 < \dots < a_m < n$, the *circulant digraph* generated by a_1, a_2, \dots, a_m is the digraph $G(n, a_1, a_2, \dots, a_m) = (N, E)$, where $E = \{(i, j) : i, j \in N; (j - i) = a_t \pmod{n} \text{ for some } t \in \{1, \dots, m\}\}$.

Theorem 95 [831] *Let $G = G(n, a_1, a_2)$ be a Hamiltonian circulant digraph. For some c_1, c_2 , let C be a cost matrix compatible with G such that $c_{ij} = c_u$ if $(j - i) = a_u \pmod{n}$ for $u \in \{1, 2\}$. Then $TSP(G, C)$ can be solved in $O(n)$ time.*

10.6. The master tour problem

For a cost matrix C with finite non-diagonal entries, a tour γ on N is called a *master tour* with respect to C [652] if and only if for any $N' \subseteq N$, the tour obtained from γ by removing from it nodes not in N' is an optimal solution to $TSP(C')$, where C' is the principal submatrix of C corresponding to row/column set N' . For example, convex-hull TSP has a master tour. The following result is proved in [244].

Theorem 96 [244] *For a symmetric cost matrix C , the tour $(1, 2, \dots, n, 1)$ is a master tour if and only if C is a Kalmanson matrix.*

Proof. Every principal submatrix of a Kalmanson matrix is a Kalmanson matrix. (See Section 6 for definition of a Kalmanson matrix.) The sufficiency part of the theorem therefore follows from Corollary 64.

To prove the converse, suppose $(1, 2, \dots, n, 1)$ is a master tour with respect to some symmetric cost matrix C . Then, by definition of master tour, for any $1 \leq u < v < w < x \leq n$, the tour (u, v, w, x, u) is an optimal solution to $TSP(C')$, where C' is the principal submatrix of C corresponding to row/column set $\{u, v, w, x\}$. But this implies that the edges (u, w) and (v, x) intersect. Hence, C is a Kalmanson matrix. This proves the theorem. ■

Characterization of non-symmetric cost matrices for which a master tour exists is an open problem.

10.7. Recognition of various solvable classes

Given an instance of TSP, an important related problem is to check if it belongs to one of the solvable cases of TSP. Results on polynomial time recognition schemes for various specially structures cost matrices for which the TSP is polynomially solvable can be found in [57, 145, 176, 193, 242, 243, 244].

Acknowledgement: Special thanks to K.G. Murty and Abraham Punnen for carefully reading the manuscript and suggesting significant improvements; to Lushu Li and Bambang Pramujati for help in preparing the manuscript and to Dr. Maria Storożczuk for support. This work was partially supported by a research grant from NSERC (Canada).

Chapter 12

THE MAXIMUM TSP

Alexander Barvinok *

Department of Mathematics

University of Michigan

Ann Arbor, MI 48109-1109, USA

barvinok@math.lsa.umich.edu

Edward Kh. Gimadi †

Sobolev Institute of Mathematics

Novosibirsk, Russia

gimadi@math.nsc.ru

Anatoliy I. Serdyukov ‡

Sobolev Institute of Mathematics

Novosibirsk, Russia

1. Introduction

The Maximum Traveling Salesman Problem (MAX TSP), also known informally as the “taxicab ripoff problem”, is stated as follows:

Given an $n \times n$ real matrix $c = (c_{ij})$, called a *weight matrix*, find a Hamiltonian cycle $i_1 \mapsto i_2 \mapsto \dots \mapsto i_n \mapsto i_1$, for which the maximum value of $c_{i_1 i_2} + c_{i_2 i_3} + \dots + c_{i_{n-1} i_n} + c_{i_n i_1}$ is attained. Here (i_1, \dots, i_n) is a permutation of the set $\{1, \dots, n\}$.

Of course, in this general setting, the Maximum Traveling Salesman Problem is equivalent to the Minimum Traveling Salesman Problem, since the maximum weight Hamiltonian cycle with the weight matrix c corresponds to the minimum weight Hamiltonian cycle with the weight

*Partially supported by NSF Grant DMS 9734138

†Partially supported by RFFI grant 99-01-00601

‡Partially supported by RFFI grant 99-01-00601

matrix $-c$. What makes the MAX TSP special is that there are some interesting and natural special cases of weights c_{ij} , not preserved by the sign reversal, where much more can be said about the problem than in the general case. Besides, methods developed for the Maximum Traveling Salesman Problem appear useful for other combinatorial and geometric problems.

In most cases, in particular in problems P1–P3 and P5–P6 below, we assume that the weight matrix is *non-negative*: $c_{ij} \geq 0$ for $i, j = 1, \dots, n$. This is the main condition that breaks the symmetry, since the corresponding condition of non-positive weights for the Minimum Traveling Salesman Problem is rarely, if ever, imposed and considered unnatural. It also makes the MAX TSP somewhat easier than the MIN TSP. We will discuss the following special cases.

- P1. The symmetric problem.** We have $c_{ij} = c_{ji}$ for all i, j .
- P2. The semimetric problem.** We have $c_{ij} + c_{jk} \geq c_{ik}$ for all triples (i, j, k) .
- P3. The metric problem.** We have $c_{ij} = c_{ji}$ for all pairs (i, j) and $c_{ij} + c_{jk} \geq c_{ik}$ for all triples (i, j, k) .
- P4. The problem with warehouses.** In this case, the matrix c has some special structure. We are given $r \times n$ matrices $u = (u_{ij})$ and $v = (v_{ij})$ such that

$$c_{ij} = \max_{k=1, \dots, r} (u_{ki} + v_{kj}) \quad \text{for all } i, j = 1, \dots, n. \quad (1)$$

The number r is assumed to be small (fixed) and n is allowed to vary. The smallest r for which representation (1) exists is called in [86] the *combinatorial rank* of c . The problem has the following interpretation [88]: suppose that together with n cities, there are r *warehouses*. Before visiting each city, the Traveling Salesman has to pick up goods at any of the r warehouses. The cost of going from the i -th city to the k -th warehouse is u_{ki} and the cost of going from the k -th warehouse to the j -th city is v_{kj} . If the Traveling Salesman wants to maximize the cost of the tour visiting each city only once (with no restrictions regarding visiting warehouses imposed), he should use the cost matrix c determined by the above equations. We do not assume c_{ij} to be non-negative and the analysis of the MAX TSP in this case mirrors that of the MIN TSP with “max” replaced by “min” throughout.

- P5. The problem in a normed space.** Let us fix a norm $\|\cdot\|$ in Euclidean space \mathbb{R}^d . More precisely, we require $\|x\|$ to be a non-negative real number for any $x \in \mathbb{R}^d$, that $\|\lambda x\| = \lambda \|x\|$ for all

$x \in \mathbb{R}^d$ and all $\lambda \geq 0$ and that $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^d$. We do not require that $\|-x\| = \|x\|$. It is convenient, although not necessary, to assume that $\|x\| = 0$ implies $x = 0$. Although the symmetry condition may be violated, we call $\|\cdot\|$ a norm anyway, hoping that this will not lead to a confusion. In this case, the weight matrix c has the following structure: we are given n points p_1, \dots, p_n in \mathbb{R}^d such that $c_{ij} = \|p_j - p_i\|$ for all i and j .

P6. The problem in a polyhedral norm. This is a particular case of P5 with the additional assumption that the unit ball $\mathbf{B} = \{x \in \mathbb{R}^d : \|x\| \leq 1\}$ is a polyhedron, that is the intersection of finitely many halfspaces.

Some relations between P1–P6 are seen instantly: it is immediate that P3 is the intersection of P1 and P2, that P6 is a particular case of P5 and that P5 is a particular case of P2. It is also clear that P5 can be viewed as a “limit” of P6. Moreover, for any particular instance of MAX TSP with n points in a general norm, there is an equivalent instance in a polyhedral norm with the unit ball \mathbf{B} having $O(n^2)$ facets (the facets of \mathbf{B} account for the directions $p_i - p_j$, where p_1, \dots, p_n are given points). This idea is from [285], where it is attributed to J. Mitchell. Although not very difficult, it may seem a little surprising that P6 is a particular case of P4, with the warehouses in P4 corresponding to the facets of \mathbf{B} .

In this chapter, we describe what is known (to us) about special cases P1–P6. We also describe some results on probabilistic analysis of the problem when weights c_{ij} are sampled at random from some distribution.

Before we proceed, we discuss an interesting application of the MAX TSP.

1.1. An application: the shortest superstring problem

Given strings s_1, \dots, s_n over some alphabet, find the shortest string s (superstring) containing each s_i as a substring. This problem, known as the shortest superstring problem has applications to DNA sequencing and data compression (see [515]).

In [114], the following reduction to the MAX TSP was suggested. Let c_{ij} be the length of the overlap of s_i and s_j , that is the length of the longest string v such that $s_i = uv$ and $s_j = vw$ for some strings u and w . It turns out that an approximate solution to the MAX TSP with the weight matrix c gives rise to an approximate solution to the shortest superstring problem. More precisely, let us fix a real number ρ in the interval $[1/2, 1]$. Given a Hamiltonian cycle whose weight approximates

the maximum weight of a Hamiltonian cycle within a factor of ρ , one can produce a superstring s whose length approximates the minimum length of a superstring within a factor of $(4 - 2\rho)$.

2. Hardness Results

In [659] it is shown that the MIN TSP is MAX SNP-hard even when restricted to matrices with weights $c_{ij} \in \{1, 2\}$. MAX SNP-hardness means that unless $P=NP$, there is a constant $\epsilon > 0$ for which there is no polynomial time algorithm approximating the minimum weight of a Hamiltonian cycle up to within relative error ϵ . It follows that P1–P3 are MAX SNP-hard. In P4, as long as the number r of warehouses is allowed to vary, the problem is as hard as the general TSP since any $n \times n$ matrix c_{ij} can be written as

$$c_{ij} = \max_{k=1, \dots, n} (u_{ki} + v_{kj}),$$

where $v_{ij} = c_{ij}$, $u_{ii} = 0$ and $u_{ij} = -\infty$ for $i \neq j$.

However, we see in Section 8 that the problem becomes polynomially solvable when r is fixed. We also note that it is an NP-hard problem to find the combinatorial rank of a given matrix c although it is possible to check in polynomial time whether the combinatorial rank does not exceed 2 and find the corresponding representation (1), see [174].

Fekete in [285] shows that P5 with $\|x\| = \sqrt{\xi_1^2 + \dots + \xi_d^2}$ for $x = (\xi_1, \dots, \xi_d)$ being the standard Euclidean norm is NP-hard provided $d \geq 3$ (the idea is to reduce the problem of finding a Hamiltonian cycle in a grid graph to the MAX TSP in 3-dimensional Euclidean space by using a clever drawing of the graph on the unit sphere). As we noted in Section 1, any instance of the MAX TSP with n points in a normed space can be reduced to an instance of the MAX TSP in a polyhedral norm for which the unit ball \mathbf{B} has $O(n^2)$ facets. Using this observation, Fekete further shows that P6 is an NP-hard problem provided the number of facets of the unit ball \mathbf{B} is allowed to vary. As we discuss in Section 8, the problem is polynomially solvable if the number of facets of \mathbf{B} is fixed. The status of the MAX TSP in \mathbb{R}^2 with the standard Euclidean norm is not known. Note that in the case of Euclidean norm there is a standard difficulty of comparing lengths (which are sums of square roots of rational numbers), so it is not clear whether the decision version of the MAX TSP (given a rational number ρ , is there a Hamiltonian cycle whose length exceeds ρ) is in NP.

3. Preliminaries: Factors and Matchings

If G is an undirected graph, one can view a Hamiltonian cycle in G as a *connected* subset $E' \subseteq E$ of edges such that every vertex $v \in V$ is incident to exactly two edges in E' . If we do not insist that E' is connected, we come to the notion of a 2-factor.

We recall that a set $E' \subseteq E$ of edges of an undirected graph $G = (V, E)$ is called a 2-matching if every vertex of G is incident to at most two edges from E' . A set $E' \subseteq E$ of edges of an undirected graph $G = (V, E)$ is called a 2-factor if every vertex of G is incident to exactly two edges from E' (see Appendix A). More generally, suppose that to every vertex v of G a non-negative integer $f(v)$ is assigned. A set $E' \subseteq E$ is called an f -factor if every vertex v of G is incident to exactly $f(v)$ edges from E' . If $f(v) \in \{0, 1\}$ for every v , an f -factor is called a matching and if $f(v) = 1$ for all v , the matching is called perfect (see Appendix A).

As is known, (see, for example, Chapter 9 of [570] and Appendix A), a maximum weight (perfect) matching in a given weighted graph can be constructed in $O(|V|^3)$ time. Finding a maximum weight f -factor in a given weighted graph G can be reduced to finding a maximum weight perfect matching in some associated graph G'' (see Chapter 10 of [570]) with $|V''| = O(|E|)$ and can be solved in polynomial time as well. Various methods of finding an approximate solution to the MAX TSP (MIN TSP) are based on finding a 2-factor of the maximum (minimum) weight and then transforming it to a Hamiltonian cycle.

Using Edmonds' technique of "blossoms", Hartvigsen in [435] obtained an $O(n^3)$ algorithm to construct a maximum weight 2-matching in a given weighted graph with n vertices. A maximum weight 2-factor can be found in $O(n^3)$ time as well by Hartvigsen's algorithm after some initial conditioning of the weights. All weights on the edges of the graph should be increased by a large constant so that every maximum weight 2-matching is necessarily a 2-factor.

One can observe that a 2-factor is just a union of vertex-disjoint cycles S_1, \dots, S_l such that the union $S_1 \cup \dots \cup S_l$ contains every vertex of the graph. As we try to approximate an optimal Hamiltonian cycle by an optimal 2-factor, it seems natural to try to search for an optimal 2-factor satisfying some additional restrictions that would make it somewhat closer to a Hamiltonian cycle.

A 2-factor is called k -restricted if none of the cycles S_1, \dots, S_l contains k or fewer vertices. Hence a 2-factor is 2-restricted and a Hamiltonian cycle is k -restricted for any $k < |V|$.

The problem of finding a maximum weight 4-restricted 2-factor is NP-hard [812], whereas the status of the problem for a maximum weight

3-restricted 2-factor is not known. However, there is a polynomial time algorithm for finding an (unweighted) 3-restricted 2-factor in a given graph [435] and a 4-restricted 2-factor in a given bipartite graph, if there is any [436]. In some cases, see, for example, [659], knowing how to find some k -restricted 2-factor allows one to get a better solution to the weighted problem. The problem of deciding if there is a 5-restricted 2-factor is NP-complete (this result belongs to Papadimitriou, see [223] and [436]) and the status of the problem for 4-restricted 2-factors in general (not necessarily bipartite) graphs is not known.

Now we briefly describe the situation of a directed graph.

Let $G = (V, A)$ be a directed graph. We recall (see Appendix A) that collection of vertex disjoint cycles S_1, \dots, S_l is called 1-factor (or, sometimes, *cycle cover*) if every vertex of G belongs to one of the cycles.

Given a weighted directed graph with n vertices, one can find the 1-factor of the maximum (minimum) weight in $O(n^3)$ time (see, for example, Chapter 11 of [656] and Appendix A).

Recall that a partial tour is a set of edges (arcs) of an undirected (directed) graph which can be appended by including some other edges (arcs) of the graph to a Hamiltonian cycle in the graph (see Appendix A). As we remarked earlier, in almost all cases we assume that the weights on the edges are non-negative. A frequently used approach to the Maximum TSP is to construct a partial tour of a sufficiently large weight and then append it to a Hamiltonian cycle. When the weights are non-negative, adding new edges may only increase the total weight.

4. MAX TSP with General Non-Negative Weights

In this section, we assume that the weight matrix is non-negative but otherwise does not have any special properties. In other words, we are looking for a maximum weight Hamiltonian cycle in a complete weighted directed graph $G = (V, A)$ with $|V| = n$ vertices and $|A| = n(n - 1)$ arcs (note, that the arcs (i, j) and (j, i) are different). The following notion will be used throughout the paper.

Definition 1 *Suppose that we have an algorithm for a class of problems with non-negative weights. We say that the algorithm has a performance ratio ρ (where ρ is a positive real number) if it produces a Hamiltonian cycle whose weight is at least ρ times the maximum weight of a Hamiltonian cycle.*

For general non-negative weights the following performance ratios can be obtained in polynomial time: $1/2$ [308], $4/7$ [521] and $38/63$ [515], the best currently known.

The rest of this section describes the idea of the algorithm from [515].

First, a maximum weight 1-factor F is constructed in $O(n^3)$ time. Then three different ways to patch the cycles into a Hamiltonian cycle are considered. Cycles with 2 vertices, also called *2-cycles*, play a special role.

The first way consists of deleting a minimum weight arc from each of the cycles of F and extending the obtained paths into a tour H_1 (this is the idea of the algorithm in [308]).

In the second way, a weighted undirected graph G' is produced from the given complete digraph G and a Hamiltonian cycle H_2 is constructed using the maximum weight matching in G' (cf. also Section 5).

In the third way, cycles of F with many vertices are broken into pieces and an auxiliary directed graph G'' is constructed by contracting the pieces. A Hamiltonian cycle H_3 is obtained by using a maximum weight matching in G'' . Let W be the weight of F , let bW be the total weight of the heavier arcs in the 2-cycles of F and let cW be the weight of the lighter arcs in the 2-cycles of F .

It turns out that H_1 provides a $(2/3 + (b - 2c)/3)$ performance guarantee, H_2 provides a $(7/12 - (b - 2c)/12)$ performance guarantee and H_3 provides a $(2/3 + 4(b - 2c)/15)$ performance guarantee. It follows then that the best of H_1, H_2 and H_3 provides a $38/63$ performance guarantee.

Although the original paper uses all three cycles H_1, H_2 and H_3 , it appears that H_2 and H_3 alone are sufficient to obtain a $38/63$ performance guarantee.

In the next three sections, we are going to discuss approximation algorithms for problems P1-P3.

5. The Symmetric MAX TSP

We consider the MAX TSP with non-negative weights c_{ij} subject to the symmetry condition of P1. In other words, we are looking for a maximum weight Hamiltonian cycle in a weighted complete undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = \binom{n}{2}$ edges.

In [308] a polynomial time algorithm with the performance ratio of $2/3$ was suggested. The algorithm finds the maximum weight 2-factor and modifies it to a Hamiltonian cycle, cf. Section 3. In [520] a better performance ratio of $13/18$ was achieved and in [438] a polynomial time algorithm with the performance ratio $5/7$ was constructed. Below we sketch an algorithm due to Serdyukov [751], which achieves the performance ratio of $3/4$ and has $O(n^3)$ complexity.

5.1. Sketch of the Algorithm from [751]

First, we describe the algorithm when the number n of vertices is even. This is also the simplest case to analyze.

Part I, n is even. Let us construct a maximum weight 2-factor F and a maximum weight perfect matching M (cf. Section 3). The 2-factor F is a union $F = S_1 \cup \dots \cup S_l$ of vertex disjoint cycles, each of which contains at least 3 edges of the graph. The idea is to construct two partial tours T_1 and T_2 , choose the one of the largest weight and arbitrarily extend it to a Hamiltonian cycle. We start with $T_1 = M$ and $T_2 = \emptyset$. Now we process cycles S_1, \dots, S_l one by one, so that precisely one edge e_i of each cycle S_i is assigned to T_1 and the remaining edges of S_i are assigned to T_2 . Clearly, T_2 will always be a partial tour.

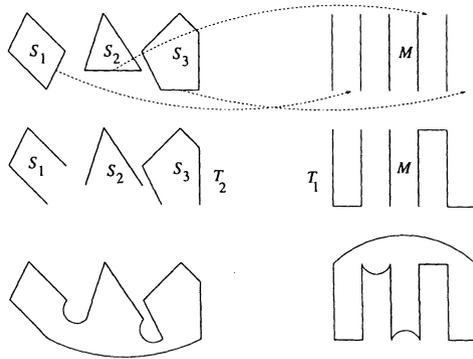


Figure 12.1. Constructing a Hamiltonian cycle when n is even

Now, we observe that we can choose e_i in such a way that T_1 remains a partial tour. Indeed, we can choose e_1 arbitrarily. Let us assume that $e_1, \dots, e_{i-1}, i \geq 2$ are chosen, hence T_1 is a union of vertex-disjoint paths. Let us pick two adjacent candidate edges e'_i and e''_i of S_i . The common vertex of e'_i and e''_i is an end-vertex of a path in T_1 , hence we choose $e_i \in \{e'_i, e''_i\}$ so that e_i is not incident to the other end-vertex of that path.

When all cycles S_1, \dots, S_l are processed, we choose T to be the one of T_1 and T_2 with the larger weight $c(T)$ (either of the two if $c(T_1) = c(T_2)$). Finally, we extend T to a Hamiltonian cycle.

We observe that after the cycle S_i has been processed, the weight $c(T_1) + c(T_2)$ increases by $c(S_i)$ and hence in the end we have $c(T_1) + c(T_2) = c(M) + c(F)$. Thus $c(T) \geq \frac{c(F) + c(M)}{2}$ and the 3/4 per-

formance grantee follows from the observation that $c(F) \geq c(H)$ and $c(M) \geq c(H)/2$ for every Hamiltonian cycle H .

The case of an odd number n of vertices requires some adjustments and its relies on some of the arguments and constructions in Part I.

Part II, n is odd. This time, there is no perfect matching M but one can find a maximum weight almost 1-factor, which covers all but one vertex v of the graph and which we also call M . As above, we construct a maximum weight 2-factor $F = S_1 \cup \dots \cup S_l$. Let us assume that v is a vertex of S_1 and let e be an edge which is incident to v , is not an edge of F and has the maximum weight among all such edges. Depending on whether the other end-vertex of e belongs to S_1 or to some other cycle S_2 , we construct a partial tour T_2 by either removing two non-adjacent edges of S_1 adjacent to e or by removing an edge adjacent to e in S_1 and another edge adjacent to e in S_2 and patching S_1 and S_2 via e .

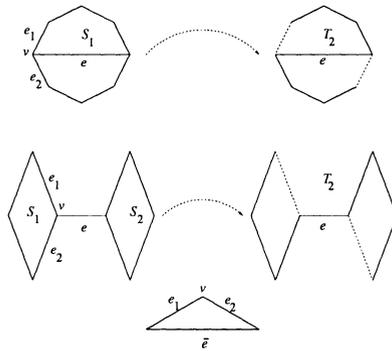


Figure 12.2. Constructing a partial tour T_2 when n is odd

The two removed edges are added to M thus forming a partial tour T_1 . Next, we process the remaining cycles of F (that is, the cycles S_i for $i > 1$ in the first case and the cycles S_i for $i > 2$ in the second case) as in Part I and produce a partial tour \tilde{T} . The weight of \tilde{T} is compared to the weight of a partial tour \bar{T} in a modified weighted graph \bar{G} on $n - 1$ vertices constructed as follows. Let e_1 and e_2 be the edges of S_1 that are incident to v . We remove vertex v and all edges incident to it from G and preserve the weights on all the remaining edges of G , except for a single edge \bar{e} for which e_1, e_2 and e form a triangle. The weight of \bar{e} is modified: $c(\bar{e}) := c(e_1) + c(e_2)$. Since the new graph \bar{G} has an even number of vertices, we construct a partial tour \bar{T} as in Part 1. If \bar{T} happens to contain \bar{e} , we modify it by removing \bar{e} and inserting e_1 and

e_2 instead. Hence we obtain a partial tour \bar{T} in G . From the partial tours \tilde{T} and \bar{T} we choose the one of the largest weight and extend it to a Hamiltonian cycle H .

It turns out that $c(H) \geq \frac{3}{4}c(H^*)$, where H^* is a maximum weight Hamiltonian cycle in G . Suppose that H^* contains both e_1 and e_2 . Replacing e_1 and e_2 by \bar{e} , we obtain a Hamiltonian cycle in \bar{G} and hence by Part 1 we have $c(H) \geq c(\bar{T}) \geq \frac{3}{4}c(H^*)$. Suppose that H^* contains at most one edge from the set $\{e_1, e_2\}$ and let e^* be an edge of the minimum weight in H^* . Since H^* contains at least one edge incident to v and not in the cycle S_1 , we must have $c(e) \geq c(e^*)$. Then the required estimate follows from the inequalities

$$c(F) \geq c(H^*) \quad \text{and} \quad c(M) \geq \frac{c(H^*) - c(e^*)}{2}$$

and

$$c(H) \geq c(\tilde{T}) \geq \frac{c(F) + c(M) + c(e)}{2}.$$

The latter inequality is obtained as in Part I.

Hassin and Rubinstein in [439] combined the ideas of their earlier paper [438] with those of Serdyukov to obtain a randomized algorithm, which, for any fixed $\rho < 25/33$, achieves an expected performance ratio of at least ρ in $O(n^3)$ time.

5.2. The Idea of the Algorithm from [439]

The algorithm constructs three “candidate” Hamiltonian cycles H_1 , H_2 and H_3 and chooses the one with the largest weight. As in Section 5.1, a maximum weight 2-factor $F = S_1 \cup \dots \cup S_l$ is constructed. Given a $\rho < 25/33$, a number $\epsilon > 0$ is computed (ϵ approaches 0 when ρ approaches $25/33$) and the cycles S_1, \dots, S_l are assigned to be “long” or “short”, depending on whether the number of vertices in the cycle is greater or less than ϵ^{-1} . Then, for each short cycle, a maximum weight Hamiltonian path on its vertices is constructed (using dynamic programming, one can do it in $O(n^2 2^{1/\epsilon})$ time). From each long cycle a minimum weight edge is excluded. All paths obtained (from both short and long cycles) are extended into a Hamiltonian cycle H_3 .

The cycles H_1 and H_2 are obtained by extending partial tours T_1 and T_2 , which are constructed in a somewhat similar way as in Section 5.1. As in Section 5.1, a maximum weight perfect matching M is constructed. Furthermore, a maximum weight perfect matching M' is constructed on the edges with the endpoints in different cycles S_1, \dots, S_l . Then, from

each cycle S_i roughly half of the edges are assigned to T_1 (randomization is involved in choosing which half) and the rest is assigned to T_2 . After that, edges of M' are used to patch T_2 (randomization is used here as well).

The gain in the performance ratio compared to the “pure” algorithm from Section 5.1 comes from the observation that if the optimal Hamiltonian cycle follows cycles of F closely, then H_3 alone provides a good approximation. If, however, the optimal cycle does not follow F closely, it has to use sufficiently many edges from M' . The algorithm is randomized and it is proved that the expected weight of the produced cycle is at least ρ times the maximum weight of a Hamiltonian cycle.

One can show that for any prescribed probability $p < 1$ of success, after running the algorithm independently $O(n/(1-p))$ times, one obtains the desired cycle with the probability at least p .

We note also that the complexity bound includes an $O(n^{22^{1/\epsilon}})$ term which is dominated by $O(n^3)$ for any fixed $\epsilon > 0$ but quickly takes over if ϵ is allowed to approach 0, that is, if we want the performance ratio to approach 25/33.

6. The Semimetric MAX TSP

In this section, we discuss Problem P2. Note, that we do not assume the symmetry condition of P1. Again, it is convenient to restate the problem of as the problem of finding a maximum weight Hamiltonian cycle in a (this time directed) weighted complete graph $G = (V, A)$ with $|V| = n$ vertices and $|A| = n(n - 1)$ arcs.

The following result is crucial.

Theorem 2 *Let $G = (V, A)$ be a directed graph with $|V| = n$ vertices and non-negative weights $c(a)$ on its arcs. Let $F = S_1 \cup \dots \cup S_l$ be a 1-factor of G . Let m_i be the number of vertices in S_i , so $m_1 + \dots + m_l = n$. Let $m = \min\{m_i : 1 \leq i \leq l\}$. Then there exist Hamiltonian cycles H_1 and H_2 in G whose weights satisfy the inequalities*

$$c(H_1) \geq \left(1 - \frac{1}{n}\right)^{l-1} c(F) \tag{2}$$

and

$$c(H_2) \geq \left(1 - \frac{1}{2m}\right) c(F). \tag{3}$$

Moreover, given a 1-factor F of G , Hamiltonian cycles H_1 and H_2 can be constructed in $O(mn)$ time.

Inequality (2) was obtained in [754] and inequality (3) was obtained in [517].

Sketch of Proof. To construct H_1 , we repeatedly patch two cycles of the factor into one, thus reducing the number of cycles by 1 until we reach a Hamiltonian cycle. In doing so, we arrange current cycles S_i in a non-decreasing order of the weight per vertex ratios $c(S_i)/m_i$ and always patch the first two cycles, say S_1 and S_2 .

Let us fix an arc of S_2 , say $(1, 2)$. We delete arc $(1, 2)$ of S_2 and a certain arc (x, y) of S_1 and then add arcs $(1, y)$ and $(x, 2)$, thus obtaining a new cycle $S_1 * S_2$.

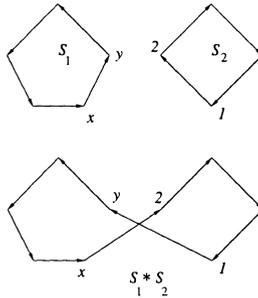


Figure 12.3. Patching cycles

Using the triangle inequality $c(1, 2) \leq c(1, z) + c(z, 2)$ for every vertex z of S_1 , one can deduce that the average weight $c(S_1 * S_2)$ over all possible choices of arcs (x, y) of S_1 is at least as big as $c(S_1) + c(S_2) - c(S_1)/m_1$. Hence if we choose (x, y) so as to maximize the weight of $c(S_1 * S_2)$, we get

$$c(S_1 * S_2) \geq c(S_1) + c(S_2) - c(S_1)/m_1,$$

from which it follows that the weight of the 1-factor multiplies by at least $(1 - 1/n)$ after each patch (we recall that S_1 has the lowest weight per vertex ratio, so $c(S_1)/m_1 \leq c(F)/n$).

To construct H_2 , we find two families P_1, \dots, P_m and Q_1, \dots, Q_m of Hamiltonian cycles such that

$$\sum_{i=1}^m c(P_i) + \sum_{i=1}^m c(Q_i) \geq (2m - 1) \sum_{i=1}^l c(S_i). \tag{4}$$

Then H_2 is a maximum weight cycle from the collections P_1, \dots, P_m and Q_1, \dots, Q_m . Each cycle P_i or Q_j uses all but one arc from every cycle S_i and some additional arcs bundling the cycles together. Let us choose a

cyclical order S_1, \dots, S_l, S_1 on the cycles S_i and a cyclical order on the vertices of each cycle S_i compatible with the direction of the cycles. To construct P_j , from each cycle S_i we remove the arc coming into the j -th vertex and connect the $(j - 1)$ -st vertex of the next cycle with the j -th vertex of S_i .

To construct Q_j , we delete the arc coming into the j -th vertex of S_1 and then for each next cycle we remove the arc coming into the vertex whose index drops by 1 each time we pass to a new cycle. New arcs connecting each cycle with the next one are added to make a Hamiltonian cycle.

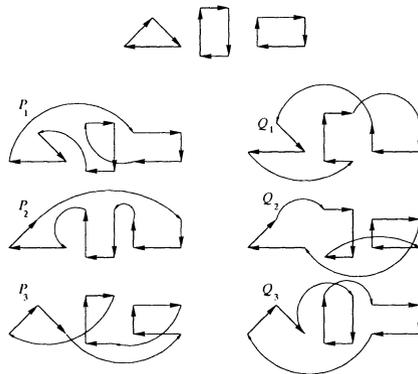


Figure 12.4. Constructing P_i and Q_j

Inequality (4) is obtained by the repeated application of the triangle inequality.

Since we can construct a maximum weight 1-factor with $m \geq 2$ in $O(n^3)$ time, by (3) we get a polynomial time algorithm with the performance ratio $3/4$ [517]. The construction of H_1 and inequality (2) will be later used in Section 9.

7. The Metric MAX TSP

When both the triangle inequality of P2 and the symmetry condition of P1 are satisfied, the performance ratio can be improved further. Indeed, constructing a maximum weight 2-factor in $O(n^3)$ time, (see Appendix A), we have $m \geq 3$ in inequality (3) of Theorem 2. Hence we get a polynomial time algorithm with the performance ratio of $5/6$ [517]. The same performance ratio is obtained in [521]. As in (3), the performance ratio of the Kovalev and Kotov algorithm is expressed in terms of the minimum number m of vertices in a cycle of the underlying

maximum weight 2-factor. It turns out to be equal to $(m+2)/(m+3)$, which agrees with (3) for $m=3$ but is somewhat weaker for $m>3$.

As this survey was nearing completion, we learned that recently Hasin and Rubinstein in [440] designed a randomized algorithm for the metric MAX TSP with $O(n^3)$ complexity and a $7/8$ expected performance ratio.

8. TSP with Warehouses

In this section, we discuss Problem P4. Hence we assume that we are given $r \times n$ matrices u and v with the weight matrix c represented in the form (1). As we already mentioned in Section 1, by switching “max” to “min” we get corresponding results for the MIN TSP and we do not need to assume that c is non-negative.

Our main result is a polynomial time algorithm from [90] for the case when r is fixed. Note that in [90], the matrix c is assumed to be symmetric, and the corresponding problem is referred to as the *Tunneling* TSP, as warehouses are replaced by tunnels connecting cities (we also note that if the Traveling Salesman travels by air, the role of the warehouses is naturally played by airline hubs). The symmetry condition is not crucial (the earlier version of the paper treats the general case) although it leads to some improvement of the complexity estimates. Here we discuss the general, not necessarily symmetric case.

8.1. Sketch of the Algorithm from [90]

Let us consider a graph $G = (V, E)$ with $n+r$ nodes. The set V of nodes is the union $V = C \cup W$ of $n = |C|$ nodes, called *cities* and $r = |W|$ nodes called *warehouses*. Each city is connected to each warehouse by two arcs, one, from the city to the warehouse, colored red and the other, from the warehouse to the city, colored blue. The weight of the red arc connecting the i -th city and the k -th warehouse is u_{ki} whereas the weight of the blue arc is v_{ki} . We are looking for a closed walk in G of the maximum total weight, which visits every city exactly once. The problem reduces to finding a maximum weight subset $E' \subseteq E$ which satisfies the following three conditions:

(8.1.1) every city is incident to exactly two arcs in E' , one red and one blue;

(8.1.2) for every warehouse, the number of incident red edges in E' is equal to the number of incident blue edges in E' ;

(8.1.3) the subgraph of G induced by E' is connected.

Then an Euler tour through E' will produce the desired solution. Since the set E' must be connected, all visited warehouses must be connected in E' . The idea is to enumerate all possible *minimal ways* to connect warehouses, that is, we enumerate all minimal connected subgraphs of G containing a subset of W . Since such a connected subgraph is a tree and contains at most $2r - 2$ edges, the direct enumeration of all possible minimal connected subgraphs $F \subseteq E$ takes $O(n^{2r-2})$ time, which is polynomial in n if r is fixed. Hence we enumerate all possible connected subgraphs F and for each such an F , we find a subset $E'_F \subseteq E$ of the maximum weight which contains $E(F)$ and satisfies (8.1.1)–(8.1.2). The problem of finding E'_F reduces to solving of $O(n^r)$ maximum weight f -factor problems, since the only conditions we have to satisfy on the edges of $E'_F \setminus E(F)$ are the degree constraints on the cities and warehouses enforced by (8.1.1)–(8.1.2), cf. Section 3. Finally, we choose E' of the maximum weight among all E'_F .

The resulting complexity of the algorithm is $O(n^{3r+1})$, some ways to improve it are discussed in [90] and [87].

The following simple observation turns out to be quite useful (we use it in Section 9 comparing two different approaches to the MAX TSP in a space with a polyhedral norm).

Lemma 3 *Suppose that the combinatorial rank of the weight matrix c is r . Then there exists a maximum weight 1-factor F of the complete directed graph on the vertex set containing not more than r cycles. Moreover, if c is given in the form (1) the 1-factor F can be constructed in $O(n^3)$ time.*

Sketch of Proof. We have

$$c_{ij} = \max_{k=1, \dots, r} (u_{ki} + v_{kj})$$

for some $r \times n$ matrices u and v . For each pair (i, j) of cities let us identify a warehouse $k = k(i, j)$ such that $c_{ij} = u_{ki} + v_{kj}$.

Suppose that a maximum weight 1-factor contains more than r cycles. Then there will be two arcs (s, t) and (p, q) of different cycles that correspond to the same warehouse k . Let us patch the cycles by deleting (s, t) and (p, q) and inserting (s, q) and (p, t) . The weight of the factor can only increase since

$$c_{sq} + c_{pt} \geq u_{ks} + v_{kq} + u_{kp} + v_{kt} = c_{st} + c_{pq}. \tag{5}$$

Some other special classes of matrices c for which the MAX TSP is polynomially solvable (in particular, via the “pyramidal tour” approach) are surveyed in Chapter 11 and [142]. In [112] it is shown that if c is an $n \times n$ non-negative matrix and $c_{ij} = 0$ for all i, j with $|i - j| \neq 1$, then a maximum weight tour can be found in $O(n)$ time.

9. MAX TSP in a Space with a Polyhedral Norm

Let \mathbb{R}^d be Euclidean space endowed with the standard scalar product $\langle \cdot, \cdot \rangle$. Let us fix some non-zero vectors $a_1, \dots, a_r \in \mathbb{R}^d$ and let

$$\mathbf{B} = \left\{ x \in \mathbb{R}^d : \langle a_i, x \rangle \leq 1 : i = 1, \dots, r \right\}. \tag{6}$$

Hence \mathbf{B} is a full-dimensional polyhedron, which contains the origin in its interior. We do not assume that \mathbf{B} is symmetric about the origin but it is convenient, although not necessary, to assume that \mathbf{B} is bounded. Given \mathbf{B} , let us define the Minkowski functional $\| \cdot \|$ by

$$\|x\| = \min\{\lambda \geq 0 : x \in \lambda\mathbf{B}\}.$$

Hence $\| \cdot \|$ defines a distance function $d(x, y) = \|y - x\|$.

Given n points $p_1, \dots, p_n \in \mathbb{R}^d$, let us consider their distance matrix $c = (c_{ij})$, $c_{ij} = \|p_j - p_i\|$. One can observe that the combinatorial rank of c (see Section 1) is bounded above by the number r of inequalities defining \mathbf{B} in (6). Moreover, one can represent c in the form (1) provided the vectors a_1, \dots, a_r in (6) are given. Indeed,

$$\begin{aligned} c_{ij} &= \|p_j - p_i\| = \min\{\lambda \geq 0 : p_j - p_i \in \lambda\mathbf{B}\} \\ &= \min\{\lambda \geq 0 : \langle a_k, p_j - p_i \rangle \leq \lambda : k = 1, \dots, r\} \\ &= \max\{\langle a_k, p_j - p_i \rangle : k = 1, \dots, r\} = \max_{k=1, \dots, r} (u_{ki} + v_{kj}), \end{aligned}$$

where $u_{ki} = -\langle a_k, p_i \rangle$ and $v_{kj} = \langle a_k, p_j \rangle$.

Geometrically, the combinatorial rank of the matrix of pairwise distances between points in a polyhedral norm does not exceed the number of facets of the unit ball. This observation is from [86] and, as mentioned there, is joint with S. Onn and A. Gerards. In the context of Section 8, facets of the unit ball \mathbf{B} correspond to warehouses.

Note that, if \mathbf{B} is unbounded, we have to write

$$c_{ij} = \max\{0, \langle a_k, p_j - p_i \rangle : k = 1, \dots, r\}$$

and the combinatorial rank of c may increase by 1.

Using results of Section 8, we get a polynomial time algorithm for solving the MAX TSP problem in polyhedral norm, provided the number of facets r of the unit ball is fixed [90]. When the unit ball \mathbf{B} is symmetric about the origin and hence $\|\cdot\|$ is a genuine norm, the complexity $O(n^{r-2} \ln n)$ can be achieved, see [90] and [87]. Particularly interesting cases are those of the L^1 norm in the plane \mathbb{R}^2 (the so called “Manhattan norm” alluding to the other name of the MAX TSP, the “taxicab ripoff” problem) and the L^∞ norm in the plane, for both of which Fekete in [285] obtained an algorithm of linear complexity $O(n)$.

A different approach was suggested by Serdyukov in [755]. The algorithm in [755] is based on the following observation.

Lemma 4 *Let $\mathbf{B} \subset \mathbb{R}^d$ be a bounded polyhedron, given by (6) and let $\|\cdot\|$ be the corresponding Minkowski functional. For points $p_i, p_j \in \mathbb{R}^d$, let $c_{ij} = \|p_j - p_i\|$. Then*

$$c_{12} + c_{23} \geq c_{13} \tag{7}$$

for any three points p_1, p_2 and p_3 .

Suppose that for $k = 1, \dots, r$, the set $G_k = \{x \in \mathbf{B} : \langle a_k, x \rangle = 1\}$ is a facet of \mathbf{B} and let Γ_k be the cone with the vertex at the origin spanned by G_k . Suppose further, that p_1, p_2 and p_3, p_4 are points such that for some k , one has $p_2 - p_1 \in \Gamma_k$ and $p_4 - p_3 \in \Gamma_k$. Then

$$c_{12} + c_{34} \leq c_{14} + c_{32}. \tag{8}$$

Indeed, (7) is standard and follows from the convexity of \mathbf{B} . Inequality (8) is, essentially, inequality (5) of Section 8, since if $p_j - p_i \in \Gamma_k$ then $\|p_j - p_i\| = \langle a_k, p_j - p_i \rangle$ and the arc (p_i, p_j) corresponds to the k -th warehouse.

9.1. Sketch of the Algorithm from [755]

Given points $p_1, \dots, p_n \in \mathbb{R}^d$ and the weight matrix $c_{ij} = \|p_j - p_i\|$, we construct a maximum weight 1-factor $F = S_1 \cup \dots \cup S_l$. The idea is to show that the number l of cycles can be reduced to at most $\lfloor r/2 \rfloor$, where r is the number of facets of \mathbf{B} .

Indeed, suppose that $l > \lfloor r/2 \rfloor$. Then there are two arcs (x_1, x_2) and (y_1, y_2) which belong to different cycles of F and such that $x_2 - x_1, y_2 - y_1 \in \Gamma_k$ for some cone Γ_k spanned by a facet of \mathbf{B} (note that the cones Γ_k cover the whole space \mathbb{R}^d and that each cycle has arcs from at least two cones Γ_k). Now we patch the cycles by deleting (x_1, x_2) and (y_1, y_2) and inserting (x_1, y_2) and (y_1, x_2) . Lemma 4 implies that the weight of the 1-factor can not become smaller.

Repeating this procedure, we obtain a maximum weight 1-factor with at most $\lfloor r/2 \rfloor$ cycles. Up to this point, the construction goes as in Lemma 3 of Section 8, only cutting the required number of cycles by half because of a special geometric feature of the problem.

Now we construct a Hamiltonian cycles H_1 as in Theorem 2, Section 6 (note that by inequality (7), the weight function satisfies the triangle inequality).

Clearly, the complexity of the algorithm is $O(n^3)$, dominated by the complexity of constructing a maximum weight 1-factor. As follows by (2), Section 6, the weight of the constructed Hamiltonian cycle is at least $(1 - 1/n)^{\lfloor r/2 \rfloor - 1}$ times the maximum weight of a Hamiltonian cycle.

It is interesting to compare algorithms of [90] and [755]. While algorithm from [755] does not solve the problem exactly except in the case of the norm in the plane \mathbb{R}^2 whose unit ball is a triangle, its relative error is asymptotically 0 for $n \rightarrow +\infty$ as long as $r = o(n)$ and its complexity is linear in the number r of facets of the unit ball \mathbf{B} . On the other hand, the algorithm from [90] gives an exact solution, but its complexity is exponential in the number of facets of \mathbf{B} .

10. MAX TSP in a Normed Space

Given a norm in \mathbb{R}^d , we can always approximate it by a polyhedral norm within an arbitrarily small relative error. Hence each of the two approaches described in the previous section leads to a fully polynomial approximation scheme for the MAX TSP in a finite-dimensional vector space with a fixed norm, see [86].

Let us consider the case of the Euclidean norm $\|x\| = \sqrt{\xi_1^2 + \dots + \xi_d^2}$ for $x = (\xi_1, \dots, \xi_d)$ in some more detail. Assuming that d is fixed, to approximate the norm $\|\cdot\|$ by a polyhedral norm within relative error ϵ , we need to approximate the unit ball \mathbf{B} in \mathbb{R}^d by a polyhedron \mathbf{P} such that

$$\mathbf{P} \subset \mathbf{B} \subset (1 + \epsilon)\mathbf{P}.$$

Such a polyhedron \mathbf{P} will have $O(\epsilon^{-d})$ facets, so the complexity of the algorithm from [90] will be $n^{O(\epsilon^{-d})}$. However, since the complexity of algorithm of Section 9.1 depends linearly on the number of facets of \mathbf{P} , by Serdyukov's approach, we get an algorithm of $O(n^3 + \epsilon^{-d})$ complexity and relative error $\epsilon + O(\epsilon^{-d}/n)$. Thus for any fixed $\epsilon > 0$ the algorithm of [755] achieves asymptotically the relative error ϵ with the complexity $O(n^3)$ as the number n of points grows to infinity.

Using a different approach, in the case of Euclidean norm, Serdyukov in [753] proposed a method for which the relative error is asymptotically 0, as n grows. The approach is based on the following geometric result.

Lemma 5 *Let $\{I_1, \dots, I_l\}$ be a set of l straight line intervals in \mathbb{R}^d . Then the smallest angle between a pair of intervals from $\{I_1, \dots, I_l\}$ is bounded from above by a constant $\alpha(d, l)$ such that $\lim_{l \rightarrow +\infty} \alpha(d, l) = 0$. Moreover,*

$$\cos \alpha(d, l) \geq 1 - \gamma(d)l^{\frac{-2}{d-1}}$$

for some constant $\gamma(d)$ independent on the number of intervals.

Intuitively, if we have many intervals in a space of low dimension, some of them will be nearly parallel. We denote by $\alpha(I_i, I_j)$ the angle between I_i and I_j .

The proof of Lemma 5 can be obtained as follows. Without loss of generality, we may assume that the intervals I_1, \dots, I_l are diameters of the unit sphere $S^{d-1} \subset \mathbb{R}^d$ (that is, each interval is symmetric about the origin and has length 2). Representing a diameter by its endpoints on the sphere, we observe that for every interval I , the set of all diameters J of S^{d-1} such that $\alpha(I, J) \leq \beta$ is represented by a set $C(I, \beta)$, which is a pair of antipodal spherical caps centered at the endpoints of I and of radius β in the intrinsic (geodesic) metric of the sphere. Using well-known formulas for spherical volumes, we estimate the smallest β for which some two sets $C(I_i, \beta)$ and $C(I_j, \beta)$ are bound to intersect. Then $\alpha(I_i, I_j) \leq 2\beta$.

10.1. Sketch of the Algorithm from [753]

Step 1. Given n points p_1, \dots, p_n in \mathbb{R}^d with the weight matrix $c_{ij} = \|p_j - p_i\|$, we construct a maximum weight matching. Hence we get a set M of $m = \lfloor n/2 \rfloor$ straight line intervals in \mathbb{R}^d .

Step 2. Let us choose a number $l \leq m/2$ (to be specified later). Let us choose some l intervals in M of the smallest weight and call them *light*. All other intervals in M we call *heavy*. Let I_1, \dots, I_m be an ordering of the intervals from M . We call a subsequence $R = (I_i, I_{i+1}, \dots, I_{i+k})$, $k \geq 0$, a *run* if R is a maximal subsequence with the properties that all intervals in R are heavy and the angle between each pair I_j, I_{j+1} of consecutive intervals in R does not exceed $\alpha(d, l)$, cf. Lemma 5. In particular, a subsequence consisting of a single heavy interval may constitute a run. Our goal is to construct an ordering I_1, \dots, I_m with not more $l - 1$ runs. To achieve that, we start with an arbitrary ordering I_1, \dots, I_m and identify all runs. We observe that if the number of runs exceeds $l - 1$, by Lemma 5, we can find a run R_1 with the rightmost interval I_i and a run

R_2 with the rightmost interval I_j such that $\alpha(I_i, I_j) \leq \alpha(d, l)$. Then, by a proper reordering of the intervals, we merge R_1 and R_2 into a larger run R . After repeating this procedure not more than, say, m times, we get an ordering with not more than $l - 1$ runs.

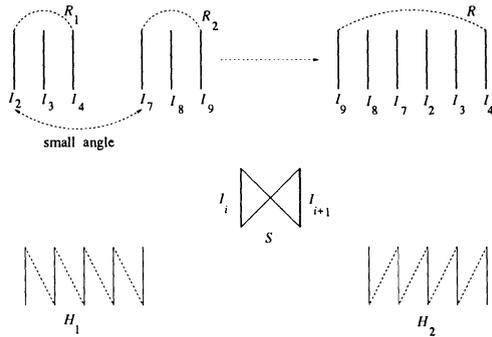


Figure 12.5. Patching runs into Hamiltonian cycles

Step 3. For every pair of consecutive intervals I_i and I_{i+1} , we find a pair of edges patching them into a maximum weight 4-cycle S . Let E be the set of all newly constructed edges. Using edges from E we consecutively patch the intervals I_1, \dots, I_m into a Hamiltonian cycle H . The idea is to choose H to be the best of the two main types of patching H_1 and H_2 , see Figure 4. More precisely, in [753], Serdyukov constructs four cycles, two having the H_1 type and two having the H_2 type, to take care of “boundary effects”, and chooses the best of the four.

The algorithm achieves $O(n^3)$ complexity, again dominated by the complexity of constructing a maximum weight matching. It turns out that if l is chosen to be $\lfloor n^{\frac{d-1}{d+1}} \rfloor$, then the algorithm achieves $1 - \beta(d)n^{\frac{-2}{d+1}}$ performance ratio, where $\beta(d)$ is a constant depending on the dimension alone. We observe that as long as d is fixed, the performance ratio approaches 1 as the number n of points increases.

The analysis is based on the following observations. The total weight of heavy intervals is at least $c(M)(1 - l/m)$, that is, asymptotically equal to the weight $c(M)$ of the matching M . Hence it suffices to analyze how does weight $c(H)$ relate to the total weight of heavy intervals in M . If the intervals I_i and I_{i+1} had been parallel, we would have had $c(S) \geq 2c(I_1) + 2c(I_2)$ for the optimal patching S of I_1 and I_2 into a 4-cycle. If I_i and I_{i+1} are nearly parallel, $c(S)$ is guaranteed to be almost twice the weight $c(I_1) + c(I_2)$ and an exact bound can be obtained in

terms of the angle $\alpha(I_1, I_j)$. Let us consider the ordering I_1, \dots, I_m obtained after Step 2 is performed. Let $R = (I_i, I_{i+1}, \dots, I_k)$ be a run and let P_i be the path obtained from R in $H_i, i = 1, 2$. It follows that the average $\frac{c(P_1) + c(P_2)}{2}$ is guaranteed to be almost twice the total weight of the intervals in R . Since the number of runs is small, “boundary effects” are asymptotically negligible and we conclude that the average weight of the H_1 and H_2 patchings is almost as large as $2c(M)$. Since for any Hamiltonian cycle H^* we have $2c(M) \geq c(H^*)$ (for n even) and $2c(M) \geq c(H^*)(1 - 1/n)$ (for n odd), we conclude that H should be asymptotically optimal.

Exploiting geometric duality between minimum stars and maximum matchings in [286] and [287] Fekete et al. obtained a practical near-linear heuristic in the case of Euclidean norm in \mathbb{R}^2 that has a worst-case guarantee of $2/\sqrt{3} \approx 1.15$. It is interesting to note [784] that the standard linear programming relaxation of a maximum matching problem for an even number of points has always integral optima in the case of a norm in \mathbb{R}^2 and hence can be solved by a network flow algorithm, whereas the general maximum matching algorithm of $O(n^3)$ complexity becomes impractical for large n .

11. Probabilistic Analysis of Heuristics

In this section, we assume that instances of the MAX TSP are sampled at random from some probability space. We discuss how some obvious heuristics, “farthest neighbor” (FN) and “subtour patching” (SP), typically behave. Such heuristics produce a Hamiltonian cycle, whose weight often approximates the maximum weight of a Hamiltonian cycle reasonably well.

11.1. Asymptotic Optimality

Let \mathcal{K}_n be a probability space of MAX TSP instances with n cities. We say that an algorithm (heuristic) has an (ϵ_n, δ_n) performance guarantee on problems from \mathcal{K}_n if the probability that the weight of the produced Hamiltonian cycle approximates the maximum weight within relative error ϵ_n is at least $1 - \delta_n$. If $\mathcal{K}_n, n = 1, 2, \dots$, is a series of probability spaces of MAX TSP instances of increasing sizes, we say that the algorithm is asymptotically optimal if one can choose performance guarantees (ϵ_n, δ_n) is such a way that $\epsilon_n \rightarrow 0$ and $\delta_n \rightarrow 0$ as $n \rightarrow +\infty$.

11.2. The Farthest Neighbor Heuristic

Given a weight matrix $c = (c_{ij})$, we start with any vertex, go to the farthest, then to the farthest remaining and so on. The following result is obtained in [362] and [364].

Theorem 6 *Suppose that the weights c_{ij} are sampled independently at random from some distribution in the interval $[a_n, b_n]$ with $b_n > a_n > 0$. Let $P(x) = \mathbf{P}\{\xi < x\}$ be the distribution function of a random variable $\xi = (c_{ij} - a_n)/(b_n - a_n)$.*

Let

$$\psi(n) = \int_0^{1-1/n} \frac{dx}{1 - P(x)}.$$

Suppose that at least one of the following holds:

- 1) We have $\psi(n) \rightarrow +\infty$ and $\psi(n)/n \rightarrow 0$ as $n \rightarrow +\infty$;
- 2) We have $P(x) > x$ for all x and $\psi(n) = o(n)$;
- 3) We have $P(x) \leq x$ for all x .

Then FN is asymptotically optimal.

It follows that FN is asymptotically optimal for any convex distribution and for the uniform distribution without any additional assumptions. This is in contrast to the "Nearest Neighbor" heuristic for the MIN TSP, see [365] and [366] and Chapter 6.

The proof of Theorem 6 goes along the following lines: we estimate the expected weight of the constructed Hamiltonian cycle, use Chebyshev's inequality to show that a typical weight is sufficiently close to the expectation and use a trivial upper bound nb_n on the largest weight of a Hamiltonian cycle (it turns out that the expected weight is sufficiently close to nb_n).

In the case of the uniform distribution $P(x) = x$, Vohra in [811] proved that for any $\delta_n > 0$ one can choose $\epsilon_n = O(\sqrt{n^{-1} \ln(1/\delta_n)})$ so that (ϵ_n, δ_n) is a performance guarantee for the heuristic. One can get a better bound $\epsilon_n = O(n^{-1} \ln(1/\delta_n))$ by using large deviation inequalities. Letting $\delta_n = n^{-\lambda/2}$, where λ is any positive constant, we have

$$\epsilon_n = \frac{\lambda \ln n}{n}, \quad \delta_n = n^{-\lambda/2},$$

which implies asymptotically optimality of the farthest city heuristic. Moreover, this result is valid for a wider class of distributions where $P(x) \leq x$.

11.3. The Subtour Patching Heuristic

Given a complete directed graph on n vertices, suppose that the arc weights c_{ij} are chosen independently at random from some distribution.

The subtour patching heuristics starts with constructing a maximum weight 1-factor and then patches the subtours into a tour. The procedure turns out to be quite efficient under some mild assumption about the distribution. Essentially, we do not even need the weights c_{ij} to be independent, the following two conditions will be sufficient:

(11.3.1) The resulting distribution on the optimal 1-factors is uniform;

(11.3.2) Individual weights c_{ij} are not very large compared to sums of several weights.

It is well known (see, for example, Sections 7–9 of [510]), a random permutation of n elements with probability very close to 1 will have $O(\ln n)$ cycles. Therefore, (11.3.1) implies that the resulting maximum weight subtour cover with the overwhelming probability will have $O(\ln n)$ subtours. The condition (11.3.2) implies that the patching procedure does not “skew” the weight of the tour too much.

For example, in [363] it was shown that if (in the directed case) the columns of the distance matrix form a sequence of *symmetrically dependent* random variables (we recall that random variables X_1, \dots, X_m are called symmetrically dependent if the distribution of the vector $(X_{\sigma(1)}, \dots, X_{\sigma(m)})$ does not depend on the permutation σ of the set $\{1, \dots, m\}$), then the MAX TSP can be solved in $O(n^3)$ time with the performance estimates

$$\epsilon_n = 2c^* \ln n / F_{AP}^*, \quad \delta_n = (e/n)^{0.38},$$

where c^* is the maximum entry of the distance matrix and F_{AP}^* is the optimal value of the objective function in the maximum assignment problem. Hence for $0 < a_n \leq c_{ij} \leq b_n$ with $b_n/a_n = o(n/\ln n)$ the subtour patching algorithm is asymptotically optimal.

Results similar to the ones described in Sections 11.2 and 11.3 can be obtained for discrete distributions, e.g. for FN see [364]. We note also that Dyer, Frieze and McDiarmid [263] presented an asymptotically optimal linear-time partitioning heuristic when the points are chosen uniformly in the unit square. They also computed the expected value of the longest Hamiltonian cycle through n random points which turned out to be approximately equal to $0.7652n$.

Chapter 13

THE GENERALIZED TRAVELING SALESMAN AND ORIENTEERING PROBLEMS

Matteo Fischetti

D.E.I., University of Padova

Via Gradenigo 6/A, 35100 Padova, Italy

fisch@dei.unipd.it

Juan-José Salazar-González

D.E.I.O.C., University of La Laguna

38271 La Laguna, Tenerife, Spain

jjsalaza@ull.es

Paolo Toth

D.E.I.S., University of Bologna

Viale Risorgimento 2, 40136 Bologna, Italy

ptoth@deis.unibo.it

1. Introduction

Routing and Scheduling problems often require the determination of optimal sequences subject to a given set of constraints. The best known problem of this type is the classical *Traveling Salesman Problem* (TSP), calling for a minimum cost Hamiltonian cycle on a given graph.

In several applications the cycle is allowed to visit only a subset of the nodes of the graph, chosen according to a specified criterion. A basic version of this problem is the following *Simple Cycle Problem* (SCP). We are given a complete undirected graph $K_n = (V, E)$ on $n := |V|$ nodes, a cost c_e associated with each edge $e \in E$, and a prize p_v associated with each node $v \in V$. Recall that a (simple) cycle of K_n is a subset \tilde{E} of E , $|\tilde{E}| \geq 3$, inducing a subgraph $(V(\tilde{E}), \tilde{E})$ which is connected and in

which all nodes in $V(\tilde{E})$ have degree two. The *cost* of a cycle \tilde{E} is given by $\sum_{e \in \tilde{E}} c_e - \sum_{v \in V(\tilde{E})} p_v$. The problem is to find a min-cost cycle of K_n .

Without loss of generality one can assume $c_e \geq 0$ for all $e \in E$ and $p_v \geq 0$ for all $v \in V$, since the addition of any constant to all edge costs and to all node prizes does not affect the cycle cost.

SCP is a useful model for problems involving simultaneous selection and sequencing decisions. Indeed, the problem involves two related decisions:

- 1 choosing a convenient node subset $S \subseteq V$,
- 2 finding a minimum cost Hamiltonian cycle in the subgraph induced by S .

Many variants of SCP have been studied in the literature, a non-exhaustive list of which is given later in this section. Roughly speaking, we can classify these variants into two main classes, the first including all variants subsuming the TSP (i.e., those for which every Hamiltonian cycle is feasible), and the second including all the variants in which additional constraints may prevent some Hamiltonian cycles from being feasible. This property has important consequences when analyzing the structure of the polytope P associated with a certain SCP variant. Indeed, whenever P contains the TSP polytope, one can apply simple lifting constructions to extend known TSP facets to P , whereas more involved constructions are required for the problems in the second class. We will therefore describe in a rather detailed way a specific variant for each of the two classes, namely the Generalized TSP (Section 2) and the Orienteering Problem (Section 3). These two problems have been chosen as they seem to be the most-widely studied cycle-type problems, along with the Prize-Collecting TSP considered in Chapter 14. In particular, for both problems we will concentrate on exact solution methods based on the branch-and-cut approach, which proved to be the most effective framework for optimally solving cycle-type problems.

1.1. The Simple Cycle Problem

The Simple Cycle Problem, SCP, has the following natural Integer Linear Programming formulation. For any $S \subseteq V$, let $\delta(S)$ represent the set of edges with exactly one endnode in S , and let $E(S)$ be the set of edges with both endnodes in S , i.e.,

$$\begin{aligned}\delta(S) &:= \{(i, j) \in E : i \in S, j \notin S\}, \\ E(S) &:= \{(i, j) \in E : i, j \in S\}.\end{aligned}$$

As it is customary, we write $\delta(v)$ instead of $\delta(\{v\})$ for $v \in V$. Moreover, for any real function $f : Q \rightarrow \mathbb{R}$ on a finite domain Q and for any $T \subseteq Q$, we write $f(T)$ instead of $\sum_{q \in T} f_q$.

Our model introduces a binary variable x_e associated with each edge $e \in E$ (where $x_e = 1$ if and only if e belongs to the optimal cycle), and a binary variable y_v associated with each node $v \in V$ (where $y_v = 1$ if and only if v is visited by the optimal cycle). The model reads:

$$v(SCP) \equiv \min \sum_{e \in E} c_e x_e - \sum_{v \in V} p_v y_v \tag{1}$$

subject to:

$$x(\delta(v)) = 2y_v \quad v \in V \tag{2}$$

$$x(\delta(S)) \geq 2(y_i + y_j - 1) \quad S \subset V, i \in S, j \in V \setminus S \tag{3}$$

$$y(V) \geq 3 \tag{4}$$

$$x_e \in \{0, 1\} \quad e \in E \tag{5}$$

$$y_v \in \{0, 1\} \quad v \in V. \tag{6}$$

Constraints (2) impose that the number of edges incident to a node v is either 2 (if v is visited) or 0 (otherwise). Inequalities (3) are connectivity constraints saying that each cut separating two visited nodes (i and j) must be crossed at least twice. Constraint (4) forces at least three nodes to be visited by the cycle.

A variant of SCP requires the cycle visits a specified “depot” node, say node 1. This can be easily obtained by adding a large positive value to prize p_1 , or by introducing explicitly the additional constraint:

$$y_1 = 1. \tag{7}$$

In this case, the connectivity constraints (3) can be replaced by

$$x(\delta(S)) \geq 2y_v \quad \text{for } S \subset V, 1 \in S, v \in V \setminus S.$$

SCP is known to be strongly \mathcal{NP} -hard, as it contains as a special case the problem of finding a Hamiltonian cycle of a given undirected arbitrary graph $G = (V, \bar{E})$ (just set $c_e = 0$ for all $e \in \bar{E}$, $c_e = 1$ for all $e \in E \setminus \bar{E}$, $p_v = 1$ for all $v \in V$, and check whether the min-cost cycle on K_n has cost equal to $-n$).

1.2. The Weighted Girth Problem

This problem arises from SCP when $p_v = 0$ for all $v \in V$, and negative edge costs are allowed. Notice that, because of (2), one can always

convert node prizes into edge costs, by redefining $c_{(i,j)} := c_{(i,j)} - (p_i + p_j)/2$ for all $(i, j) \in E$, and $p_v = 0$ for all $v \in V$. As a consequence, the Weighted Girth Problem (WGP) is equivalent to SCP, hence it is strongly \mathcal{NP} -hard.

A relevant polynomially-solvable special case arises when the graph does not contain negative cycles (negative costs being allowed). Indeed, in this case the problem can be transformed into $|V|$ non-bipartite matching problems on G with loops (see, e.g., Coullard and Pulleyblank [227]), hence it can be solved in $O(|V|^4)$ time. A simpler algorithm can be obtained along the following lines.

We replace each edge $e = (i, j)$ of G by two arcs (i, j) and (j, i) with cost $c_{ij} := c_{ji} := c_e$ (we set $c_{ij} := \infty$ for all missing arcs, including loops). Observe that this construction induces a negative circuit of length 2 for each negative-cost edge, but no negative-cost circuit involving more than 2 arcs. Therefore, WGP can be restated as the problem of finding a minimum-cost (possibly nonsimple) circuit (closed trail) on the new digraph, with the constraint that the circuit contains no 2-length circuit. As such, it can be solved by dynamic programming using the recursions originally proposed by Christofides, Mingozzi and Toth [192] to derive the so-called q -path lower bound for the Vehicle Routing Problem, as outlined next.

Let us consider the case where a certain node r (e.g., $r = 1$) is assumed to be visited by the circuit (the most general case can be solved by trying all possible nodes r), and let $\gamma(j) := \{i \in V : (i, j) \in E\}$. For each node $j \in V$ and for each integer $h = 1, \dots, n$, let $f(h, j)$ represent the minimum cost of a directed path (not necessarily simple) with h arcs that starts from node r , arrives at node j , and contains no 2-length circuit. Moreover, let $\pi(h, j)$ denote the node immediately preceding node j in the path corresponding to $f(h, j)$, and let $g(h, j)$ be the minimum cost of a path having the same properties as the one corresponding to $f(h, j)$, but with the node immediately preceding j forced to be different from $\pi(h, j)$. Initially, for each $j \in V$ we set $f(1, j) := c_{rj}$, $\pi(1, j) := r$, and $g(1, j) := \infty$. Then, for $h = 2, \dots, n$ and for each $j \in V$ we compute:

$$f(h, j) := \min_{i \in \gamma(j)} \begin{cases} f(h-1, i) + c_{ij} & \text{if } \pi(h-1, i) \neq j \\ g(h-1, i) + c_{ij} & \text{otherwise,} \end{cases}$$

(let $\pi(h, j)$ be the node corresponding to the minimum above)

$$g(h, j) := \min_{i \in \gamma(j) \setminus \{\pi(h, j)\}} \begin{cases} f(h-1, i) + c_{ij} & \text{if } \pi(h-1, i) \neq j \\ g(h-1, i) + c_{ij} & \text{otherwise.} \end{cases}$$

The optimal solution value of WGP can then be computed as:

$$v(\text{WGP}) := \min\{f(h, r) : h = 3, \dots, n\}.$$

The above computation can be clearly performed in $O(|E|n)$ time, hence the method requires $O(n^3)$ time in the worst case.

WGP is a basic relaxation of several problems with important practical applications in routing and location, and has been studied mainly from a theoretical point of view.

The polyhedral structure of the weighted girth problem has been deeply investigated by Bauer [92], who studied several classes of facet-defining inequalities, some of which derived from the TSP polytope. Lifting theorems relating the TSP and the WGP polytopes are given in Salazar [736]. Balas and Oosten [79] investigated the corresponding polytope for a directed graph; see Chapter 14 for details.

1.3. The Prize-Collecting TSP

In the Prize-Collecting TSP each node $v \in V$ has an associated non-negative *weight* w_v , and a cycle \tilde{E} is considered to be feasible only if the total weight $w(V(\tilde{E}))$ of the visited nodes is not less than a given threshold w_0 . Therefore, the problem can be formulated as (1)–(6), plus the additional constraint:

$$\sum_{v \in V} w_v y_v \geq w_0. \quad (8)$$

Such an \mathcal{NP} -hard problem arises, for instance, when a factory located at node 1 needs a given amount w_0 of a product, which can be provided by a set of suppliers located at nodes $2, \dots, n$. Let w_v be the indivisible amount supplied at node v , $-p_v$ be the corresponding cost ($v = 2, \dots, n$), and let $c_{(i,j)}$ be the transportation cost from node i to node j ($i, j \in V, i \neq j$). Assuming that only one trip is required, such a problem can be formulated as an instance of the Prize-Collecting TSP—in the version requiring the visit of node 1.

The directed counterpart of the problem also arises in several scheduling problems. Balas and Martin [77] introduced the Prize-Collecting TSP as a model for scheduling the daily operations of a steel rolling mill. A rolling mill produces steel sheets from slabs by hot or cold rolling. The cost of arc (i, j) is given by the cost of processing order j just after order i , and w_v is the weight of the slab associated with order v . Scheduling the daily operations consists of selecting a subset of orders whose total weight satisfies a given lower bound w_0 , and of sequencing them so as to minimize the global cost.

The Prize-Collecting TSP has been mainly investigated in its directed version. Heuristic methods have been proposed by Balas and Martin [77]. Balas [63, 65] analyzed the problem from a polyhedral point of view. Fischetti and Toth [302] proposed a branch-and-bound exact algorithm based on additive bounding procedures. Bienstock, Goemans, Simchi-Levi and Williamson [109] presented an approximation algorithm with constant bound. The reader is referred to Chapter 14 of this book for a comprehensive treatment of the subject.

1.4. The Capacitated Prize-Collecting TSP

The *Capacitated Prize-Collecting TSP* is an extension of the Prize-Collecting TSP where (8) is replaced by

$$\sum_{v \in V} w_v y_v \leq w_0. \quad (9)$$

Here w_v represents the weight of node (customer) v , and w_0 is the capacity of a vehicle originally located at the depot (say node 1). Constraints (9) specify that the total load carried by the vehicle cannot exceed the vehicle capacity. This \mathcal{NP} -hard problem was introduced by Bixby, Coullard and Simchi-Levi [110] as a column generation subproblem in a set partitioning formulation of the classical Capacitated Vehicle Routing Problem (CVRP). They also presented a branch-and-cut algorithm and solved to optimality instances ranging in size from 50 to 280 nodes.

1.5. The Orienteering Problem

The *Orienteering Problem* (OP), also called the “Selective TSP”, is in a sense the “dual” of the Prize-Collecting TSP. Here, the cycle cost only depends on the node prizes, i.e., $c_e = 0$ for all $e \in E$, and the objective is to maximize the global prize of the visited nodes. On the other hand, each edge $e \in E$ has an associated nonnegative *duration* t_e , and a cycle \tilde{E} is considered to be feasible only if its total duration $t(\tilde{E})$ does not exceed a given threshold t_0 . Moreover, the cycle is required to visit node 1. Model (1)–(7) then needs to be amended by the additional constraint:

$$\sum_{e \in E} t_e x_e \leq t_0 \quad (10)$$

OP is strongly \mathcal{NP} -hard as it contains as a special case the problem of finding a Hamiltonian cycle of a given undirected arbitrary graph $G = (V, \bar{E})$ (just set $p_v = 1$ for all $v \in V$, $t_e = 0$ for all $e \in \bar{E}$, $t_e = 1$ for

all $e \in E \setminus \bar{E}$, and $t_0 = 0$, and check whether the optimal solution has value n).

The problem derives its name from the Orienteering sport, where each participant has to maximize the total prize to be collected, while returning to the starting point within a given time limit. OP also arises in several routing and scheduling applications, see, e.g., Golden, Levy and Vohra [387].

Heuristic algorithms for OP and some generalizations have been proposed by Tsiligirides [796], Golden, Levy and Vohra [387], Golden, Wang and Liu [389], Chao, Golden and Wasil [178] and Fink, Schneiderei and Voss [290]. Exact branch-and-bound methods have been proposed by Laporte and Martello [536], and by Ramesh, Yoon and Karwan [693]. Leifer and Rosenwein [553] have discussed an LP-based bounding procedure. Recently, Fischetti, Salazar and Toth [301] and Gendreau, Laporte and Semet [355] have proposed branch-and-cut algorithms; see Section 3 for more details.

1.6. The Generalized TSP

In the *Generalized TSP* (GTSP), also known as the “International TSP”, we are given a proper partition of V into $m \geq 3$ clusters

$$C_1, C_2, \dots, C_m,$$

and the cycle is feasible only if it visits each cluster at least once. Typically we also have $p_v = 0$ for all $v \in V$. The corresponding additional constraints for model (1)–(6) are:

$$\sum_{v \in C_h} y_v \geq 1 \quad \text{for } h = 1, \dots, m \quad (11)$$

A different version of the problem, called *E-GTSP* (where E stands for Equality), arises when imposing that exactly one node of each cluster must be visited, i.e., (11) is replaced by:

$$\sum_{v \in C_h} y_v = 1 \quad \text{for } h = 1, \dots, m \quad (12)$$

The two versions are clearly equivalent when the costs satisfy the triangle inequality, i.e., $c_{(i,j)} \leq c_{(i,k)} + c_{(k,j)}$ for all node triples (i, j, k) .

Both GTSP and E-GTSP find practical applications in the design of ring networks, sequencing of computer files, routing of welfare customers through governmental agencies, airport selection and routing for courier planes, flexible manufacturing scheduling, and postal routing; see, e.g., Noon [629], Noon and Bean [630], and Laporte, Asef-Vaziri and Sriskandarajah [535].

The two problems are clearly \mathcal{NP} -hard, as they reduce to the TSP when $m = n$, i.e., $|C_h| = 1$ for all h . They have been studied, among others, by Laporte and Nobert [538], Salazar [735], Sepehri [749], and Fischetti, Salazar and Toth [299, 300]. Their asymmetric counterparts have been investigated in Laporte, Mercure and Nobert [537], and Noon and Bean [630]. Transformations from the GTSP to the asymmetric TSP have been proposed by Noon and Bean [631], Lien, Ma and Wah [561], Dimitrijević and Saric [255], and Laporte and Semet [541]. See also Cvetkrović, Dimitrijević and Milosavljević [236]. Semet and Renaud [748] presented a tabu-search algorithm for the E-GTSP.

A more detailed analysis of both GTSP and E-GTSP will be given in Section 2.

1.7. The Covering Tour Problem

The *Covering Tour Problem* is a variant of the Generalized TSP arising when the clusters C_k are not necessarily disjoint. This problem was introduced by Gendreau, Laporte and Semet [354] as a model for the location of post boxes and for the planning of routes for medical teams in developing countries, where each cluster C_k corresponds to the subset of nodes located within a given distance of a certain customer k . They analyzed the polyhedral structure of the problem, presented a branch-and-cut algorithm, and tested its performance on random instances with up to 100 nodes. Current and Schilling [233] studied a multi-objective version of the same problem, that they called the *Covering Salesman Problem*.

1.8. The Median Cycle Problem

The *Median Cycle Problem* looks for a min-cost cycle \tilde{E} such that the sum of the distances between each node not in \tilde{E} and its closest node in \tilde{E} does not exceed a given value d_0 . In other words, the Median Cycle Problem looks for a min-cost cycle \tilde{E} such that

$$\sum_{i \notin V(\tilde{E})} \min_{j \in V(\tilde{E})} d_{ij} \leq d_0,$$

where d_{ij} represents the distance between nodes i and j .

This problem was introduced by Labbé, Laporte, Rodríguez and Salazar [527] as a location model for circular-shaped transportation infrastructures. These authors provided a polyhedral analysis and a branch-and-cut algorithm tested on random instances with up to 150 nodes. Current and Schilling [234] proposed heuristics for a variant of this problem which consists of finding a cycle \tilde{E} visiting no more than p nodes,

while minimizing the weighted sum of the cycle cost and of the largest distance of the nodes in \tilde{E} from the unrouted nodes (the latter being computed as $\max_{i \notin V(\tilde{E})} \min_{j \in V(\tilde{E})} d_{ij}$). They also studied the problem of minimizing the cycle cost while imposing

$$\max_{i \notin V(\tilde{E})} \min_{j \in V(\tilde{E})} d_{ij} \leq d_0.$$

1.9. The Traveling Purchaser Problem

In the *Traveling Purchaser Problem* node 1 corresponds to the purchaser’s domicile, and the other nodes to markets. There are m products to be purchased, the k th product being available in a given cluster of markets C_k . The problem looks for a cycle starting at the domicile and purchasing each product while minimizing the sum of the cycle cost plus the purchasing costs. To be more specific, let f_{ik} be the cost of purchasing product k at node $i \in C_k$. As in the Covering Tour problem, a cycle \tilde{E} is considered feasible if and only if $1 \in V(\tilde{E})$ and $V(\tilde{E}) \cap C_k \neq \emptyset$ for all $k = 1, \dots, m$. The Traveling Purchaser Problem then calls for a min-cost feasible cycle, the cost of a feasible cycle \tilde{E} being computed as

$$\sum_{e \in \tilde{E}} c_e + \sum_{k=1}^m \min\{f_{ik} : i \in C_k \cap V(\tilde{E})\}$$

In the *Capacitated Traveling Purchaser Problem*, for each product k we also have a required amount d_k to be purchased, while the quantity of product k available at each node $i \in C_k$ is q_{ik} . In this version, f_{ik} represents the cost of purchasing one unit of product k at node $i \in C_k$, and the objective is to find a route collecting the required amount d_k for each product k , while minimizing the sum of the routing and the purchasing costs.

The uncapacitated version of the problem was originally introduced by Burstall [152] and Ramesh [694]. Heuristic methods have been proposed by, e.g., Voss [814], Golden, Levy and Dahl [386], Ong [632], Pearn and Chien [662], and Boctor, Laporte and Renaud [119],

Branch-and-bound exact algorithms have been studied by, e.g., Singh and van Oudheusden [762], reporting the solution of 25-node instances. Laporte, Riera and Salazar [540] proposed a branch-and-cut algorithm for the exact solution of the capacitated version, which is capable of solving random instances involving up to 200 nodes.

2. The Generalized Traveling Salesman Problem

As already stated, in the GTSP we are given a proper partition of V into $m \geq 3$ node subsets C_1, \dots, C_m , called *clusters*. A cycle is consid-

ered feasible if it goes through each cluster at least once. The GTSP then consists of finding a feasible cycle $\tilde{E} \subset E$ whose global cost $\sum_{e \in \tilde{E}} c_e$ is a minimum, and can be formulated as

$$v(GTSP) \equiv \min \sum_{e \in E} c_e x_e \tag{13}$$

subject to

$$\sum_{e \in \delta(v)} x_e = 2 y_v \quad \text{for } v \in V \tag{14}$$

$$\sum_{v \in C_h} y_v \geq 1 \quad \text{for } h = 1, \dots, m \tag{15}$$

$$\sum_{e \in \delta(S)} x_e \geq 2(y_i + y_j - 1) \quad \text{for } S \subset V, 2 \leq |S| \leq n - 2, \tag{16}$$

$$i \in S, j \in V \setminus S$$

$$x_e \in \{0, 1\} \quad \text{for } e \in E \tag{17}$$

$$y_v \in \{0, 1\} \quad \text{for } v \in V. \tag{18}$$

As to the E-GTSP, arising when imposing that each cluster must be visited exactly once, a mathematical model is obtained from (13)–(18) by replacing (15) with

$$\sum_{v \in C_h} y_v = 1 \quad \text{for } h = 1, \dots, m. \tag{19}$$

Model (13)–(18) heavily relies on the integrality of the y variables. If this requirement is relaxed, solutions like those of Figure 2 become feasible. Therefore, the LP relaxation of this model can be very poor. Additional valid inequalities will be described in the sequel, whose introduction in the model leads to a considerable strengthening of its LP relaxation.

As shown in [300], the E-GTSP is polynomially solvable when the sequence of the clusters is known, which implies the polynomial solvability for fixed m (see Subsection 2.5 for more details).

The remaining part of the present section is mainly based on the results given by Fischetti, Salazar and Toth in [299] and [300]. We first analyze the facial structure of the GTSP and the E-GTSP polytopes; in particular, in Section 2.2 we introduce a general theorem that allows one to lift any facet of the TSP polytope into a facet of the GTSP polytope. This result is used to derive classes of facet-inducing inequalities related to the *subtour elimination* and *comb* constraints. In Section 2.3 we analyze the E-GTSP polytope and discuss the cases in which the

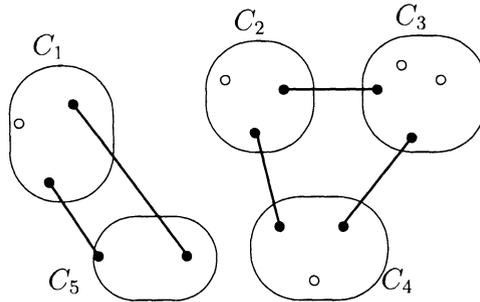


Figure 13.1. Infeasible GTSP solution satisfying (14)–(17). The drawn edges have $x_e = 1$, the blank nodes $y_v = 0$, and the black nodes $y_v = 1/2$.

inequalities of Section 2.2 are facet-inducing. The analysis is based on a general result which allows one to inductively reduce the polyhedral analysis of the E-GTSP polytope to that of the TSP polytope.

These theoretical results are used in Section 2.6 to design a branch-and-cut algorithm for the exact solution of large-scale E-GTSP instances. The algorithm is based on exact/heuristic separation procedures for the main classes of inequalities previously analyzed. We finally report results on test instances involving up to 442 nodes, showing that these inequalities lead to a substantial improvement of the LP relaxation of the original model.

2.1. Basic notations

Let P , $P^=$, and Q denote, respectively, the *GTSP*, *E-GTSP*, and *TSP polytopes*, defined as

$$P := \text{conv}\{(x, y) \in \mathbb{R}^{E \cup V} : (14)\text{--}(18) \text{ hold}\},$$

$$P^= := P \cap \{(x, y) \in \mathbb{R}^{E \cup V} : (19) \text{ holds}\},$$

and

$$Q := P \cap \{(x, y) \in \mathbb{R}^{E \cup V} : y_v = 1 \text{ for all } v \in V\}.$$

Clearly, $P^=$ and Q are faces of P . These faces are disjoint when $m < n$, whereas for $m = n$ the three polytopes P , $P^=$, and Q coincide.

We assume the reader is familiar with the foundations of polyhedral theory. For the sake of simplicity, in the following we will not distinguish between a GTSP (or E-GTSP) solution and its characteristic vector, and assume $m \geq 5$. Moreover, we will make use of the following notation:

$$\begin{aligned} \mu(S) &:= |\{h : C_h \subseteq S\}| && \text{for } S \subseteq V, \\ \eta(S) &:= |\{h : C_h \cap S \neq \emptyset\}| && \text{for } S \subseteq V, \end{aligned}$$

and denote by $C_{h(v)}$ the cluster containing a given node v . We also define

$$W := \{v \in V : |C_{h(v)}| = 1\}.$$

2.2. Facet-defining inequalities for the GTSP polytope

In this section we study the GTSP polytope, P . The facial structure of P is clearly related to that of the TSP polytope, Q , arising when imposing the additional equations $y_v = 1$ for all $v \in V$. In order to link these two polytopes, let us define the intermediate polytopes

$$P(F) := P \cap \{(x, y) \in \mathbb{R}^{E \cup V} : y_v = 1 \text{ for all } v \in F\},$$

where $\emptyset \subseteq F \subseteq V$. By definition, $P(V) = Q$ and $P(\emptyset) = P$.

Our first order of business is to determine the dimension of $P(F)$ for any given F . This amounts to studying the equation system for $P(F)$. This system includes the $|V|$ linearly independent equations (14), plus the variable fixing equations

$$y_v = 1 \quad \text{for all } v \in F \cup W, \tag{20}$$

where W has been defined previously. Actually, no other linearly independent equations satisfied by all the points of $P(F)$ exist, as implied by the following result.

Theorem 1 *For all $F \subseteq V$, $\dim(P(F)) = |E| - |F \cup W|$.*

Proof: Clearly $\dim(P(F)) \leq |E| - |F \cup W|$ since $P(F) \subset \mathbb{R}^{E \cup V}$ and the $|V| + |F \cup W|$ valid equations (14) and (20) are linearly independent. We claim the existence of $|E| - |F \cup W| + 1$ affinely independent points in $P(F)$. This will prove $\dim(P(F)) \geq |E| - |F \cup W|$, and hence the theorem. The proof of the claim is by induction on the cardinality of F .

When $|F| = n$ the claim is true, since $P(F)$ corresponds to the TSP polytope (see, e.g., Grötschel and Padberg [405] or Chapter 2).

Assume now the claim holds for $|F| = \sigma$, and consider any node set F' with $|F'| = \sigma - 1$. Let v be any node not in F' , and define $F := F' \cup \{v\}$. Because of the induction hypothesis, there exist $|E| - |F \cup W| + 1$ affinely independent points belonging to $P(F)$ hence to $P(F')$. If $v \in W$ then $|F \cup W| = |F' \cup W|$, and we are done. Otherwise, $|F \cup W| = |F' \cup W| + 1$, i.e., we need an additional point. Such a point always exists, and corresponds to any Hamiltonian cycle in the subgraph induced by $V \setminus \{v\}$. ■

Corollary 2 $\dim(P) = |E| - |W|$.

According to Theorem 1, given any nonempty $F \subseteq V$ and any $v \in F$ one has the following: if $v \in W$ then $\dim(P(F \setminus \{v\})) = \dim(P(F))$, else $\dim(P(F \setminus \{v\})) = \dim(P(F)) + 1$. In other words, the removal of a node from F increases the dimension of $P(F)$ by, at most, one unit. As a consequence, any facet-defining inequality for $P(F)$ can be lifted in a simple way so as to be facet-inducing for $P(F \setminus \{v\})$ as well.

Theorem 3 Let $F \subseteq V$ and $u \in F$. In addition, let

$$\sum_{e \in E} \alpha_e x_e + \sum_{v \in V} \beta_v (1 - y_v) \geq \gamma$$

be any facet-inducing inequality for $P(F)$. Then the lifted inequality

$$\sum_{e \in E} \alpha_e x_e + \sum_{v \in V \setminus \{u\}} \beta_v (1 - y_v) + \tilde{\beta}_u (1 - y_u) \geq \gamma$$

is valid and facet-defining for $P(F \setminus \{u\})$, where $\tilde{\beta}_u$ is an arbitrary value if $u \in W$, whereas

$$\tilde{\beta}_u = \gamma - \min \left\{ \sum_{e \in E} \alpha_e x_e + \sum_{v \in V \setminus \{u\}} \beta_v (1 - y_v) : \begin{array}{l} (x, y) \in P(F \setminus \{u\}), \\ \text{and } y_u = 0 \end{array} \right\}$$

holds when $u \notin W$.

Proof: The claim follows from the well-known sequential lifting theorem (Padberg [641]), as described, e.g., in Grötschel and Padberg [405]. ■

Theorem 3 leads to a lifting procedure to be used to derive facet-inducing inequalities for the GTSP polytope from those of the TSP polytope. To this end one has to choose any *lifting sequence* for the nodes, say $\{v_1, \dots, v_n\}$, and iteratively derive a facet of $P(\{v_{t+1}, \dots, v_n\})$ from a facet of $P(\{v_t, \dots, v_n\})$ for $t = 1, \dots, n$. Different lifting sequences can produce different facets.

By using the above lifting procedure one can easily prove the following results.

Theorem 4 *The following inequalities define facets of P :*

$$x_e \geq 0 \quad \text{for every } e \in E, \tag{21}$$

$$x_e \leq 1 \quad \text{whenever } e \in E(W), \tag{22}$$

$$y_v \leq 1 \quad \text{whenever } v \notin W, \tag{23}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \text{for } S \subset V : \begin{matrix} 2 \leq |S| \leq n-2, \\ \mu(S) \neq 0, \mu(V \setminus S) \neq 0. \end{matrix} \tag{24}$$

$$\sum_{e \in \delta(S)} x_e \geq 2y_i \quad \text{for } S \subset V : \begin{matrix} 2 \leq |S| \leq n-2, \\ \mu(S) = 0, \mu(V \setminus S) \neq 0, \\ i \in S, \end{matrix} \tag{25}$$

$$\sum_{e \in \delta(S)} x_e \geq 2(y_i + y_j - 1) \quad \text{for } S \subset V : \begin{matrix} 2 \leq |S| \leq n-2, \\ \mu(S) = \mu(V \setminus S) = 0, \\ i \in S, j \in V \setminus S. \end{matrix} \tag{26}$$

By possibly interchanging the role of S and $V \setminus S$, one can always assume that inequalities (24) and (26) are written for $S \subset V$ such that $|S| \leq \lfloor n/2 \rfloor$. The same holds for inequalities (25) by choosing $i \in V \setminus S$ when $\mu(S) \neq 0$ and $\mu(V \setminus S) = 0$.

Notice that (25) are also valid (but not facet-inducing) when $\mu(S) \neq 0$. Analogously, inequalities (26) hold for any $S \subset V$ and coincide with (16).

By exploiting equations (14), inequalities (24), (25) and (26) above can be rewritten, for any $S \subset V$ with $2 \leq |S| \leq n-2$, as the following *Generalized Subtour Elimination Constraints* (GSEC's):

$$\sum_{e \in E(S)} x_e \leq \sum_{v \in S} y_v - 1 \quad \text{for } \mu(S) \neq 0, \mu(V \setminus S) \neq 0, \tag{27}$$

$$\sum_{e \in E(S)} x_e \leq \sum_{v \in S \setminus \{i\}} y_v \quad \text{for } \mu(S) = 0, \mu(V \setminus S) \neq 0, \tag{28}$$

$$i \in S,$$

$$\sum_{e \in E(S)} x_e \leq \sum_{v \in S \setminus \{i\}} y_v - y_j + 1 \quad \text{for } \mu(S) = \mu(V \setminus S) = 0, \tag{29}$$

$$i \in S, j \notin S.$$

This form of the constraints has the advantage of having fewer nonzero coefficients (assuming $|S| \leq \lfloor n/2 \rfloor$), hence it is more suitable for a cutting plane approach.

Particular cases of GSEC's arise when $|S| = 2$, leading to

$$x_e \leq y_v \quad \text{for } v \in V, e \in \delta(v). \tag{30}$$

Note that inequality $\sum_{v \in C_h} y_v \geq 1$ does not define a facet of P for any $h = 1, \dots, m$. Indeed, because of (14), constraint (15) is equivalent to $\sum_{e \in \delta(C_h)} x_e + 2 \sum_{e \in E(C_h)} x_e \geq 2$, hence it is dominated by the valid

inequality $\sum_{e \in \delta(C_h)} x_e \geq 2$ when $E(C_h) \neq \emptyset$, whereas for $E(C_h) = \emptyset$ (i.e., when $|C_h| = 1$) it defines the improper face of P .

We finally consider the TSP *comb* inequalities. A comb is a family $C = (H, T_1, \dots, T_s)$ of $s + 1$ node subsets, where $s \geq 3$ is an odd integer. H is called the *handle* of C , whereas T_1, \dots, T_s are called *teeth*. Moreover, the following conditions must be satisfied: (i) T_1, \dots, T_s are pairwise disjoint; (ii) $T_j \cap H \neq \emptyset$ and $T_j \setminus H \neq \emptyset$ for $j = 1, \dots, s$. The *size* of C is defined as $\sigma(C) := |H| + \sum_{j=1}^s (|T_j| - 1) - (s + 1)/2$.

The *comb inequality* associated with C is

$$\sum_{e \in E(H)} x_e + \sum_{j=1}^s \sum_{e \in E(T_j)} x_e \leq \sigma(C), \tag{31}$$

and is valid and facet-defining for the TSP (see Grötschel and Padberg [405], or Chapter 2). It is well known that interchanging the role of H and $V \setminus H$ produces an equivalent formulation of (31).

Starting with (31), one can obtain related facet-defining inequalities for the GTSP by using the lifting Theorem 3 (trivially modified so as to deal with “ \leq ” inequalities).

Theorem 5 *Let $C = (H, T_1, \dots, T_s)$ be a comb. For $j = 1, \dots, s$, let a_j be any node in $T_j \cap H$ if $\mu(T_j \cap H) = 0$, $a_j = 0$ (a dummy value) otherwise; and let b_j be any node in $T_j \setminus H$ if $\mu(T_j \setminus H) = 0$, $b_j = 0$ otherwise. Then the following generalized comb inequality is valid and facet-defining for P :*

$$\sum_{e \in E(H)} x_e + \sum_{j=1}^s \sum_{e \in E(T_j)} x_e + \sum_{v \in V} \tilde{\beta}_v (1 - y_v) \leq \sigma(C), \tag{32}$$

where $\tilde{\beta}_v = 0$ for all $v \in V \setminus (H \cup T_1 \cup \dots \cup T_s)$, $\tilde{\beta}_v = 1$ for all $v \in H \setminus (T_1 \cup \dots \cup T_s)$, and for $j = 1, \dots, s$: $\tilde{\beta}_v = 2$ for $v \in T_j \cap H$, $v \neq a_j$; $\tilde{\beta}_{a_j} = 1$ if $a_j \neq 0$; $\tilde{\beta}_v = 1$ for $v \in T_j \setminus H$, $v \neq b_j$; $\tilde{\beta}_{b_j} = 0$ if $b_j \neq 0$.

2.3. Facet-defining inequalities for the E-GTSP polytope

We now address the polyhedral structure of the E-GTSP polytope, $P^=$. This polytope is clearly a face of P , hence all facet-defining inequalities for P studied in Section 2.2 are also valid (but not necessarily facet-defining) for $P^=$.

Since in the E-GTSP exactly one node of each cluster must be visited, we can drop intra-cluster edges, and re-define the edge-set as

$$E := \{(i, j) : i \in V, j \in V \setminus C_{h(i)}\}.$$

In view of this reduction, constraints (19) are equivalent to

$$\sum_{e \in \delta(C_h)} x_e = 2 \quad \text{for all } h = 1, \dots, m,$$

and a simplified model for the E-GTSP can be obtained by replacing constraints (16) and (18) by the two families of constraints:

$$\sum_{e \in E(S)} x_e \leq r - 1 \quad \text{for } S = \cup_{i=1}^r C_{l_i} \text{ and } 3 \leq r \leq m - 3, \quad (33)$$

$$\sum_{e \in \delta(C_l) \cap \delta(w)} x_e \leq y_w \quad \text{for } l = 1, \dots, m \text{ and } w \in V \setminus C_l, \quad (34)$$

respectively.

Inequalities (33) will be called *Basic Generalized Subtour Elimination Constraints* (Basic GSEC's), and (34) will be called *fan inequalities*. Both are particular cases of the GSEC's (27) because of (19). Indeed, (33) are equivalent to the GSEC's (27) written for $S = \cup_{i=1}^r C_{l_i}$, where $\sum_{v \in S} y_v = r$ since $\sum_{v \in C_{l_i}} y_v = 1$ for $i = 1, \dots, r$. Analogously, (34) arise from (27) when $S = C_l \cup \{w\}$, since $E(S) = \delta(C_l) \cap \delta(w)$ and $\sum_{v \in S} y_v = \sum_{v \in C_l} y_v + y_w = 1 + y_w$. Notice that constraints (34) dominate (30).

As in the previous subsection, we aim at relating the facial structure of $P^=$ to that of the TSP polytope, Q , even though Q is not a relaxation of $P^=$. To this end, let us introduce some basic definitions.

Definition 6 Given a valid inequality $\alpha x + \beta y \leq \gamma$ for $P^=$, let $\mathcal{H}_{\alpha, \beta, \gamma}^= := P^= \cap \{(x, y) \in \mathbb{R}^{E \cup V} : \alpha x = \beta y + \gamma\}$ denote the face of $P^=$ induced by $\alpha x + \beta y \leq \gamma$. Let $v \in V \setminus W$ be an arbitrary but fixed node, and let $P_v^=$ denote the E-GTSP polytope associated with the subgraph of G induced by $V \setminus \{v\}$. The v -restriction of $\alpha x + \beta y \leq \gamma$ is the inequality obtained from $\alpha x + \beta y \leq \gamma$ by dropping the variables y_v and x_e for all $e \in \delta(v)$. The v -compatibility graph of $\alpha x + \beta y \leq \gamma$ is the graph $G_v^* = (V \setminus C_{h(v)}, E^*)$ with $(u, w) \in E^*$ if and only if there exists $(x, y) \in \mathcal{H}_{\alpha, \beta, \gamma}^=$ with $x_{(v,u)} = x_{(v,w)} = 1$.

The rank of a graph is defined as the rank of its edge-node incidence matrix, i.e., the number of its nodes minus the number of its bipartite connected components. The graph is said to be of full rank when its edge-node incidence matrix is of full rank, i.e., when it has an odd cycle for each connected component.

Lemma 7 For every valid inequality $\alpha x + \beta y \leq \gamma$ for $P^=$ and every node $v \in V \setminus W$ the dimension of $\mathcal{H}_{\alpha, \beta, \gamma}^=$ is greater or equal to the dimension

of the face of $P_v^=$ induced by its v -restriction, plus the rank of its v -compatibility graph.

Proof: Let X be the matrix in which every row is an extreme point of $\mathcal{H}_{\alpha,\beta,\gamma}^=$. Since $\mathcal{H}_{\alpha,\beta,\gamma}^=$ is contained in a hyperplane not passing through the origin (e.g., that induced by (19) for $h = 1$), a subset of rows of X is affinely independent if and only if it is linearly independent. Hence the dimension of $\mathcal{H}_{\alpha,\beta,\gamma}^=$ coincides with the rank of X minus 1. Now, X can be partitioned into

$$X = \begin{bmatrix} X_{11} & 0 & 0 \\ X_{21} & X_{22} & 1 \end{bmatrix},$$

where the last column corresponds to variable y_v , and the columns of X_{22} correspond to variables x_e for $e \in \delta(v)$. Then the rank of X is, at least, the sum of the rank of X_{11} plus the rank of $[X_{22} \ 1]$. By construction, the rank of X_{11} is the dimension of the face of $P_v^=$ induced by the v -restriction of $\alpha x + \beta y \leq \gamma$, plus 1. As to X_{22} , we observe that each of its rows contains exactly two 1's and (barring repeated rows) can be viewed as the edge-node incidence matrix of the v -compatibility graph G_v^* associated with $\alpha x + \beta y \leq \gamma$. Moreover, the last column of $[X_{22} \ 1]$ is a linear combination (with coefficients $1/2$) of the other columns. This proves the claim. ■

Lemma 7 allows us to extend some known results from the TSP polytope to the E-GTSP case by using induction on $\rho = \sum_{h=1}^m (|C_h| - 1) = n - m$.

As shown in Lemma 7, the rank of the v -compatibility graph G_v^* associated with a given inequality $\alpha x + \beta y \leq \gamma$ plays a central role when analyzing the polyhedral structure of $P^=$. Unfortunately, determining whether an edge is present in G_v^* requires the construction of a suitable E-GTSP solution (x, y) with $\alpha x = \beta y + \gamma$, hence it is an \mathcal{NP} -hard problem in general. In practice, one is interested in finding sufficient conditions for the existence of an edge in G_v^* . We next describe one such condition, related to the work of Naddef and Rinaldi [618] for the graphical TSP, and of Balas and Fischetti [72, 73] for the asymmetric TSP.

Definition 8 *An inequality $\alpha x + \beta y \leq \gamma$ is said to be Tight-Triangular (TT, for short) when for all $v \in V$ one has*

$$\beta_v = \max\{\alpha_{iv} + \alpha_{jv} - \alpha_{ij} : (i, j) \in E \setminus \delta(C_{h(v)})\}.$$

For $v \in V$, we denote by

$$\Delta(v) := \{(i, j) \in E \setminus \delta(C_{h(v)}) : \beta_v = \alpha_{iv} + \alpha_{jv} - \alpha_{ij}\}$$

the set of the tight edges for v .

Recall that a face \mathcal{H} of $P^=$ is called *trivial* when $\mathcal{H} \subseteq \{(x, y) \in \mathbb{R}^{E \cup V} : x_e = 0\}$ for some $e \in E$; *nontrivial* otherwise.

Lemma 9 *Let $v \in V \setminus W$, and let $\alpha x + \beta y \leq \gamma$ be a valid TT inequality for $P^=$ whose v -restriction defines a nontrivial face of $P_v^=$. Then G_v^* contains all the edges in $\Delta(v)$.*

A more sophisticated lifting procedure for the E-GTSP, which allows one to extend any given facet of the TSP polytope to a facet of $P^=$, is given in [299].

We are now ready to study the facial structure of $P^=$.

Theorem 10 $\dim(P^=) = |E| - m$.

Proof: Clearly, $\dim(P^=) \leq |E| - m$ as equations (14) and (19) are linearly independent. Hence, it remains to be proved that the dimension of the (improper and nontrivial) face $\mathcal{H}(0, 0, 0)$ induced by $0x \leq 0y + 0$ is not less than $|E| - m$. We use induction on $\rho = n - m$.

When $\rho = 0$ we have the standard TSP case, and the claim is true.

Assume now the claim holds for $\rho = \bar{\rho}$, and consider any E-GTSP instance with $\rho = \bar{\rho} + 1$. Then there exists a node $v \in V \setminus W$. Because of Lemma 7, we have $\dim(\mathcal{H}(0, 0, 0)) \geq d_1 + d_2$, where d_1 is the dimension of the face of $P_v^=$ induced by the v -restriction of $0x \leq 0y + 0$, and d_2 is the rank of the v -compatibility graph G_v^* associated with $0x \leq 0y + 0$. By the induction hypothesis, $d_1 \geq |E \setminus \delta(v)| - m$, thus it remains to be shown that $d_2 = |\delta(v)| = |V \setminus C_{h(v)}|$, i.e., that G_v^* is of full rank. But this follows easily from Lemma 9, since $\Delta(v)$ contains all the edges in $E \setminus \delta(C_{h(v)})$ and, therefore, G_v^* is connected and contains an odd cycle (recall that $m \geq 5$ is assumed). ■

Using similar arguments, one can prove the following results.

Theorem 11 *The following inequalities define facets of $P^=$:*

- (1) $x_e \geq 0$ for all $e \in E$.
- (2) $x_e \leq 1$ for all $e \in E(W)$.
- (3) the GSEC (27) whenever one of the following conditions holds:
 - (i) $S \subseteq W$ and $|S| = 2$,
 - (ii) $S = C_l \cup \{w\}$ for some $w \in V \setminus (C_l \cup W)$,
 - (iii) $\eta(S) \geq 3$ and $\eta(V \setminus S) \geq 3$,

where $\eta(\cdot)$ has been defined in Subsection 2.1.

(4) The fan inequality (34) for all $w \notin W$.

(5) The GSEC inequality (27) whenever both S and $V \setminus S$ overlap, at least, 3 clusters each, i.e., when $\eta(S) \geq 3$ and $\eta(V \setminus S) \geq 3$.

Note that, because of (34), $y_v \geq 0$ does not define a facet for any $v \in V$. Analogously, the GSEC's (28) and (29) do not define facets of P° . Indeed, a GSEC (29) can be written as $\sum_{e \in E(S)} x_e - \sum_{v \in S \setminus \{i\}} y_v + y_j - 1 \leq 0$. If $C_{h(i)} = C_{h(j)}$, then $y_i + y_j \leq 1$, hence the inequality (29) is a weakening of $\sum_{e \in E(S)} x_e - \sum_{v \in S} y_v \leq 0$ which is, in turn, strictly dominated by the equation $\frac{1}{2} \sum_{v \in S} (\sum_{e \in \delta(v)} x_e - 2y_v) = 0$. Otherwise (29) is dominated by the GSEC (28) written for $S' := S \setminus C_{h(j)}$, i.e., by $\sum_{e \in E(S')} x_e - \sum_{v \in S' \setminus \{i\}} y_v \leq 0$.

Similarly, one can show that a GSEC (28) is dominated by the GSEC (27) written for $S' := S \cup C_{h(i)}$.

The bound constraint $x_{(i,j)} \leq 1$ does not define a facet whenever $i \notin W$ or $j \notin W$, since in this case it is dominated by the fan inequality $\sum_{e \in \delta(C_{h(j)} \cap \delta(i))} x_e \leq y_i$ (if $i \notin W$), or $\sum_{e \in \delta(C_{h(i)} \cap \delta(j))} x_e \leq y_j$ (if $j \notin W$). In addition, the bound constraints $y_v \leq 1$ never define a facet of P° because of equations (19).

Finally, the GSEC's (27) not covered by Theorems 11 do not define facets in that $E(S)$ induces a bipartite graph, hence they can be obtained as the sum of certain fan inequalities, as shown in [299].

2.4. Separation algorithms

In this subsection we address the following *separation* (or *identification*) *problem*: Given a (fractional) point $(x^*, y^*) \in [0, 1]^{E \cup V}$, find a member $\alpha x + \beta y \geq \gamma$ of a given family \mathcal{F} of valid inequalities for GTSP (or E-GTSP), such that $\alpha x^* + \beta y^* < \gamma$. An effective exact/heuristic solution of this problem is of fundamental importance in order to use the inequalities of \mathcal{F} within a cutting plane algorithm for the exact/heuristic solution of the problem. In the following we describe the separation algorithms proposed by Fischetti, Salazar and Toth [300].

2.4.1 An exact separation algorithm for GSEC's. We consider the family \mathcal{F} of the generalized subtour elimination constraints, in their cut form (24)–(26). We will assume that node subset $S \subset V$ satisfies $2 \leq |S| \leq n - 2$.

We start with constraints (26):

$$\sum_{e \in \delta(S)} x_e \geq 2(y_i + y_j - 1) \quad \text{if } \mu(S) = \mu(V \setminus S) = 0, i \in S, j \in V \setminus S.$$

Suppose nodes i and j have been fixed. Then, finding a most violated inequality (26) calls for the minimum-capacity cut $(S, V \setminus S)$ with $i \in S$ and $j \in V \setminus S$ in the capacitated undirected graph G^* obtained from G by imposing a capacity x_e^* for each $e \in E$. This can be done in $O(n^3)$ time, as it amounts to finding the maximum flow from i to j (see, e.g., Ahuja, Magnanti, Orlin [6]). If the maximum flow value is not less than $2(y_i^* + y_j^* - 1)$, then all the inequalities (26) for the given pair (i, j) are satisfied; otherwise the capacity of the minimum cut separating i and j is strictly less than $2(y_i^* + y_j^* - 1)$ and a most violated inequality (26) has been detected among those for the given pair (i, j) . Trying all possible pairs (i, j) then produces an $O(n^5)$ overall separation algorithm. Actually, a better algorithm having overall $O(n^4)$ time complexity can be obtained, in analogy with the TSP case (see Padberg and Grötschel [642] or Chapter 2) by using the Gomory-Hu [390] scheme for the multiterminal flow problem. A simpler algorithm with the same time complexity is based on the simple observation that, for any S , the most violated inequality (26) arises when the chosen i and j are such that $y_i^* = \max\{y_v^* : v \in S\}$ and $y_j^* = \max\{y_v^* : v \in V \setminus S\}$. Therefore, any node s with $y_s^* = \max\{y_v^* : v \in V\}$ can always be fixed to play the role of, say, node i . In this way, one as to solve (at most) $n - 1$ max-flow problems in the attempt to send $2(y_s^* + y_j^* - 1)$ units of flow from s to any $j \in V \setminus \{s\}$. Clearly, nodes j with $y_s^* + y_j^* - 1 \leq 0$ need not be considered.

We now address inequalities (25):

$$\sum_{e \in \delta(S)} x_e \geq 2y_i \quad \text{if } \mu(S) = 0, \mu(V \setminus S) \neq 0, i \in S.$$

As before, we assume that cluster C_h and node $i \notin C_h$ are fixed. In this case a most violated constraint (25) corresponds to a minimum-capacity cut $(S, V \setminus S)$ with $i \in S$ and $C_h \subseteq V \setminus S$ in the capacitated graph G^* . Hence it can be detected by finding the maximum flow from i to t , where t is an additional node connected with each $j \in C_h$ through an edge having very large capacity (this corresponds to shrinking cluster C_h into a single node). Trying all (i, C_h) pairs leads to an $O(mn^4)$ time algorithm. Clearly, nodes i with $y_i^* = 0$ need not to be considered.

We now address constraints (24)

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \text{if } \mu(S) \neq 0, \mu(V \setminus S) \neq 0.$$

For all pairs (C_h, C_k) of distinct clusters, a most violated inequality (24) is detected by finding the maximum flow from s to t , where s (resp. t) is an additional node connected with each $j \in C_h$ (resp. $j \in C_k$) by means of an edge having very large capacity. The overall time complexity of this phase is $O(m^2 n^3)$.

Notice that a violated inequality (25) or (26) found by the above described separation algorithm, is not necessarily facet-defining. For (26) this occurs when there exists a cluster C_h contained in S or $V \setminus S$; for (25), this happens when there exists a cluster contained in the shore of the cut including node i . In these cases one should obviously reject the inequality in favor of its facet-inducing strengthening (24) or (25).

According to the above scheme, the separation algorithm for the overall family \mathcal{F} containing inequalities (24)–(26) requires $O(m n^4)$ time in the worst case. In practice, the computing time required is typically much smaller as the capacitated graph G^* is very sparse, and has many isolated nodes. Moreover, as previously explained, several max-flow computations can be avoided because some entries of y^* have a small value. In addition, parametric considerations on the structure of the cuts can further reduce the number of max-flows computations.

We now consider the important case in which $y_s^* := \max\{y_v^* : v \in V\} = 1$, that arises very frequently during the cutting-plane algorithm. In this case one can find a most violated generalized subtour elimination constraint by computing no more than $n + m - 2$ max-flows, with overall $O(n^4)$ time complexity. Indeed, the degree of violation of any inequality (24) with, say, $C_h \subseteq S$ and $C_k \subseteq V \setminus S$ is the same as that associated with inequality (25) written for the same S and for $i = s$. Hence inequalities (24) need not to be considered. Now consider any inequality (25) with $i \neq s$. To fix the ideas, let $i \in S$ and $C_h \subseteq V \setminus S$. If $s \in S$, then the degree of violation of the inequality does not decrease by replacing i with s . Otherwise, the degree of violation is the same as that of inequality (26) written for $j = s$. It follows that inequalities (25) with $i \neq s$ need not be considered. As a result, one has to consider explicitly only the inequalities (25) with $i = s$, and the inequalities (26) (for which $i = s$ can again be assumed).

The reader is referred to [300] for an efficient (parametric) implementation of the above separation procedures, called GSEC_SEP in the sequel.

A heuristic separation algorithm for GSEC's The exact separation algorithm given in the previous subsection can be excessively time consuming. We now outline two faster heuristic procedures.

The first procedure, GSEC_H1, considers the following subset of the inequalities (24):

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \text{if } \mu(S) \neq 0, \mu(V \setminus S) \neq 0,$$

with S containing a cluster C_l of smallest size. For each $h \in \{1, \dots, m\} \setminus \{l\}$, the procedure computes a most violated inequality (24) with $C_l \subseteq S$ and $C_h \subseteq V \setminus S$ by finding the maximum flow from C_l to C_h . This procedure has $O(mn^3)$ time complexity, and typically runs much faster than GSEC_SEP.

Both the exact separation procedure and GSEC_H1 produce a list of violated inequalities chosen on the basis of their individual degree of violation, rather than on their combined effect. In order to speed up the convergence of the cutting plane phase, instead, for each round of separation it is advisable to produce a family of violated inequalities “spanning” the overall graph. To support this point, consider the simplest problem involving subtour-elimination constraints, namely the *Shortest Spanning Tree* (SST) problem. It is known from matroid theory that the node subsets whose associated subtour elimination constraints are active at the optimum, define a nested family covering all the nodes of the graph. Therefore, a cutting plane SST algorithm that adds violated cuts chosen only on the basis of their individual degree of violation, is likely to require a high number of iterations before producing the optimal family of cuts. In this view, the *shrinking* technique used in Padberg and Rinaldi [645, 647] for the TSP, besides reducing the computational effort spent in each separation, has the advantage of quickly producing a nested family of constraints spanning the graph.

We next describe a heuristic separation algorithm for GSEC’s, based on the previous considerations. In order to illustrate the basic idea underlying the algorithm, let us restrict our attention to the standard TSP. Given the fractional point x^* , we look for a family of violated subtour elimination constraints. To this end, let us consider the polytope

$$Q^{SEC} := \{x \geq 0 : \sum_{e \in E(S)} x_e \leq |S| - 1, \text{ for } S \subset V, |S| \geq 2\},$$

whose vertices are the incidence vectors of the forests spanning the graph. A node of Q^{SEC} “close” to x^* , say \tilde{x} , is found, and the violation of (some of) the subtour elimination constraints defining facets of Q^{SEC} passing through \tilde{x} is checked. To be more specific, \tilde{x} is determined by solving the problem $\max\{x^*x : x \in Q^{SEC}\}$, i.e., by finding a maximum weight spanning tree of G with edge weights $x_e^* \geq 0, e \in E$. The classical

greedy algorithm of Kruskal [523] is used, and a check is performed on the violation of the $n - 1$ SEC's associated with the subsets $S_i \subset V$, $i = 1, \dots, n - 1$, corresponding to the connected components iteratively found. From matroid theory (see, e.g., Nemhauser and Wolsey [625, page 669]), the SEC's associated with these subsets S_i are the only ones needed to prove the optimality of \tilde{x} (since all other SEC's can be relaxed without affecting the optimality of \tilde{x}), hence they are likely to be violated by x^* . Notice that (some of) the S_i sets found by the above sketched procedure could equivalently be found by detecting the connected components of the subgraphs induced by $E(\vartheta) := \{e \in E : x_e^* \geq \vartheta\}$ for all possible threshold values $\vartheta \in \{x_e^* > 0 : e \in E\}$. In this view, the above heuristic is an improved version of the one used in Grötschel and Holland [398], that checks the connected components of the subgraph $G' = (V, E(\vartheta))$ for $\vartheta = \min\{x_e^* > 0 : e \in E\}$.

The above scheme can easily be adapted to deal with generalized SEC's, leading to the heuristic separation procedure called GSEC.H2 in [300].

Heuristic separation algorithms for generalized comb inequalities Two simple heuristic separation procedures for generalized comb inequalities are next described.

We first consider the generalized 2-matching constraints. Using a construction similar to that proposed by Padberg and Rao [644] for the b -matching problem, one can transform the separation problem for generalized 2-matching inequalities into a minimum capacity odd cut problem; hence this separation problem is exactly solvable in polynomial time. This task is however rather time consuming, hence the branch-and-cut code makes use of the following simple heuristic, derived from similar TSP procedures [642]. Given the fractional point (x^*, y^*) , the subgraph $\tilde{G} = (\tilde{V}, \tilde{E})$ induced by $\tilde{E} := \{e \in E : 0 < x_e^* < 1\}$ is defined. Then, each connected component H of \tilde{G} is considered, in turn, as the handle of a possibly violated generalized 2-matching inequality, whose 2-node teeth correspond to the edges $e \in \delta(H)$ with $x_e^* = 1$ (if the number of these edges is even, the inequality is clearly rejected). The procedure takes $O(n + |\tilde{E}|)$ time, if properly implemented.

The second separation procedure consists of applying the above described heuristic for generalized 2-matching inequalities after having shrunk each cluster into a single supernode, in a vein similar to that described in Padberg and Rinaldi [647].

2.5. Heuristic algorithms

A number of known tour construction and tour improvement heuristic algorithms for the TSP (see, e.g., Golden and Stewart [388] or Chapter 8) can be adapted to both GTSP and E-GTSP. We next concentrate on the heuristics producing feasible E-GTSP (and hence GTSP) solutions proposed by Fischetti, Salazar and Toth [300].

As to tour construction procedures, we describe a possible adaptation of the well-known *farthest insertion* TSP procedure; *nearest insertion* and *cheapest insertion* procedures can be adapted in a similar way. For each pair of clusters C_h and C_k , let the corresponding *distance* d_{hk} be defined as $d_{hk} := \min\{c_{ij} : i \in C_h, j \in C_k\}$. The procedure starts by choosing the two clusters, say C_a and C_b , that are farthest from each other (with respect to distances d_{hk}), and defines a partial tour T between the two closest nodes $i \in C_a$ and $j \in C_b$. At each iteration, T is enlarged by first determining the uncovered cluster C_h farthest from the clusters currently visited by T , and then by inserting a node v of C_h between two consecutive nodes i and j of T so as to minimize $c_{iv} + c_{vj} - c_{ij}$. The procedure stops when T covers all the clusters. As in the TSP case, the procedure is likely to produce better solutions when the costs satisfy the triangle inequality.

We next describe two tour improvement procedures.

The first procedure, RP1, is based on 2-opt and 3-opt exchanges. Let T be the current E-GTSP solution, visiting exactly one node for each cluster, and let $S \subseteq V$ be the set of the visited nodes. Clearly, any near-optimal TSP solution on the subgraph induced by S (found heuristically through, e.g., 2- or 3-opt exchanges) can lead to an improved GTSP solution. This approach has however the drawback of leaving the set of visited nodes unchanged. In order to remove this restriction, the following generalized 2-opt scheme has been proposed. Let $(\dots, C_\alpha, C_\beta, \dots, C_\gamma, C_\delta, \dots)$ be the cluster sequence corresponding to the current tour T . All the edges of T not incident with the nodes in $C_\alpha \cup C_\beta \cup C_\gamma \cup C_\delta$ are fixed. The scheme tries to exchange the current cluster sequence into $(\dots, C_\alpha, C_\gamma, \dots, C_\beta, C_\delta, \dots)$. To this end, two node pairs (u^*, w^*) and (v^*, z^*) are determined such that

$$\begin{aligned} c_{iu^*} + c_{u^*w^*} + c_{w^*h} &= \min\{c_{ia} + c_{ab} + c_{bh} : a \in C_\alpha, b \in C_\gamma\}, \\ c_{jv^*} + c_{v^*z^*} + c_{z^*k} &= \min\{c_{ja} + c_{ab} + c_{bk} : a \in C_\beta, b \in C_\delta\}, \end{aligned}$$

where nodes i, j, h and k are the nodes visited by T belonging to the clusters preceding C_α , following C_β , and preceding C_γ and following C_δ , respectively.

This computation requires $|C_\alpha||C_\gamma| + |C_\beta||C_\delta|$ comparisons. On the whole, trying all the possible pairs (C_α, C_β) and (C_γ, C_δ) leads to an $O(n^2)$ time complexity, since each edge of G needs to be considered only twice.

Moreover, RP1 considers a 3-opt exchange trying to modify the cluster sequence $(\dots, C_\alpha, C_\beta, C_\gamma, \dots, C_\delta, C_\varepsilon, \dots)$ into $(\dots, C_\alpha, C_\gamma, \dots, C_\delta, C_\beta, C_\varepsilon, \dots)$.

We next describe a second refinement procedure, RP2, that proved to be quite effective in our computational study. Let T be the current E-GTSP solution, and let $(C_{h_1}, \dots, C_{h_m})$ be the sequence in which T goes through the clusters. The refinement consists of finding the best feasible tour, T^* , visiting the clusters according to the given sequence. This can be done, in polynomial time, by solving $|C_{h_1}|$ shortest path problems, as described below.

We construct a layered acyclic network, LN , having $m + 1$ layers corresponding to clusters $C_{h_1}, \dots, C_{h_m}, C_{h_1}$; see Figure 2.5, where all edges are directed from left to right. LN contains all the nodes of G , plus an extra node j' for each $j \in C_{h_1}$. There is an arc (i, j) for each $i \in C_{h_t}$ and $j \in C_{h_{t+1}}$ ($t = 1, \dots, m - 1$), having cost c_{ij} . Moreover, there is an arc (i, j') for each $i \in C_{h_m}$ and $j \in C_{h_1}$, having cost c_{ij} (these arcs connect the last two layers of the network). For a given $w \in C_{h_1}$, any path in LN from w to w' visits exactly one node for each layer (cluster), hence it gives a feasible E-GTSP tour. Conversely, every E-GTSP tour visiting the clusters according to sequence $(C_{h_1}, \dots, C_{h_m})$ corresponds to a path in LN from a certain $w \in C_{h_1}$ to w' . It then follows that the best E-GTSP tour T^* visiting the clusters in the same sequence, can be found by determining the shortest path from each $w \in C_{h_1}$ to the corresponding w' . The overall time complexity is then $|C_{h_1}|O(n^2)$, i.e., $O(n^3)$ in the worst case. In practice, the time typically spent is significantly reduced by choosing C_{h_1} as the cluster with minimum cardinality, and using a shortest-path algorithm specialized for acyclic digraphs (e.g., Bang-Jensen and Gutin [84] and Cormen, Leiserson and Rivest [218]).

Notice that the above refinement procedure leads to an $O((m-1)!n^3)$ -time exact algorithm for E-GTSP, obtained by trying all the $(m-1)!$ possible cluster sequences. Therefore, E-GTSP is polynomially solvable for fixed m (independently of n).

2.6. A branch-and-cut algorithm

In this subsection we describe the enumerative algorithm for the exact solution of the problem proposed by Fischetti, Salazar and Toth in [300]. Since all the instances considered in the computational study have trian-

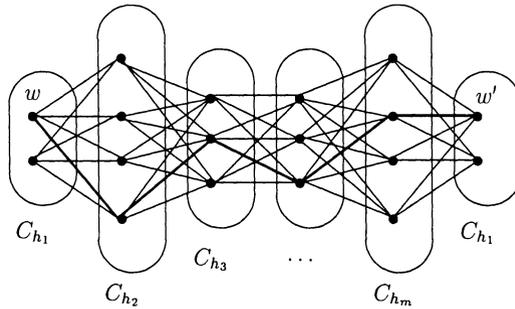


Figure 13.2. The layered network LN .

gular costs, we give a rather detailed description of the implementation of the algorithm for the E-GTSP. The algorithm can easily be adapted to the GTSP. We assume that all costs c_e are integer.

The algorithm follows a branch-and-bound scheme, in which lower bounds are computed by solving an LP relaxation of the problem. The relaxation is iteratively tightened by adding valid inequalities to the current LP, according to the so-called *cutting plane* approach. The overall method is commonly known as a *branch-and-cut* algorithm; we refer to Padberg and Rinaldi [648] and Jünger, Reinelt and Rinaldi [474] for a thorough description of the technique and to [160] for recent developments. We next describe some important implementation issues, including the best parameter setting resulting from the computational experience.

Lower bound computation At each node of the decision tree, the lower bound is computed by solving the LP problem defined by (13), (14), (19), the bound constraints on the variables, the constraints derived from branching, plus a subset of GSEC's and generalized comb inequalities. This subset initially coincides with that of the parent node (for the root node an 'ad hoc' initialization procedure, based on Lagrangian optimization, will be described later). Notice that the y variables are not projected away through equations (14), as this would result in a much denser LP coefficient matrix. Then, in an iterative way, the LP is solved, and the computation starts by retrieving the optimal LP basis of the parent node. Some inequalities that are violated by the current LP optimal solution are added. To this end, we applied in sequence the separation procedure for fan inequalities, GSEC_H2, GSEC_H1, and GSEC_SEP. All the violated constraints found (except the fan inequalities) are permanently stored in compact form in a global data structure

called the *constraint pool*. Whenever an inequality introduced in the current branch-node is slack for 5 (say) consecutive LP solutions, it is removed from the LP (but not from the pool). Moreover, whenever the LP-solver takes too long to solve the current LP, all the slack inequalities introduced in the previous nodes are removed.

LAGRANGIAN RELAXATION

At the beginning of the root node computation, Lagrangian optimization is applied with the aim of determining a good subset of constraints for the initial LP, as well as a near-optimal heuristic solution. The following (simplified) model for the E-GTSP, in which the y variables have been projected away through (14), is considered.

$$\min \sum_{e \in E} c_e x_e \tag{35}$$

subject to

$$\sum_{e \in E} x_e = m \tag{36}$$

$$\sum_{e \in \delta(C_h)} x_e = 2 \quad \text{for } h = 1, \dots, m \tag{37}$$

$$\sum_{e \in \delta(C_h) \cap \delta(v)} x_e \leq \sum_{e \in \delta(v) \setminus \delta(C_h)} x_e \quad \text{for } h = 1, \dots, m; v \in V \setminus C_h \tag{38}$$

$$\sum_{e \in E(S)} x_e \leq r - 1 \quad \text{for } \begin{matrix} S = \cup_{i=1}^r C_{l_i} \\ C_1 \subset V \setminus S \\ 2 \leq r \leq m - 2 \end{matrix} \tag{39}$$

$$x_e \in \{0, 1\} \quad \text{for } e \in E. \tag{40}$$

Equation (36) is redundant in this formulation. Inequalities (38) and (39) are fan and Basic GSEC's, respectively (notice however that not all GSEC's are included in the model).

The fan inequalities (38), plus the degree constraints (37) for $h \neq 1$, are dualized in a Lagrangian fashion. The Lagrangian relaxed problem calls for $m - 2$ edges (each connecting two different clusters) in $E \setminus \delta(C_1)$ inducing no intra-cluster cycles, plus two edges incident with C_1 . Therefore, it can be efficiently solved as follows:

- i) *shrink* G with respect to the m clusters, i.e., replace each cluster C_h with a single *super-node* h , and define for each super-node pair h, k a *super-edge* (h, k) with cost

$$\tilde{c}_{hk} := \min\{c'_{ij} : i \in C_h, j \in C_k\}, \tag{41}$$

where c'_{ij} is the Lagrangian cost of edge $(i, j) \in E$;

- ii) compute the min-cost 1-tree (Held and Karp [445]) on the shrunken graph;
- iii) obtain an optimal solution to the Lagrangian relaxed problem by replacing each super-edge (h, k) in the 1-tree found at Step ii), with its corresponding edge $(i, j) \in E$ (the one producing the minimum in (41)).

The computation of near-optimal Lagrangian multipliers is done using classical subgradient optimization. The multipliers are iteratively updated through two nested loops. In the external loop the multipliers for the fan inequalities (38) are updated. With these multipliers fixed, the internal loop adjusts the multipliers for the degree constraints (37) so as to hopefully produce a tour in the shrunken graph. This is in the spirit of the successful Held-Karp approach to the standard TSP. At the end of the internal loop, if the final 1-tree on the shrunken graph is a tour, a heuristic E-GTSP solution is determined through the refinement procedure RP2 of Section 2.5, where the cluster sequence C_{h_1}, \dots, C_{h_m} is the one induced by the tour in the shrunken graph. This approach computationally proved to be quite effective in determining near optimal solutions at the very beginning of the root node computation. At most 1000 and 50 subgradient iterations in the external and internal loops, respectively, are performed.

ROOT NODE INITIALIZATION

Let λ_h^* and μ_{hj}^* be the best Lagrangian multipliers for constraints (37) and (38), respectively. The initial LP at the root node contains constraints (2), (36), the bound restrictions on the variables, plus the subset of the fan inequalities (34) with $\mu_{hj}^* > 0$. Moreover, the LP contains the Basic GSEC's (39) that were active in the computation of the 1-tree on the shrunken graph with respect to (λ^*, μ^*) . To be more specific, the procedure includes in the LP all the constraints (39) whose subset S corresponds to a connected component detected by the Kruskal [523] algorithm used for determining the best 1-tree on the shrunken graph. With this initialization, the optimal value of the first LP relaxation is guaranteed to be at least as good as the one provided by the Lagrangian relaxation.

Upper bound computation At the root node, the farthest insertion, nearest insertion and cheapest insertion procedures are applied, each followed by the tour improvement procedures, as described in Section 2.5. Moreover, as explained above, for each tour among clusters found

during the Lagrangian relaxation a new feasible solution is obtained through procedure RP2. All solutions found are refined through the tour improvement procedures of Section 2.5.

At any branching node, the information associated with the fractional point available after each LP solution is exploited, in the attempt of improving the current *UB*. To this end, let (x^*, y^*) be the optimal LP solution. A heuristic solution is initialized by taking all the edges e with $x_e^* = 1$, and then completed through a nearest insertion scheme. Again, the resulting solution is refined through the tour improvement procedures.

Branching Two possibilities for branching are considered: branching on variables and branching on cuts. Let (x^*, y^*) be the fractional LP solution at the end of the current node.

Branching on variables (the standard approach for branch-and-cut) consists of selecting a fractional x_e^* , and generating two descendent nodes by fixing the value of x_e to either 0 or 1. As usual, x_e^* is chosen as close as possible to 0.5 (ties are broken by choosing the edge e having maximum cost c_e).

Branching on cuts consists of choosing a subset $S \subset V$ such that $\sum_{e \in \delta(S)} x_e^*$ is not an even integer, and imposing the disjunction

$$\left(\sum_{e \in \delta(S)} x_e \leq 2k \right) \text{ or } \left(\sum_{e \in \delta(S)} x_e \geq 2k + 2 \right)$$

where $k := \lfloor \sum_{e \in \delta(S)} x_e^* / 2 \rfloor$. Subset S is determined as follows.

Let v_1, \dots, v_m be the node sequence corresponding to the current best E-GTSP solution, say (\tilde{x}, \tilde{y}) , where the subscripts of v are intended to be taken as modulo m . Only a few sets S are considered, namely those obtained as the union of consecutive clusters in the sequence (i.e., of the form $S := C_{h(v_a)} \cup C_{h(v_{a+1})} \cup \dots \cup C_{h(v_b)}$ for some pair (a, b)), and such that $2 + \varepsilon \leq \sum_{e \in \delta(S)} x_e^* \leq 4 - \varepsilon$ for $\varepsilon = 0.2$. Among these sets S , if any, the one maximizing

$$L(S) := \min\{d(v_i, v_j) : i = a, a + 1, \dots, b - 1; j = b + 1, b + 2, \dots, a - 2\}$$

is chosen, where $d(v_i, v_j) := c_{v_i v_j} + c_{v_{i+1} v_{j+1}} - c_{v_i v_{i+1}} - c_{v_j v_{j+1}}$ is the additional cost corresponding to the new solution obtained from (\tilde{x}, \tilde{y}) by exchanging the edge pairs $((v_i, v_{i+1}), (v_j, v_{j+1}))$ and $((v_i, v_j), (v_{i+1}, v_{j+1}))$. $L(S)$ is an estimate on the increase of the cost of the optimal solution (and of the LP lower bound as well) when imposing $\sum_{e \in \delta(S)} x_e \geq 4$. Choosing $L(S)$ as large as possible then hopefully produces a significant increase in the lower bound of one of the two children of the current node.

In the computational study, the “branching on cuts” strategy (that turned out to be superior) was used, resorting to the “branching on variables” approach when the procedure does not find a suitable set S . Since the heuristic solutions computed at the root node are quite good, a depth-first tree search scheme was implemented (although, in general, this is not the best strategy one can choose).

2.7. Computational results

In this subsection, the computational behaviour of the branch-and-cut algorithm proposed by Fischetti, Salazar and Toth in [300] and described in Section 2.6, is analyzed. The algorithm, implemented in ANSI C, was run on a Hewlett Packard 9000 Series 700 Apollo. As to the LP solver, the package CPLEX 2.1, implementing both primal and dual Simplex algorithms, was used.

The instances of the testbed were obtained by taking all the TSP test problems from the Reinelt TSPLIB library [709] having $137 \leq n \leq 442$. The node clustering has been done so as to simulate geographical regions (using the internal costs as the metric), according to the following procedure. For a given instance, the number of clusters is given by $m := \lceil n/5 \rceil$. Then m centers are determined by considering m nodes as far as possible one from each other. The clusters are finally obtained by assigning each node to its nearest center.

In addition, for the Grötschel and Holland [398] geographical problems GR137 (America), GR202 (Europe), GR229 (Australia-Asia), and GR431 (Australia-Asia-Europe) the “natural” clustering has been considered, in which clusters correspond to countries. The resulting instances are 35GR137, 31GR202, 61GR229, and 92GR431, respectively.

Tables 13.1 and 13.2 give computational results for the above test problems. Times are given in HP 9000/720 CPU seconds. For each problem, Table 13.1 gives the following information for the root node:

Name : in the form $mXXXXn$, where m is the number of clusters, and $XXXXn$ is the name of the problem in TSPLIB (n gives the number of nodes);

Lagr-LB : percentage ratio $LB/(\text{optimal solution value})$, where LB is the lower bound value computed through the Lagrangian relaxation of Section 2.6;

Lagr-UB : percentage ratio $UB/(\text{optimal solution value})$, where UB is the upper bound value at the end of the Lagrangian relaxation (see Section 2.6);

Lagr-t : CPU time, in seconds, for the Lagrangian relaxation;

Name	Lagr-LB	Lagr-UB	Lagr-t	basic-LB	r-LB	r-UB	r-time
35gr137	84.82	100.00	8.0	86.81	99.44	100.00	31.9
31gr202	85.19	100.21	13.8	85.35	99.85	100.05	464.8
61gr229	85.26	102.39	24.8	85.42	99.81	100.87	156.8
92gr431	86.36	103.55	83.8	86.49	99.84	100.05	2256.5
28gr137	86.53	101.02	4.6	86.64	100.00	100.00	96.7
29pr144	99.64	100.00	2.3	99.82	100.00	100.00	8.0
30kroa150	84.13	100.00	7.6	84.24	100.00	100.00	100.0
30krob150	88.15	100.00	9.9	88.35	100.00	100.00	60.3
31pr152	94.91	100.00	9.6	95.14	98.45	100.00	51.4
32u159	86.84	100.00	10.9	86.97	99.96	100.00	139.6
39rat195	81.50	101.87	8.2	81.71	100.00	100.00	245.5
40d198	93.85	100.48	12.0	93.90	100.00	100.00	762.5
40kroa200	82.64	100.00	15.3	82.88	99.99	100.00	183.3
40krob200	83.29	100.05	19.1	83.47	100.00	100.00	268.0
41gr202	92.30	100.05	20.9	92.44	100.00	100.00	1021.3
45ts225	83.54	100.09	19.4	83.62	99.11	100.09	1298.4
46pr226	96.70	100.00	14.6	97.21	100.00	100.00	106.2
46gr229	89.83	100.37	49.6	89.92	99.58	100.00	995.2
53gil262	85.29	103.75	15.8	85.44	99.80	100.89	1443.5
53pr264	91.90	100.33	24.3	91.98	100.00	100.00	336.0
60pr299	84.48	100.00	33.2	84.67	100.00	100.00	811.4
64lin318	92.48	100.36	52.5	92.64	99.79	100.36	847.8
80rd400	85.28	103.16	59.8	85.52	99.94	102.97	5031.5
84fl417	93.61	100.13	77.2	93.67	100.00	100.00	16714.4
87gr431	93.46	101.18	408.3	93.49	99.94	100.54	26774.0
88pr439	92.26	101.42	146.6	92.39	100.00	100.00	5418.9
89pcb442	82.74	104.22	78.8	83.03	99.49	100.29	5353.9

Table 13.1. Root node statistics.

basic-LB : percentage ratio $LB/(\text{optimal solution value})$, where LB is the optimal value of the LP relaxation of the simplified model (35)–(40);

r-LB : percentage ratio $LB/(\text{optimal solution value})$, where LB is the final lower bound at the root node;

r-UB : percentage ratio $UB/(\text{optimal solution value})$, where UB is the final upper bound at the root node;

r-time : CPU time, in seconds, for the root node (including $Lagr-t$).

According to the table, the upper bound computed using Lagrangian relaxation is quite tight. On the other hand, the quality of the Lagrangian lower bound is rather poor, with an average gap of 11.6%. This is mainly due to the fact that it is derived from the simplified model (35)–(40). Indeed, notice that the best theoretical lower bound

Name	optval	t-time	LP-t	SEP-t	nodes	cuts	fan	GSEC	Gcomb
35gr137	28709	41.5	13.0	6.2	6	296	150	25	8
31gr202	14416	504.7	366.7	46.6	2	1112	325	709	0
61gr229	65508	215.6	109.3	28.5	4	793	355	333	2
92gr431	75616	3365.2	2342.8	278.2	4	2219	713	1172	2
28gr137	35957	96.9	65.0	6.6	0	549	237	257	0
29pr144	45886	8.2	2.5	0.1	0	209	175	7	0
30kroa150	11018	100.3	63.3	8.0	0	594	254	270	0
30krob150	12196	60.6	33.0	5.2	0	511	243	234	0
31pr152	51576	94.8	54.0	6.9	2	574	246	250	5
32u159	22664	146.4	98.0	11.2	2	599	260	288	0
39rat195	854	245.9	167.7	26.3	0	1104	395	634	0
40d198	10557	763.1	571.8	56.9	0	1189	334	750	0
40kroa200	13406	187.4	108.2	27.4	2	710	339	308	4
40krob200	13111	268.5	182.7	23.5	0	947	358	519	0
41gr202	23239	1022.2	764.1	119.3	0	1597	335	1154	0
45ts225	68340	37875.9	34071.0	2481.6	190	8590	499	3089	165
46pr226	64007	106.9	45.4	5.0	0	513	314	139	0
46gr229	71641	1187.5	870.3	132.2	2	1428	393	873	14
53gil262	1013	6624.1	5342.7	943.4	16	2676	516	1427	27
53pr264	29549	337.0	204.6	34.4	0	1016	479	407	0
60pr299	22615	812.8	583.9	43.3	0	1358	536	685	0
64lin318	20765	1671.9	1038.8	292.6	10	1680	585	867	1
80rd400	6361	7021.4	4721.7	1658.3	2	3092	762	1852	0
84fl417	9651	16719.4	12232.3	2964.2	0	5102	962	3806	0
87gr431	101523	31544.6	24873.2	4540.0	2	4354	672	2937	5
88pr439	60099	5422.8	3636.5	896.9	0	2979	778	1918	0
89pcb442	21657	58770.5	43753.5	12712.1	46	9427	949	4591	38

Table 13.2. Branch-and-cut statistics.

for the Lagrangian relaxation equals the optimal value of the LP relaxation of model (35)–(40). The latter value was computed through a simplified version of the cutting plane algorithm, and is reported in the table (column *basic-LB*). It can be seen that the improvement with respect to the Lagrangian lower bound is negligible.

Table 13.2 shows the performance of the overall enumerative algorithm. For each problem the table gives:

Name : the problem name;

optval : value of the optimal solution;

t-time : CPU time, in seconds, for the overall execution;

LP-t : overall CPU time, in seconds, spent by the LP solver;

SEP-t : overall CPU time, in seconds, spent for separation;

- nodes** : number of nodes of the branch-decision tree (=0 if no branching is required);
- cuts** : total number of cuts generated, including those found by the Lagrangian initialization (Section 2.6) and those recovered from the pool;
- fan** : total number of fan inequalities generated;
- GSEC** : total number of GSEC's found by the heuristic procedures GSEC_H1 and GSEC_H2 of Section 2.4.1.
- Gcomb** : total number of generalized comb inequalities generated.

The table shows that all the considered instances can be solved to optimality within an acceptable computing time. Moreover, a significant part of the total computing time is spent within the LP solver. In about 50% of the cases, no branching is needed. The results also show that natural clustering produces easier instances than those obtained through the clustering procedure.

As to procedure GSEC_SEP, it never found violated cuts, with the only exception of instance 45TS225 for which 9 cuts were detected. This proves the effectiveness of the heuristic separations for GSEC's. The inequalities which are most frequently recovered from the pool are the GSEC's (24).

In order to evaluate the effect of different clusterizations of the nodes, a second clustering procedure has also been considered to simulate geographical regions. Given a TSP instance, let (x_i, y_i) be the geographical coordinates of the i th node ($i = 1, \dots, n$). This information is provided in TSPLIB for all the instances considered in Table 13.3. Let $xmin$, $xmax$, $ymin$ and $ymax$ be the minimum and maximum x - and y -coordinates, respectively. The procedure considered the rectangle whose vertices have coordinates $(xmin, ymin)$, $(xmin, ymax)$, $(xmax, ymax)$, and $(xmax, ymin)$, and subdivided it so as to obtain an $NG \times NG$ grid in which each cell has edges of length $(xmax - xmin)/NG$ and $(ymax - ymin)/NG$. Each cell of the grid containing at least one node corresponds to a cluster. As to NG , it is determined so as to have a prefixed average number μ (an input parameter) of nodes in each cluster. To this end, let $CLUSTER(NG)$ be the number of nonempty clusters corresponding to the $NG \times NG$ grid, and define NG as the minimum integer such that $CLUSTER(NG) \geq n/\mu$.

Table 13.3 gives, for each test problem and value of $\mu = 3, 5, 10$, the overall CPU time (in HP 9000/720 CPU seconds), the number of nodes of the branch-decision tree, and the number m of clusters.

Comparing Table 13.3 (for $\mu = 5$) and Table 13.2 shows that the grid clusterization produces harder instances. No correlation exists, instead,

Name	$\mu = 3$			$\mu = 5$			$\mu = 10$		
	t-time	nodes	m	t-time	nodes	m	t-time	nodes	m
gr137	81.7	0	46	94.7	0	28	972.0	0	14
pr144	28.8	0	48	25.3	0	30	21.8	0	16
kroa150	56.3	2	57	72.5	0	36	111.2	0	16
krob150	40.5	0	56	100.0	2	36	79.2	0	16
pr152	68.7	4	54	455.2	42	33	35.5	0	16
u159	36.5	0	58	154.3	0	38	107.4	0	23
rat195	84.4	2	81	1409.5	18	49	423.5	0	25
d198	539.5	0	67	591.2	0	40	2849.9	0	25
kroa200	207.2	2	72	1696.3	30	47	339.5	0	25
krob200	2031.9	54	76	119.1	0	48	422.2	0	25
gr202	2644.3	34	73	727.4	2	43	450.1	0	21
ts225	453.4	14	75	–	–	45	10601.5	0	25
pr226	130.7	0	78	65.6	0	50	105.7	0	24
gr229	312.0	0	80	1895.1	8	46	9391.9	2	23
gil262	5674.6	104	96	10763.1	42	63	1141.0	0	36
pr264	747.1	16	101	109.6	0	55	376.8	0	27
pr299	761.3	2	102	4629.9	12	69	2730.6	0	35
lin318	37117.4	220	108	16784.8	40	64	71010.7	26	36
rd400	21764.6	118	135	87308.1	198	81	21156.8	2	49
fl417	7687.9	0	142	10373.7	0	93	919.2	0	43
pr439	1905.7	6	163	18876.5	14	96	35652.6	8	48
pcb442	23226.1	86	155	39155.3	24	96	15266.6	0	48

Problem ts225 with $\mu = 5$ required more than 100,000 CPU seconds.

Table 13.3. Some computational results with different clusterizations.

between the difficulty of the problem and the average number of nodes in each cluster.

On the whole, the computational performances of the branch-and-cut algorithm are quite satisfactory for our families of instances. All the test problems in the test bed were solved to optimality within acceptable computing time, with the only exception of problem TS225 with grid clusterization (case $\mu = 5$ of Table 13.3). Moreover, the heuristic algorithms proposed allow one to compute very good solutions within short computing time. As shown in Table 13.1, after the Lagrangian phase the average percentage error with respect to the optimum is 0.9% (see column *Lagr-UB*), and 0.2% at the end of the root node (see column *r-UB*).

3. The Orienteering Problem

As stated in the introduction, we are given a set of n nodes, each having an associated nonnegative prize p_v , and a distinguished “depot” node, say node 1. Let $t_{(i,j)}$ be the *time* spent for routing nodes i and j in

sequence. The *Orienteering Problem* (OP) is to find a cycle C through node 1 whose total duration $t(C)$ does not exceed a given bound t_0 , and visiting a node subset with a maximum total prize. Without loss of generality, we can assume that cycle C contains at least three nodes.

The problem can be formulated as

$$v(\text{OP}) \equiv \max \sum_{v \in V} p_v y_v \tag{42}$$

subject to

$$\sum_{e \in E} t_e x_e \leq t_0, \tag{43}$$

$$x(\delta(v)) = 2 y_v \quad \text{for } v \in V, \tag{44}$$

$$x(\delta(S)) \geq 2 y_v \quad \text{for } S \subset V, 1 \in S, v \in V \setminus S, \tag{45}$$

$$y_1 = 1, \tag{46}$$

$$x_e \in \{0, 1\} \quad \text{for } e \in E, \tag{47}$$

$$y_v \in \{0, 1\} \quad \text{for } v \in V \setminus \{1\}, \tag{48}$$

Because of the degree constraints (44), inequalities (45) can equivalently be written as

$$x(E(S)) \leq y(S) - y_v \quad \text{for } S \subset V, 1 \in S, v \in V \setminus S \tag{49}$$

and

$$x(E(\bar{S})) \leq y(\bar{S}) - y_v \quad \text{for } \bar{S} \subset V, 1 \in V \setminus \bar{S}, v \in \bar{S}. \tag{50}$$

Notice that the inequalities (16), although valid, are dominated by (45) as $y_i - 1 \leq 0$ for all $i \in V$.

This section is mainly based on the results given by Fischetti, Salazar and Toth in [301]. Section 3.1 discusses a number of additional constraints, which improve the quality of the LP relaxation of the basic model. We also analyze a family of *conditional cuts*, i.e., cuts which cut off the current optimal solution. Separation procedures are described in Section 3.2, whereas Section 3.3 presents heuristic algorithms for finding approximate OP solutions. An overall branch-and-cut algorithm is described in Section 3.4. In that section, an effective way of integrating conditional cuts within the overall framework is also presented. Extensive computational results on several classes of test problems involving up to 500 nodes are presented in Section 3.5.

3.1. Additional inequalities

In this section we describe five classes of additional inequalities for OP. These inequalities are capable of strengthening the LP-relaxation

of model (42)–(48). The first two classes do not rely on the total time restriction (43), and are derived from the *cycle relaxation* of OP [79, 92]. The remaining classes, instead, do exploit the total time restriction.

A polyhedral analysis of the OP appears very difficult, and to our knowledge it has not been addressed in the literature. From the practical point of view, however, these cuts proved to be of fundamental importance for the solution of most instances.

Logical constraints Clearly, $x_e = 1$ for some $e \in \delta(j)$ implies $y_j = 1$. Hence the *logical constraints*

$$x_e \leq y_j \text{ for all } e \in \delta(j), j \in V \setminus \{1\} \quad (51)$$

are valid for OP. Whenever $e = (v, j) \notin \delta(1)$, inequality (51) is a particular case of (50) arising for $\bar{S} = \{v, j\}$. On the other hand, for $e \in \delta(1)$ these inequalities do improve the LP relaxation of model (42)–(48). To see this, consider the fractional point (x^*, y^*) with $x_{12}^* = x_{13}^* = 1$, $y_1^* = 1$, $y_2^* = y_3^* = 1/2$ (all other components being 0). Assuming $t_{12} + t_{13} \leq t_0$, this point satisfies all the constraints of the relaxation, but not the constraints (51) associated with $e = (1, 2)$ and $j = 2$, and with $e = (1, 3)$ and $j = 3$.

We observe that the addition of (51) to model (42)–(48) makes the integrality requirement on the y -variables redundant. Indeed, let (x^*, y^*) be any point satisfying (43)–(47) and $0 \leq y_v \leq 1$ for all $v \in V$, and define $T^* := \{e \in E : x_e^* = 1\}$. Then from (44) we have $y_v = |T^* \cap \delta(v)|/2$ for all $v \in V$, i.e., $y_v \in \{0, 1/2, 1\}$. But $y_v = 1/2$ would imply $T^* \cap \delta(v) = \{e\}$ for some $e \in \delta(v)$, which is impossible since in this case the corresponding logical constraint (51) would be violated.

2-matching inequalities The well-known *2-matching constraints* for the TSP have the following counterpart in the cycle relaxation of OP:

$$x(E(H)) + x(T) \leq y(H) + \frac{|T| - 1}{2}, \quad (52)$$

where $H \subset V$ is called the *handle*, and $T \subset \delta(H)$ is a set with $|T| \geq 3$, $|T|$ odd, pairwise disjoint *teeth*. This inequality is obtained by adding up the degree constraints for all $v \in H$ and the bound constraints $x_e \leq 1$ for all $e \in T$, dividing by 2, and then rounding down all the coefficients to the nearest integer.

Cover inequalities The total time constraint (43), along with the requirements $x_e \in \{0, 1\}$ for $e \in E$, defines an instance of the *0-1 Knapsack Problem* (KP), in which items correspond to edges. Therefore, every

valid KP inequality can be used to hopefully improve the LP relaxation of the OP model. Among the several classes of known KP inequalities, let us consider the *cover inequality* (see, e.g., Nemhauser and Wolsey [625]):

$$x(T) \leq |T| - 1, \tag{53}$$

where $T \subseteq E$ is an inclusion-minimal edge subset with $\sum_{e \in T} t_e > t_0$. This constraint stipulates that not all the edges of T can be selected in a feasible OP solution.

A cover inequality can in some cases be strengthened. In particular, one can easily obtain the valid *extended inequality*

$$x(T \cup Q) \leq |T| - 1, \tag{54}$$

where $Q := \{e \in E \setminus T : t_e \geq \max_{f \in T} t_f\}$.

A different improvement is next proposed, which exploits the fact that the selected edges have to define a cycle. The improvement can only be applied in case T defines an infeasible cycle passing through node 1, and leads to the *cycle cover inequality*:

$$x(T) \leq y(V(T)) - 1. \tag{55}$$

Validity of (55) follows from the easy observation that $x(T) \geq y(V(T))$ would imply $x_e = 1$ for all $e \in T$. Figure 3.1 shows a fractional point violating a cycle cover inequality but not other previous inequalities. More generally, (55) is a valid inequality whenever T does not contain any feasible cycle. This generalization will be studied in the forthcoming subsection on conditional cuts.

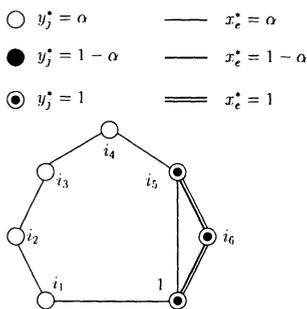


Figure 3.3. Fractional point violating a cycle cover inequality for the OP instance with $t_0=6$ and $t_e=1$ for all $e \in E$ ($0 < \alpha \leq 1/2$). Here $T = \{(1, i_1), (i_1, i_2), \dots, (i_6, 1)\}$, $x(T) = 2 + 5\alpha$, and $y(V(T)) = 3 + 4\alpha$.

Path inequalities The previous classes of additional inequalities (except the cycle cover inequalities) are based either on the cycle or on the knapsack relaxation of the problem. We next introduce a new family of constraints that exploit both relaxations.

Let $P = \{(i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k)\}$ be any simple path through $V(P) = \{i_1, \dots, i_k\} \subseteq V \setminus \{1\}$, and define the nodeset:

$$W(P) := \{v \in V \setminus V(P) : P \cup \{(i_k, v)\} \text{ can be part of a feasible OP sol.}\}.$$

We allow P to be infeasible, in which case $W(P) = \emptyset$. Then the following *path inequality*

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} - \sum_{j=2}^{k-1} y_{i_j} - \sum_{v \in W(P)} x_{i_k v} \leq 0 \tag{56}$$

is valid for OP. Indeed, suppose there exists a feasible OP solution (x^*, y^*) violating (56). Then

$$x_{i_1 i_2}^* + (x_{i_2 i_3}^* - y_{i_2}^*) + \dots + (x_{i_{k-1} i_k}^* - y_{i_{k-1}}^*) - \sum_{v \in W(P)} x_{i_k v}^* \geq 1,$$

where $x_{i_j i_{j+1}}^* - y_{i_j}^* \leq 0$ for all $j = 2, \dots, k - 1$. It then follows that $x_{i_1 i_2}^* = 1$ (hence $y_{i_2}^* = 1$), $x_{i_2 i_3}^* - y_{i_2}^* = 0$ (hence $x_{i_2 i_3}^* = 1$ and $y_{i_3}^* = 1$), \dots , $x_{i_{k-1} i_k}^* - y_{i_{k-1}}^* = 0$ (hence $x_{i_{k-1} i_k}^* = 1$), and $x_{i_k v}^* = 0$ for all $v \in W(P)$. But then solution (x^*, y^*) cannot be feasible, since it contains all the edges of P , plus an edge (i_k, w) with $w \notin W(P)$.

Figure 13.3 shows a typical fractional point that is cut off by a path inequality. This point can be viewed as the convex combination of two cycles, one of which is infeasible because of the total time requirement.

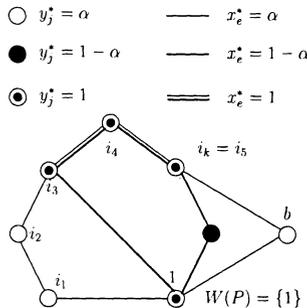


Figure 13.4. A fractional point violating a path inequality for the OP instance with $t_0=6$ and $t_e=1$ for all $e \in E$ ($0 < \alpha \leq 1/2$).

Recall that, for any given $F \subseteq E$, $t(F)$ stands for $\sum_{e \in F} t_e$. The definition of $W(P)$ amounts to checking for each $v \in V \setminus V(P)$ whether there exists a cycle of the form $C = P_1 \cup (P \cup \{i_k, v\}) \cup P_2$, where P_1 and P_2 are node-disjoint paths from 1 to i_1 and v , respectively, such that $t(P_1) + t(P) + t_{i_k v} + t(P_2) \leq t_0$. A simpler condition (producing a possibly larger set $W(P)$, and hence a weakened inequality (56)) is obtained by removing the requirement that P_1 and P_2 share no node (except node 1). This leads to the alternative definition of $W(P)$ as

$$W(P) := \{v \in V \setminus V(P) : d(1, i_1) + t(P) + t_{i_k v} + d(1, v) \leq t_0\}, \quad (57)$$

where for each $j \in V \setminus \{1\}$, $d(1, j)$ gives the total time associated with the shortest path from node 1 to node j .

Conditional cuts We next address inequalities that are not guaranteed to be valid for our problem, but can nevertheless be used in a cutting plane context.

Suppose that a heuristic OP solution of value (say) LB is available. In the process of finding an optimal OP solution we are clearly interested in finding, if any, a feasible solution of value strictly better than LB. Therefore, any inequality can be exploited as a cutting plane, provided that it is satisfied by every feasible OP solution of value greater than LB. These inequalities are called *conditional cuts*.

Let us consider a general family of inequalities of the type

$$x(T) \leq y(V(T)) - 1, \quad (58)$$

where $T \subset E$ is chosen in an appropriate way. It can be seen easily that $x(T) \leq y(V(T))$ holds for every feasible solution, no matter how T is chosen. Moreover, $x(T) = y(V(T))$ implies that the OP solution consists of a cycle entirely contained in T . It then follows that (58) can be used as a conditional cut, provided that no feasible OP solution of value strictly greater than LB is contained in T . This occurs, in particular, when

$$T = E(S) \text{ for some } S \subset V \text{ such that } 1 \in S \text{ and } \sum_{v \in S} p_v \leq \text{LB}. \quad (59)$$

A different approach for defining conditional cuts, based on enumeration, will be described in the following section.

3.2. Separation algorithms

In this section we outline exact and/or heuristic algorithms, proposed by Fischetti, Salazar and Toth [301], for the following *separation problem*: Let \mathcal{F} be one of the families of OP inequalities described in Section

3.1; given a point $(x^*, y^*) \in [0, 1]^{E \times V}$ which satisfies (43)–(44), find a member $\alpha x + \beta y \leq \gamma$ of \mathcal{F} which is (mostly) violated by (x^*, y^*) , if any.

We denote by $G^* = (V^*, E^*)$ the *support graph* associated with the given (x^*, y^*) , where $V^* := \{v \in V : y_v^* > 0\}$ and $E^* := \{e \in E : x_e^* > 0\}$.

Cut inequalities (45) Let x_e^* be viewed as a capacity value associated with each edge $e \in E^*$. For any fixed node $v \in V^* \setminus \{1\}$, a most violated inequality (45) (among those for the given v) is determined by finding a minimum-capacity $(1, v)$ -cut, say $(S_v, V^* \setminus S_v)$, on G^* . This requires $O(|V^*|^3)$ time, in the worst case, through a max-flow algorithm. Trying all possible $v \in V^* \setminus \{1\}$ then leads to an overall $O(|V^*|^4)$ -time separation algorithm.

The nodes v are considered in decreasing order of the associated y_v^* . Whenever a violated inequality (45) is found for (say) the pair S_v and v , the capacity x_{1v}^* is increased by the quantity $2 - x^*(\delta(S_v))$. This prevents the cut $(S_v, V^* \setminus S_v)$ from being generated again in the subsequent iterations. Moreover, in order to increase the number of violated inequalities detected by a single max-flow computation, two minimum-capacity $(1, v)$ -cuts are considered for each v , namely $(S'_v, V^* \setminus S'_v)$ and $(V^* \setminus S_v, S_v)$, where S_v (respectively, S'_v) contains the nodes connected to node v (respectively, node 1) in the incremental graph corresponding to the maximum flow vector. Nodeset S_v gives an hopefully violated inequality (45), whereas S'_v is used, as explained later, for producing a conditional cut.

Logical constraints (51) This family can be dealt with by complete enumeration, with an overall $O(|E^*|)$ time complexity.

2-matching constraints (52) These inequalities can be separated in polynomial time through a simple modification of the Padberg and Rao [644] odd-cut separation scheme. In order to reduce the computational effort spent in the separation, however, the following simple heuristic can be implemented. Values x_e^* are interpreted as weights associated with the edges. The greedy algorithm of Kruskal is applied to find a minimum-weight spanning tree on G^* . At each iteration in which this algorithm selects a new edge e , the connected component which contains e , say H , is determined (in the subgraph of G^* induced by all the edges selected so far). The nodeset H is then considered as the handle of an hopefully violated 2-matching constraint. In this way, the procedure generates efficiently all the connected components H of the subgraph $G_\theta = (V, E_\theta)$ induced by $E_\rho := \{e \in E : 0 < x_e^* < \theta\}$ for every

possible threshold θ . These sets H have high probability of being the handle of a violated 2-matching constraint, if one exists. For any H , tooth edges are determined, in an optimal way, through the following greedy procedure. Let $\delta(H) = \{e_1, \dots, e_p\}$ with $x_{e_1}^* \geq x_{e_2}^* \geq \dots \geq x_{e_p}^*$. The requirement that the teeth have to be pairwise disjoint is initially relaxed. For any given $|T| \geq 3$ and odd, the best choice for T consists of the edges $e_1, \dots, e_{|T|}$. Therefore, a most violated inequality corresponds to the choice of the odd integer $|T| \geq 3$ which maximizes $x_{e_1}^* + (x_{e_2}^* + x_{e_3}^* - 1) + \dots + (x_{e_{|T|-1}}^* + x_{e_{|T|}}^* - 1)$. If no violated cut can be produced in this way, then clearly no violated 2-matching constraint exists for the given handle. Otherwise a violated 2-matching constraint exists in which two tooth edges, say e and f , may overlap in a node, say v . In this case, the inequality is *simplified* by defining a new handle-tooth pair (H', T') with $T' := T \setminus \{e, f\}$, and $H' := H \setminus \{v\}$ (if $v \in H$) or $H' := H \cup \{v\}$ (if $v \notin H$). It is then easy to see that the inequality (52) associated with this new pair (H', T') is at least as violated as that associated with the original pair (H, T) . Indeed, replacing (H, T) with (H', T') increases the violation by, at least, $1 + y_v - x(\delta(v)) \geq 2y_v - x(\delta(v)) = 0$ (if $v \in H$), or $1 - y_v \geq 0$ (if $v \notin H$). By iterating this simplification step one can then always detect a violated 2-matching constraint with non overlapping teeth. In some cases this procedure could even lead to a 2-matching constraint with $|T| = 1$; if this occurs, the inequality is rejected in favour of an inequality (45) associated with the handle.

Path inequalities (56) Let us assume that the fractional point (x^*, y^*) satisfies all logical constraints (51), and observe that the path inequality associated with a given path P cannot be violated by (x^*, y^*) if $x_{i_h i_{h+1}}^* = 0$ for some $(i_h, i_{h+1}) \in P$. This follows from the fact that (56) can be rewritten as

$$\sum_{j=1}^{h-1} (x_{i_j i_{j+1}} - y_{i_{j+1}}) + x_{i_h i_{h+1}} + \sum_{j=h+1}^{k-1} (x_{i_j i_{j+1}} - y_{i_j}) - \sum_{v \in W(P)} x_{i_k v} \leq 0$$

where all terms involved in the first two summations are nonpositive by assumption. Hence every violated path inequality must be associated with a path P contained in the support graph G^* . Since this graph is usually very sparse, a simple enumeration scheme can be implemented to detect the path P producing a most violated path inequality. The procedure starts with an empty node sequence P . Then, iteratively, the current P is extended in any possible way, and the associated path inequality is checked for violation. Whenever for the current path $P =$

$$\{(i_1, i_2), \dots, (i_{k-1}, i_k)\}$$

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} - \sum_{j=2}^{k-1} y_{i_j} \leq 0,$$

holds, a backtracking step is performed, since no extension of P can lead to a violated cut.

Cover inequalities (54)–(55) We first address the separation problem for cover inequalities, in their weakest form (53), which calls for an edgese T with $\sum_{e \in T} t_e > t_0$ which maximizes $x^*(T) - |T| + 1$. It is well known that this problem can be formulated as

$$\sigma^* := \min \sum_{e \in E} (1 - x_e^*) z_e \quad (60)$$

subject to

$$\sum_{e \in E} t_e z_e \geq t_0 + 1, \quad (61)$$

$$z_e \in \{0, 1\} \quad \text{for } e \in E. \quad (62)$$

Although \mathcal{NP} -hard, the knapsack problem (60)–(62) can typically be solved within short computing time by means of specialized codes (see Martello and Toth [586]). Moreover, all variables z_e with $x_e^* = 0$ can be fixed to 0, because $z_e = 1$ would imply $\sigma^* \geq 1$. Analogously, one can set $z_e = 1$ whenever $x_e^* = 1$, since in this case its weight in (60) vanishes.

If $\sigma^* \geq 1$ then no violated cover inequality (53) exists. Otherwise, $T := \{e \in E : z_e = 1\}$ gives a most violated such cut. In both cases, it is worth checking the extended cover inequalities (54) associated with T for violation. Notice that, because of the fact that some weights in (60) can be zero, the edgese T which gives the optimum in (60) is not guaranteed to be minimal with respect to property (61). Therefore, in order to have a stronger inequality one can make T minimal (in a greedy way) before checking the extended inequality (54) for violation.

A heuristic separation algorithm for the cycle cover inequalities (55), associated with an infeasible cycle T , is now outlined. The heuristic is intended to produce several candidate cycles T with large value of $x^*(T)$. To this end, the values x_e^* are interpreted as weights associated with the edges, and a maximum-weight spanning tree on G is computed. The edges $e \in E^*$ not in the tree are then consider, in turn: if the addition of e to the tree induces a cycle T passing through node 1 and such that $\sum_{e \in T} t_e > t_0$, then a valid inequality (55) is obtained, that is checked for violation.

Conditional cuts (58) Two heuristic separation procedures for conditional cuts have been implemented. Let LB be the value of the current best solution available.

The first procedure is based on condition (59), and is embedded within the max-flow separation algorithm for inequalities (45) described earlier. For each set S'_v therein detected which satisfies $\sum_{v \in S'_v} p_v \leq LB$ and $1 \in S'_v$, the procedure sets $T = E(S'_v)$ and checks (58) for violation.

The second procedure is based on the observation that (58) can *always* be used as a conditional cut, provided that the lower bound value LB is updated by taking into account all the feasible OP solutions entirely contained in T . This amounts to computing

$$LB := \max\{LB, v(OP_T)\},$$

where $v(OP_T)$ is the optimal OP value when $x_e = 0$ is imposed for all $e \in E \setminus T$. Although the computation of $v(OP_T)$ requires exponential time in the worst case, for a sufficiently sparse edge set T it is likely that even a simple complete enumeration scheme can succeed in determining $v(OP_T)$ within short computing time. The procedure defines $T := E^*$, hence ensuring that the corresponding conditional cut (58) is violated since $x^*(T) = x^*(E)$ and $y^*(V(T)) = y^*(V)$, where $x^*(E) = y^*(V)$ because of the degree equations (44). A simple algorithm for solving OP, based on complete enumeration, is then applied on the support graph G^* . If the enumeration ends within a fixed time-limit TL then, after the updating of LB, (58) is guaranteed to be a valid conditional cut to be added to the current LP.

3.3. Heuristic algorithms

The performance of the enumerative exact algorithms improves if one is capable of early detecting “good” feasible OP solutions. To this end, Fischetti, Salazar and Toth [301] proposed the following heuristic procedure, working in two stages. In the first stage, a feasible cycle C is detected, which is likely to contain a large number of edges belonging to an optimal solution. In the second stage, refining procedures are applied to derive from C a better feasible circuit. The method is along the same lines as the heuristic proposed by Ramesh and Brown [692], but uses LP information to guide the search. A brief outline follows.

On input of the first stage, the heuristic receives, for each edge $e \in E$, an estimate w_e , $0 \leq w_e \leq 1$, of the probability of having edge e in an optimal solution. The computation of values w_e is described in Section 3.4. The edges are sorted in decreasing order of w_e , with ties broken so as to rank edges with smaller time t_e first. Then an edge subset T

containing a family of node-disjoint paths with large probability of being part of an optimal solution, is heuristically detected in a greedy way. To be specific, the procedure initializes $T := \emptyset$ and then considers, in turn, each edge e according to the given order: If $T \cup \{e\}$ contains a node with degree larger than 2, the edge is rejected; otherwise T is updated as $T := T \cup \{e\}$ if $T \cup \{e\}$ is cycle-free, else the algorithm is stopped.

Starting with T , the required feasible cycle C is obtained by means of the following steps. First, all the nodes in $V \setminus \{1\}$ that are not covered by T are removed from the graph. Then, the paths of T are linked into a cycle, C , passing through node 1. To this end, a simple nearest-neighbor scheme is applied which starts from node 1, and iteratively moves to the nearest uncovered extreme node of a path in T . At the end of this phase, a check on $\sum_{e \in C} t_e \leq t_0$ is performed. If the condition is not satisfied, the following procedure is applied to make C feasible. For any given node v covered by C , let i_v and j_v denote the two neighbors of v in C . The procedure iteratively removes a node v from C , i.e., replaces (i_v, v) and (j_v, v) with the short-cut (i_v, j_v) . At each iteration v is chosen (if possible) as a minimum-prize node whose removal makes C feasible, or else as a node that minimizes the score $p_v / (t_{i_v v} + t_{j_v v} - t_{i_v j_v})$.

In the second stage of the heuristic, the procedure receives as input the feasible cycle C computed in the first stage, and iteratively tries to improve it. At each iteration, 2-optimality edge exchanges inside C are first performed, so as to hopefully reduce its total time. Then an attempt is performed to add to C a maximum-prize node belonging to the set $Q(C)$ containing the nodes v not covered by C , and such that $\min_{(i,j) \in C} \{t_{iv} + t_{jv} - t_{ij}\} \leq t_0 - \sum_{e \in C} t_e$. If $Q(C) \neq \emptyset$, the node insertion is performed and the step is repeated. Otherwise, the whole procedure is re-applied on the cycle obtained from C by removing, in turn, one of its nodes.

3.4. A branch-and-cut algorithm

We next outline the main ingredients of the branch-and-cut algorithm proposed by Fischetti, Salazar and Toth [301] for the optimal solution of OP.

The initialization phase At the root node of the branch-decision tree, a lower bound on the optimal OP value is computed through the heuristic algorithm of Section 3.3, with edge weights $w_e = 0$ for all $e \in E$. In addition, the first Linear Program (LP) to be solved is set-up by taking:

- 1 all variables $y_v, v \in V$;

- 2 the variables x_e associated with edges belonging to the initial heuristic solution;
- 3 for all $v \in V$, the variables x_e associated with the 5 smallest-time edges $e \in \delta(v)$;
- 4 the total time constraint (43);
- 5 the n degree equations (44);
- 6 the lower and upper bounds on the variables.

Finally, the constraint pool (i.e., the data structure used to save the OP constraints that are not included in the current LP) is initialized as empty.

The cutting plane phase At each node of the branch-decision tree, the procedure determines the optimal primal and dual solutions of the current LP, say (x^*, y^*) and u^* , respectively —in case the current LP reveals infeasible, it introduces artificial variables with very large negative prize. Notice that the value of the primal solution, namely $\sum_{v \in V} p_v y_v^*$, is not guaranteed to give an upper bound on the optimal OP value, as the current LP contains only a subset of the x -variables. Then the so-called *pricing phase* is entered, in which the dual solution u^* is used to compute the reduced cost \bar{c}_e of the variables x_e that are not part of the current LP, which are by default set to 0. The variables x_e which price-out with the wrong sign (i.e., $\bar{c}_e > 0$), are added to the LP, which is then re-optimized with the primal simplex algorithm. In order to keep the size of the LP as small as possible, the procedure never adds more than 100 variables at each round of pricing (chosen among those with largest reduced costs). The pricing loop is iterated until all variables price-out correctly, i.e., until the current LP value, say UB, is guaranteed to be an upper bound on the optimal OP value. In this case, if the current node is not fathomed the following purging phase is entered. Let LB denote the value of the best OP solution known so far. The variables x_e with $\lfloor \text{UB} + 4\bar{c}_e \rfloor \leq \text{LB}$, along with the constraints that have been slack in the last 5 iterations, or whose slack exceeds 0.01, are removed from the current LP. Moreover, at the root branch-decision node, all the variables x_e with $\lfloor \text{UB} + \bar{c}_e \rfloor \leq \text{LB}$ are fixed to 0, and all the variables with $\lfloor \text{UB} - \bar{c}_e \rfloor \leq \text{LB}$ are fixed to 1 (this latter condition may only apply to LP variables at their upper bound).

The *separation phase* is next entered, in which constraints violated by (x^*, y^*) are identified and added to the current LP. The separation algorithms described in Section 3.2 are applied. The constraint pool is first searched. Then the procedure checks, in sequence, the logical constraints (51), the inequalities (45), the 2-matching constraints (52), the

cover inequalities (54)–(55), the path inequalities (56), and the conditional cuts (58). The time limit TL used for the enumeration required in the conditional cut separation, is set to $5 \cdot \text{TS}$, where TS is the computing time so far spent in the last round of separation. Whenever a separation procedure succeeds in finding violated cuts, the separation sequence is stopped, and all the cuts found are added to the LP.

In order to reduce tailing off phenomena, a branching is performed whenever the upper bound did not improve by at least 0.001 in the last 10 cutting-plane iterations of the current branching node.

At every fifth application of the separation algorithm, an attempt is performed to improve the current best OP solution through the heuristic algorithm described in Section 3.3. The values w_e required by the algorithm are set to x_e^* for all $e \in E$. This choice computationally proved very effective and typically produces very tight approximate solutions. An additional heuristic is embedded within the second separation procedure for conditional cuts (58), as described in Section 3.2. Indeed, the enumeration of the OP solutions contained in the support graph of x^* , therein required, can in some cases improve the current LB.

The branching step Whenever a branch-decision node cannot be fathomed, a branching step is performed, in a traditional way, by fixing $x_f = 0$ or $x_f = 1$ for a variable x_f chosen as follows. The 15 fractional variables x_e with x_e^* closest to 0.5 are selected. For each such candidate branching variable x_e , two values, say UB_e^0 and UB_e^1 , are computed by solving the current LP amended by the additional constraint $x_e = 0$ and $x_e = 1$, respectively. Then, the actual branching variable x_f is chosen as the one that maximizes the score $0.75 \cdot \text{UB}_e^0 + 0.25 \cdot \text{UB}_e^1$.

The overall algorithm At the root node of the branch-decision tree, the initialization phase and the cutting-plane phase are executed. When all separation algorithms fail and the current node is not fathomed, a branching step is performed. However, for the root node only, the following alternative scheme is executed.

According to computational experience, the conditional cut associated with the support graph $G^* = (V(E^*), E^*)$ of the current LP solution (x^*, y^*) , namely

$$x(E^*) \leq y(V(E^*)) - 1, \quad (63)$$

is quite effective in closing the integrality gap. Unfortunately, for rather dense G^* the simple enumeration scheme described in Section 3.2 is unlikely to complete the enumeration of all possible OP solutions contained in G^* , within the short time limit allowed. Nevertheless, cut (63) is added to the LP even when this enumeration fails (in this case the

cut is called a *branch cover cut*). This choice may however cut off the optimal OP solution as well, if this solution is contained in G^* . This possibility can be taken into account by storing the graph G^* , with the aim of dealing with it at a later time. With the branch cover cut added to the LP, the cutting plane phase is then re-entered until again all separations fail. Then, if needed, the whole scheme is iterated: the procedure adds a new branch cover cut, stores the current support graph G^* , and re-enters the cutting plane phase.

In this way, a sequence of support graphs, say $G_i^* = (V(E_i^*), E_i^*)$ for $i = 1, \dots, k$, are produced and stored until the root node is fathomed. At this point, the computation is not over, as it is necessary to consider the best OP solution within each graph G_1^*, \dots, G_k^* or, alternatively, within the “union” of these graphs, defined as $\tilde{G} = (V(\tilde{E}), \tilde{E} := \cup_{i=1}^k E_i^*)$. To this end, all the branch cover cuts are removed from the constraint pool, and the branch-and-cut algorithm is re-applied on the OP instance associated with \tilde{G} . In order to guarantee the convergence of the overall algorithm, the generation of branch cover cuts is inhibited in this second branch-and-cut round.

As explained, the branch-and-cut scheme works in two stages. In the first stage branching is avoided by adding branch cover cuts. In the second stage, a sparse graph \tilde{G} (resulting from the branch cover cuts produced in the first stage) is considered, and a classical branching strategy is used to close the integrality gap. The computational experience shows that the overall scheme typically performs better than (although does not dominate) the classical one. Indeed, the second stage takes advantage from a large number of relevant cuts (produced in the first stage and stored in the constraint pool), as well as from a very tight approximate OP solution. On the other hand, for some instances the first stage exhibits a slow convergence in the last iterations, due to tailing-off phenomena. To contrast this behavior, branching is allowed even in the first stage. Namely, at each node a branching step is performed after the addition of 5 branch cover cuts.

3.5. Computational results

The branch-and-cut algorithm proposed by Fischetti, Salazar and Toth [301], and described in the previous section (called FST in the sequel) was implemented in ANSI C language, and run on an Hewlett Packard Apollo 9000/720 computer. CPLEX 3.0 was used as LP solver. Four different classes of test problems are considered. The reader is referred to [301] for more computational results.

The first problem class (Class I) includes 15 instances from the OP and Vehicle Routing Problem (VRP) literature. Problems OP21, OP32, and OP33 are OP instances introduced by Tsiligirides [796], with travel times multiplied by 100 and then rounded to the nearest integer. Problems ATT48, EIL30, EIL31, EIL33, EIL51, EIL76, EIL101, and GIL262 are VRP instances taken from library TSPLIB 2.1 of Reinelt [709]. Problems CMT101, CMT121, CMT151, and CMT200 are VRP instances from Christofides, Mingozzi and Toth [191]. For all the VRP instances, the customer demands are interpreted as node prizes.

The second problem class (Class II) includes all the TSP instances contained in TSPLIB 2.1 involving from 137 to 400 nodes (problems GR137 to RD400). For these instances, the node prizes p_j , for $j \in V \setminus \{1\}$, have been generated in three different ways:

- Generation 1: $p_j := 1$;
- Generation 2: $p_j := 1 + (7141 \cdot (j - 1) + 73) \bmod (99)$;
- Generation 3: $p_j := 1 + \lfloor 99 \cdot t_{1j} / \theta \rfloor$, where $\theta := \max_{i \in V \setminus \{1\}} t_{1i}$.

(The above is an *errata corrige* of the prize definition for Generation 2 given in [301] which was pointed out to be incorrect by Fink, Schneiderei and Voss [290].)

Generation 1 produces OP instances in which the goal is to cover as many nodes as possible, as occurs in some applications. Generation 2 is intended to produce pseudo-random prizes in range $[1,100]$, whereas Generation 3 leads to more difficult instances, in which large prizes are assigned to the nodes far away from the depot.

For the third problem class (Class III), random instances have been obtained by using the original Laporte and Martello [536] code. In this class, both prizes and travel times are generated as uniformly random integers in range $[1,100]$, with travel times triangularized through shortest path computation.

For all problem classes, the maximum total travel time t_0 is defined as $\lceil \alpha \cdot v(\text{TSP}) \rceil$, where $v(\text{TSP})$ is the length of the corresponding shortest Hamiltonian tour, and α is a given parameter. For all instances taken from TSPLIB, the value $v(\text{TSP})$ is provided within the library. For problems OP21, OP32, OP33, CMT101, CMT121, CMT151, and CMT200, respectively, the following values for $v(\text{TSP})$ have been used: 4598, 8254, 9755, 505, 545, 699, and 764. As to the random problems of Class III, the approximate value computed by the original Laporte-Martello code has been used, namely $v(\text{TSP}) := \lfloor 0.95 \cdot UB(\text{TSP}) + 0.5 \rfloor$, where $UB(\text{TSP})$ is the length of the tour obtained by the heuristic algorithm proposed by Rosenkrantz, Stearns and Lewis [730].

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	optval	%-vis	t-time
op21	2299	0.7	0.0	0.0	1	58	205	61.9	0.7
op32	4127	1.3	0.0	0.0	1	91	160	56.2	1.3
op33	4878	1.8	0.0	0.0	1	109	500	63.6	1.8
att48	5314	0.8	0.0	0.0	1	55	30	64.6	0.8
eil30	191	5.2	0.0	0.0	1	170	7600	26.7	5.2
eil31	103	0.5	0.0	0.0	1	32	747	58.1	0.5
eil33	221	8.5	0.0	0.0	1	215	16220	48.5	8.5
eil51	213	2.4	0.0	0.0	1	150	508	54.9	2.4
eil76	269	3.1	0.0	0.0	1	118	907	59.2	3.1
eil101	315	5.6	0.0	0.0	1	159	1049	57.4	5.6
cmt101	253	36.9	1.0	0.0	2	514	1030	56.4	55.2
cmt121	273	411.1	0.0	0.8	39	1350	715	52.9	1525.6
cmt151	350	131.8	0.1	0.1	5	451	1537	55.6	167.3
cmt200	382	147.4	0.0	0.0	17	859	2198	62.0	596.3
gil262	1189	365.8	0.2	0.2	35	2370	8456	54.8	3252.7

Table 13.4. Results for problems of Class I (OP and VRP instances) with $\alpha = 0.50$.

Tables 13.4 to 13.9 report on the computational behavior of the branch-and-cut code FST. Each table (except Table 13.5) gives:

Name : the problem name;

t_0 : the maximum total time (only for Classes I and II);

r-time : the total time spent at the root node;

%-LB : the percentage ratio (optimum – LB)/optimum, where LB is the value of the best heuristic solution computed at the root node;

%-UB : the percentage ratio (UB – optimum)/optimum, where UB is the upper bound computed at the root node;

nodes : the total number of nodes generated (1 means that the problem required no branching);

cuts : the total number of cuts generated (including the total time restriction (43));

optval : the optimal solution value (only for classes I and II);

%-vis : the percentage number of nodes visited by the optimal solution;

t-time : the total computing time spent by the branch-and-cut code.

The computing times reported are expressed in seconds, and refer to CPU times on an HP Apollo 9000/720 computer running at 80 MHz (59 SPEC's, 58 MIPS, 18 MFlops). A time limit of 18,000 seconds (5 hours) has been imposed for each run. For the instances exceeding the time limit, we report 't.l.' in the *t-time* column, and compute the

Name	t-LP	t-sep	cuts	log	gsec	2-mat	cover	path	cond	b-cov
op21	0.4	0.1	58	27	14	0	0	0	16	0
op32	0.9	0.2	91	31	46	2	1	0	10	0
op33	1.0	0.5	109	31	47	2	0	0	28	0
att48	0.4	0.2	55	23	17	11	0	0	3	0
eil30	3.6	0.8	170	43	84	0	4	0	38	0
eil31	0.2	0.2	32	15	5	1	1	0	9	0
eil33	5.4	2.0	215	42	86	2	14	20	50	0
eil51	1.6	0.4	150	48	54	5	0	0	42	0
eil76	1.4	1.1	118	50	49	7	0	0	11	0
eil101	2.7	1.5	159	61	59	19	1	0	18	0
cmt101	23.9	16.0	514	114	362	14	8	6	8	1
cmt121	503.5	652.9	1350	189	719	86	82	63	172	38
cmt151	27.8	121.0	451	127	210	61	7	0	39	6
cmt200	84.8	422.6	859	177	449	82	38	0	94	18
gil262	740.0	1873.6	2370	262	1131	154	83	49	644	46

Table 13.5. Additional results for Class I ($\alpha = 0.50$) problems

corresponding results by considering the best available as the optimal solution value. Hence, for the time-limit instances the column %-UB gives an upper bound on the percentage approximation error.

Table 13.4 refers to the instances of Class I with $\alpha = 0.5$. We also report, in Table 13.5, additional information on the overall time spent within the LP solver (*t-LP*) and the separation procedures (*t-sep*), and on the number of logical (*log*), inequalities (45) (*gsec*), 2-matching (*2-mat*), cover (*cover*), path (*path*), conditional (*cond*), and branch cover (*b-cov*) constraints generated.

Tables 13.6 to 13.8 refer to the instances of Class II, with prizes computed according to Generation 1, 2, and 3, respectively. The parameter α has been set to 0.50. Cases $\alpha = 0.25$ and $\alpha = 0.75$ present comparable results.

Table 13.9 reports average results over 10 random instances belonging to Class III, with $\alpha = 0.2, 0.4, 0.6,$ and 0.8 , and $n = 25, 50, 100, 300,$ and 500 . Larger instances could be solved as well, since for this class the computing time tends to increase very slowly with n for $n \geq 200$. As a comparison, the branch-and-bound algorithm of Laporte and Martello [536] ran into difficulties when solving instances with $n = 25$ and $\alpha \geq 0.6$, and with $n = 50$ and $\alpha \geq 0.4$. For example, running (on the HP Apollo 9000/720 computer) the Laporte and Martello code on the instances with $n = 25$ required on average 0.1 seconds for $\alpha = 0.2$, 77.2 seconds for $\alpha = 0.4$, more than 2 hours for $\alpha = 0.6$; whereas for $\alpha = 0.8$ no instance was solved within the 5 hour time-limit.

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	optval	%-vis	t-time
gr137	34927	178.6	0.0	0.0	1	553	81	59.1	178.6
pr144	29269	240.3	0.0	0.0	1	1170	77	53.5	240.3
kroa150	13262	582.2	0.0	1.2	85	2594	86	57.3	4669.0
krob150	13065	145.6	0.0	0.0	1	729	87	58.0	145.6
pr152	36841	204.6	0.0	0.0	1	917	77	50.7	204.6
u159	21040	497.6	0.0	0.0	1	1912	93	58.5	497.6
rat195	1162	331.9	0.0	0.0	1	1323	102	52.3	331.9
d198	7890	716.3	0.0	0.0	1	1431	123	62.1	716.3
kroa200	14684	395.0	0.0	0.0	1	1254	117	58.5	395.0
krob200	14719	683.6	0.0	0.0	1	1635	119	59.5	683.6
gr202	20080	150.6	0.0	0.0	1	603	147	72.8	150.6
ts225	63322	9.7	0.0	0.8	107	6040	125	55.5	t.l.
pr226	40185	1955.3	0.0	5.2	25	1019	134	59.3	t.l.
gr229	1765	75.0	0.0	0.0	1	431	176	76.9	75.0
gil262	1189	120.6	0.0	0.0	1	789	158	60.3	120.6
pr264	24568	2860.2	0.0	0.0	1	1694	132	50.0	2860.2
pr299	24096	5726.3	1.2	1.2	8	4524	162	54.2	14244.0
lin318	21045	2558.0	0.0	0.5	5	2653	205	64.5	3169.9
rd400	7641	874.0	1.7	0.4	29	1821	239	59.8	4272.5

Table 13.6. Results for Class II (TSPLIB instances) and Generation 1 ($\alpha = 0.50$)

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	optval	%-vis	t-time
gr137	34927	797.9	0.0	0.3	81	1929	4294	57.7	3193.0
pr144	29269	668.0	0.0	0.7	11	1579	4003	51.4	1409.0
kroa150	13262	460.1	0.6	0.9	47	2876	4918	54.0	3950.6
krob150	13065	735.7	0.0	0.1	5	1795	4869	52.7	1018.1
pr152	36841	188.6	0.0	0.0	1	836	4279	48.0	188.6
u159	21040	518.0	0.4	0.2	21	1853	4960	54.1	1772.4
rat195	1162	1750.1	0.0	0.2	13	2691	5791	48.2	2498.6
d198	7890	1337.8	0.1	0.1	27	1999	6670	56.1	2517.1
kroa200	14684	515.2	0.0	0.1	15	1147	6547	54.5	805.1
krob200	14719	1240.7	0.1	0.1	17	2563	6419	50.5	3522.8
gr202	20080	441.7	0.8	0.0	31	1945	7848	65.8	3847.6
ts225	63322	763.3	0.0	0.1	5	1627	6834	54.2	1195.5
pr226	40185	3973.9	0.1	0.3	27	1842	6615	46.6	t.l.
gr229	1765	329.5	0.5	0.1	47	1415	9187	72.1	4261.4
gil262	1189	2783.0	0.1	0.2	23	2080	8321	50.8	5574.6
pr264	24568	4253.3	0.0	0.0	1	2211	6654	50.0	4253.3
pr299	24096	10803.8	0.0	0.0	5	1900	9161	49.8	t.l.
lin318	21045	1370.0	0.0	0.0	41	1392	10900	60.7	t.l.
rd400	7641	837.6	0.1	0.2	76	3721	13648	54.5	t.l.

Table 13.7. Results for Class II (TSPLIB instances) and Generation 2 ($\alpha = 0.50$)

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	optval	%-vis	t-time
gr137	34927	2401.9	45.5	0.1	5	5386	3979	51.8	4958.7
pr144	29269	1573.8	0.0	0.1	65	2632	3809	43.1	t.l.
kroa150	13262	269.7	0.1	0.6	181	1646	5039	52.7	3828.9
krob150	13065	1112.5	0.0	0.1	13	1472	5314	56.7	1363.9
pr152	36841	1099.0	0.7	1.3	391	3159	3905	48.7	13736.7
u159	21040	1308.4	0.0	0.2	5	2171	5272	52.8	1447.2
rat195	1162	3672.2	0.1	0.1	9	1528	6195	47.7	3975.4
d198	7890	1810.0	0.0	0.2	197	2361	6320	61.6	8635.7
kroa200	14684	3116.5	0.0	0.2	41	2161	6123	51.5	6548.9
krob200	14719	642.6	0.0	0.1	9	905	6266	51.0	783.7
gr202	20080	654.2	0.5	0.3	298	1947	8632	71.3	11113.5
ts225	63322	3437.6	0.0	0.4	27	1598	7575	55.1	5821.8
pr226	40185	3379.5	16.0	0.1	51	5310	6993	52.2	7923.2
gr229	1765	1667.1	0.0	0.0	11	1613	6347	67.7	1891.5
gil262	1189	6177.4	0.2	0.0	27	2386	9246	56.5	9574.0
pr264	24568	4011.3	0.0	0.0	1	2625	8137	39.8	4011.3
pr299	24096	14699.4	0.0	0.0	2	1787	10358	49.8	t.l.
lin318	21045	8597.5	0.0	0.0	12	1308	10382	60.7	t.l.
rd400	7641	14257.1	0.0	0.0	3	1418	13229	55.8	t.l.

Table 13.8. Results for Class II (TSPLIB instances) and Generation 3 ($\alpha = 0.50$)

On the whole, the performance of the branch-and-cut code is quite satisfactory for our families of instances. In most cases, the upper and lower bounds computed at the root node are very tight, and a few branchings are needed. The code was able to solve to proven optimality almost all the random instances of Class III (except 1 instance for $n = 500$), and most of the “real-world” instances of Classes I and II. For the instances exceeding the time limit, the computed solution is very close to the optimal one (see column %-UB).

According to Table 13.5, most of the generated constraints are inequalities (45), 2-matching, logical and conditional cuts. For some “difficult” instances, a relevant number of cover and path inequalities is generated.

Additional computational experience has been performed on the class of random instances considered in the work by Gendreau, Laporte and Semet [355], called Class IV in the sequel. These instances were generated by using the original Gendreau-Laporte-Semet code. The instances are similar to those of Class II and Generation 2, but the nodes are generated as random points in the $[0, 100]^2$ square according to a uniform distribution. The corresponding values of $v(TSP)$ were computed by means of the algorithm of Padberg and Rinaldi [648]. Table 13.10 reports average results over 5 random instances belonging to Class IV, with $\alpha=0.1, 0.3, 0.5, 0.7, 0.9$, and $n=101, 121, 161, 261$, and 301.

n	α	r-time	%-LB	%-UB	nodes	cuts	%-vis	t-time
25	0.2	1.1	0.0	0.0	1.1	63.4	28.8	1.2
25	0.4	38.8	0.0	0.9	3.8	138.2	56.8	42.7
25	0.6	46.6	0.0	0.3	2.6	101.0	74.0	63.4
25	0.8	42.1	0.0	0.4	4.0	91.5	86.0	51.5
50	0.2	41.5	0.0	0.4	1.6	169.6	32.8	53.8
50	0.4	32.1	0.0	0.5	10.0	220.2	59.8	99.6
50	0.6	30.8	0.3	0.3	21.8	263.1	76.4	147.0
50	0.8	10.5	0.2	0.1	10.4	128.1	90.2	58.9
100	0.2	61.3	0.0	0.3	8.2	359.6	35.3	149.6
100	0.4	35.9	0.2	0.2	28.4	403.5	60.9	340.3
100	0.6	33.7	0.4	0.1	34.6	421.4	80.8	445.2
100	0.8	41.9	0.3	0.0	35.4	273.9	93.2	403.8
300	0.2	102.7	0.1	0.1	15.0	484.8	30.1	363.5
300	0.4	139.1	0.2	0.0	43.6	652.4	57.5	1816.5
300	0.6	203.9	0.2	0.0	26.2	355.3	80.4	1949.5
300	0.8	237.2	1.5	0.0	4.3	186.6	99.8	2038.5
500	0.2	189.6	0.0	0.0	5.4	300.5	27.0	325.4
500	0.4	317.4	0.3	0.0	9.9	322.9	53.6	1418.0
500	0.6	408.4	1.1	0.0	9.7	284.2	78.8	5327.9
500	0.8	650.2	1.4	0.0	2.4	116.2	100.0	2454.0

Table 13.9. Average results over 10 random instances of Class III

n	α	N_1	N_2	N_3	r-time	%-LB	%-UB	nodes	cuts	%-vis	t-time
101	0.1	5	5	5	60.5	0.0	0.8	3.4	559.8	11.7	193.3
101	0.3	5	5	5	186.7	0.0	0.3	3.0	849.6	33.5	228.7
101	0.5	5	5	5	175.3	0.0	0.3	11.8	806.4	55.6	350.2
101	0.7	5	5	5	70.9	0.3	0.2	37.8	747.6	75.6	584.5
101	0.9	5	5	1	55.2	1.1	0.3	222.6	872.6	90.7	2467.5
161	0.1	5	5	5	233.8	0.0	0.2	1.4	900.6	12.2	275.9
161	0.3	5	5	2	539.6	0.4	0.5	11.0	1638.2	34.9	1254.6
161	0.5	5	5	0	249.4	0.2	0.2	20.2	956.2	55.0	737.2
161	0.7	4	3	0	185.2	0.7	0.2	101.5	1419.5	72.8	3563.2
161	0.9	5	4	-	136.0	3.8	0.1	90.0	697.2	90.3	2981.2
201	0.1	5	5	5	397.9	0.0	0.6	5.8	1345.4	11.4	673.2
201	0.3	5	4	-	555.7	0.2	0.6	37.4	2188.2	34.3	2793.0
201	0.5	5	5	-	412.1	0.0	0.1	9.8	1062.6	54.5	724.1
201	0.7	4	1	-	952.7	1.4	0.1	88.5	2136.2	71.8	6118.0
201	0.9	4	1	-	305.9	2.4	0.1	121.8	1224.5	90.4	9917.1
301	0.1	5	4	1	1358.1	0.3	1.1	17.0	2982.8	11.7	3729.6
301	0.3	2	0	-	914.5	1.5	0.6	98.0	3779.0	35.9	12145.3
301	0.5	3	1	-	530.2	0.9	0.2	59.0	3151.0	54.7	8787.9
301	0.7	1	1	-	616.6	0.3	0.0	14.0	1303.0	72.1	5064.7
301	0.9	0	0	-	—	—	—	—	—	—	t.l.

Table 13.10. Average results over 5 random instances of Class IV

Column N_3 gives the number of instances successfully solved by the Gendreau-Laporte-Semet code within a time limit of 10,000 SUN Sparc station 1000 CPU seconds. According to Dongarra [258], the HP Apollo 9000/720 computer is about 1.8 times faster than that used by Gendreau, Laporte and Semet, hence their time limit corresponds to about 5,555 HP 9000/720 CPU seconds. Columns N_1 and N_2 give the number of instances successfully solved by code FST within a time limit of 18,000 and 5,555 HP 9000/720 CPU seconds, respectively. The remaining columns are as in previous tables, and refer to the execution of code FST with the 18,000 second time limit. As in [355], averages are computed with respect to the instances solved to proven optimality.

A comparison of columns N_2 and N_3 shows that code FST is capable of solving a number of instances substantially larger than the Gendreau-Laporte-Semet code, within approximately the same time limit. Moreover, the values of both the lower and upper bounds computed by FST are tighter than those reported in [355]. For the cases in which the Gendreau-Laporte-Semet code successfully solved all the five instances, the average LB and UB ratios of CFT (0.02% and 0.28%, respectively) compare very favorably with the corresponding values reported in [355] (3.59% and 1.74%, respectively). On the whole, the instances of Class IV appear more difficult than those in the previous classes.

Acknowledgement

The work of the first and third authors has been supported by C.N.R. and by M.I.U.R., Italy. The work of the second author has been supported by the research projects TIC-2000-1750-CO6-02 and PI2000/116, Spain.

Chapter 14

THE PRIZE COLLECTING TRAVELING SALESMAN PROBLEM AND ITS APPLICATIONS

Egon Balas

Graduate School of Industrial Administration

Carnegie-Mellon University

Pittsburgh, PA 15213, USA

eb17+@andrew.cmu.edu

1. Introduction

Few combinatorial optimization problems have such widespread applicability as the traveling salesman problem (TSP). Practically any situation involving decisions that affect the sequence in which various actions, tasks or operations are to be executed, has a TSP aspect to it. Often such problems can be formulated as special TSP instances, but even more frequently the problem at hand turns out to be some “relative” of the TSP, or, mathematically speaking, some generalization of the TSP.

Examples abound. A helicopter touring the platforms of an offshore oilfield to pick up and deliver items, will follow a minimum total distance route subject to the condition that certain platforms have to be visited before others: this is a precedence-constrained TSP. The drilling head of a boring machine that makes holes in a wafer to produce a chip, has to move between holes along a route with a minimum total length, subject to the condition that after every so many holes the drilling head has to stop at one of those designated holes where its cutting edge can be sharpened: this is a TSP with refueling (every so often the salesman has to visit a refueling station). The delivery vehicle carrying perishable goods has to visit its customers in a sequence that will minimize the distance traveled, subject to the condition that certain customers have to be visited between certain hours: this is a traveling salesman problem with time windows. And so on, and so on.

The above examples represent traveling salesman problems with side conditions. A more general class of relatives consists of TSP-like problems which, however, ask for a tour on a subset of the cities. Best known among these is the Prize Collecting TSP, discussed in this paper, where a minimum cost tour of a subset of the cities is sought subject to the condition that a certain amount of prize money be collected from the cities visited. A close relative, the orienteering problem, asks for a maximum amount of prize money subject to a limit on the cost of the tour. Other such problems include the international TSP, where the cities are partitioned into groups and a tour has to contain a representative of each group; and the median cycle problem, where a subset of the cities forms a cycle and each remaining city is assigned to some node of the cycle. The common feature of the problems in this category is the combination of two kinds of decisions, the *selection* of some cities and the *ordering* of the cities selected (see also Chapter 13).

2. An Application

A steel rolling mill processes steel slabs into sheet. At any given time, the rolling mill has an inventory of slabs and a list of orders. Each order asks for the production of a coil of sheet with prescribed width, thickness (gauge) and length. Based on their technological characteristics (grade) and dimensions, the slabs are grouped into cells assigned to orders. The outcome of this operation is a *cell inventory*.

The processing of a slab into sheet is done by rolling it, i.e. passing it between layers of rolls that flatten it into a band and ultimately a sheet. Due to wear and tear, the rolling process requires periodic roll changes, both partial and general. A rolling schedule between two general roll changes, which can be anywhere between 4 and 10 hours depending on the type of material that is being rolled, is called a *round* (lineup). A round comprises cells with certain characteristics prescribed by the *round definition*, which specifies ranges for the technological and dimensional attributes of the material to be processed and sets lower and upper bounds on the total amount to be rolled. The round definition is usually selected from among 5 - 15 standard definition types.

Once a round definition has been chosen, all those cells compatible with the definition become candidates for the round. The scheduling task then consists of choosing a subset of the candidates that meets the lower and upper bounds on the total amount to be rolled, and ordering them into an appropriate sequence.

The rolling sequence affects heavily both the quality of the product and the efficiency of the operation. Furthermore, the selection of cells

for the round, apart from reflecting due dates and similar considerations, affects in a major way the overall characteristics of the rolling sequence that can be obtained. For instance, smooth transition between the gauges of thin slabs is a major objective of the sequencing; but if the selection omits to include into the round slabs with a gauge in a certain range, the smooth transition may be impossible to achieve. Therefore the two tasks – selection and sequencing – cannot be separated and solved consecutively without impairing the quality of the outcome.

This and other considerations have led Balas and Martin [61, 77, 76] to formulate this as a Prize Collecting Traveling Salesman Problem (PCTSP). A salesman gets a prize w_k in every city k that he visits and pays a penalty c_ℓ to every city ℓ that he fails to visit. Traveling at cost c_{ij} between cities i and j , our salesman wants to find a tour that minimizes his travel costs and penalties, subject to a lower bound w_0 on the amount of prize money he collects. Here city i corresponds to cell i , c_{ij} represents the cost of rolling cell j right after cell i , w_k is the amount of rolling material in cell k , w_0 is the desired size of the round, and c_ℓ is the penalty for not including cell ℓ into the round. Cell 1 is a dummy whose purpose is to make the sequence into a tour.

If we let y_i be 0 if city i is included into the tour and 1 otherwise, and let x be the incidence vector of the arcs of the tour, then our problem can be formulated on a complete directed graph $G_L = (N, A \cup L)$ with nodes $i \in N$, arcs $(i, j) \in A$, and loops $(i, i) \in L$, as

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in N} c_i y_i, \tag{1}$$

subject to

$$\begin{aligned} \sum_{j \in N \setminus \{i\}} x_{ij} + y_i &= 1 & i \in N \\ \sum_{i \in N \setminus \{j\}} x_{ij} + y_j &= 1 & j \in N \end{aligned} \tag{2}$$

$$\sum_{i \in N} w_i y_i \leq U \tag{3}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in A, \quad y_i \in \{0, 1\}, \quad i \in N \tag{4}$$

$$G_L(x, y) \text{ has one cycle of length } k \geq 2 \text{ (and } n - k \text{ loops)}. \tag{5}$$

Here x and y are the incidence vectors of arcs and loops, respectively, of G_L , c_{ij} , c_ℓ and w_i are as defined above, with $c_1 = \infty$, $U := \sum_{i \in N} w_i - w_0$, while $G_L(x, y)$ is the subgraph of G_L with node set N , an arc (i, j) for

every $i \neq j$ such that $x_{ij} = 1$ and a loop (i, i) for every i such that $y_i = 1$.

A typical solution is shown in figure 14.1.

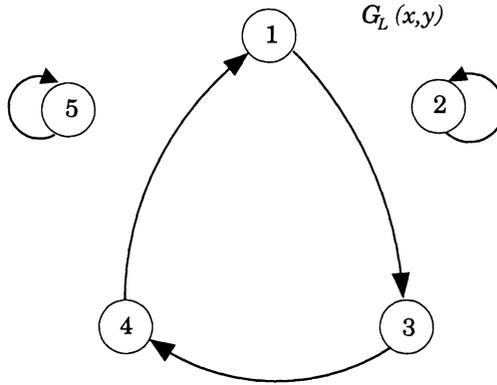


Figure 14.1. A typical solution.

The standard problem sizes that arise in the practice of rolling mills range roughly between 100-300 nodes (cells).

The ROLL-A-ROUND system [77], a software package in use at several steel mills, is based on this formulation and various adaptations of it, treated by approximation methods.

The main procedure, KCAP (for knapsack-constrained assignment problem), uses as a relaxation the problem obtained by removing constraint (5). This is an assignment problem with a knapsack constraint, which is taken into a Lagrangean objective function. The optimal value of the Lagrange multiplier is found by bisection; the Lagrangean subproblems are assignment problems solved by a shortest augmenting path routine. The resulting primal solution is an assignment that satisfies (3) but typically violates (5). Several variants of a patching procedure are then used to combine the subtours of the assignment into a tour. The procedure has some built in variations so as to produce several rather than just one solution.

Two other heuristics used are tour building procedures that construct “good” path segments that they eventually combine into a tour of appropriate length. Each of these two heuristics generates one solution.

Finally, a variable-depth interchange procedure is applied to each of the solutions found by the above mentioned routines, in an attempt to improve their quality.

A branch and bound procedure for the exact solution of the PCTSP has been developed by Fischetti and Toth [302].

The results of the next four sections are from [63, 65].

3. Polyhedral Considerations

Let P^* denote the PCTS polytope, i.e.:

$$P^* := \text{conv}\{(x, y) \in \mathbb{R}^{A \cup N} : (x, y) \text{ satisfies (2), (3), (4), (5)}\}.$$

Obviously, P^* contains as a face the ATS polytope P defined on the loopless digraph $G = (N, A)$, since

$$P := P^* \cap \{(x, y) : y_i = 0, \forall i \in N\}$$

and P^* becomes P whenever $U < w_i, \forall i \in N$.

Another polytope that contains the ATS polytope as a face is the relaxation P_0 of P^* , corresponding to the case when $U \geq \sum_{i \in N} w_i$ (which is equivalent to removing the knapsack constraint (3) altogether), i.e.:

$$P_0 := \text{conv}\{(x, y) \in \mathbb{R}^{A \cup N} : (x, y) \text{ satisfies (2), (4), (5)}\}.$$

The *cycle and loops polytope* P_0 , defined on the same digraph G_L as P^* , has a simpler structure than that of P^* and is therefore more convenient to work with. On the other hand, assuming that $w_i \leq U, i \in N$, we have that

$$\dim P_0 = \dim P^* = (n - 1)^2,$$

and so, if some inequality $\alpha x \leq \alpha_0$ is shown to be facet-defining for P_0 by exhibiting $(n - 1)^2$ affinely independent points $x \in P_0$ such that $\alpha x = \alpha_0$, and if these points also happen to satisfy inequality (3), then $\alpha x \leq \alpha_0$ will have been shown to be facet defining for P^* . Besides these considerations, P_0 is of interest also as a vehicle for the study of the cycle polytope (the convex hull of simple directed cycles) of a digraph, which is the projection of P_0 onto \mathbb{R}^A .

For $W \subseteq A$, we denote $x(W) := \sum_{(i,j) \in W} x_{ij}$. For $S, T \subseteq N$, we denote $x(S, T) := \sum_{i \in S} \sum_{j \in T \setminus \{i\}} x_{ij}$, $y(S) := \sum_{i \in S} y_i$. Throughout the rest of this chapter, we assume that $n \geq 4$, and that $w_i \leq U$ for all $i \in N$.

A first class of facets of P^* comes from the associated knapsack polytope, namely:

$$KP := \text{conv}\{y \in \{0, 1\}^N : y \text{ satisfies (3)}\}.$$

It is well known that a large class of facets of KP come from lifting inequalities of the form $y(S) \leq |S| - 1$, where S is a minimal cover, i.e. a minimal subset of N such that $\sum_{i \in S} w_i y_i > U$.

Theorem 1 Let $S, T \subset N$, $S \cap T = \emptyset$, $N \setminus (S \cup T) \neq \emptyset$, and let α_i , $i \in T$, be positive integers. If the inequality

$$x(S) + \sum_{i \in T} \alpha_i y_i \leq |S| - 1 \tag{6}$$

defines a facet of KP and $T \neq \emptyset$, then (6) defines a facet of P^* .

Next, we turn to the class of facets of P_0 and of P^* that come from the subtour elimination inequalities for the associated ATS polytope P .

Theorem 2 For all $S \subset N$, $2 \leq |S| \leq n - 1$, and all $k \in S$, $l \in N \setminus S$, the inequality

$$x(S, S) + y(S \setminus \{k\}) - y_l \leq |S| - 1 \tag{7}$$

is valid for P_0 . Further, for $|S| \leq n - 2$ (7) defines a facet of P_0 .

This result carries over to P^* whenever the coefficients w_i of (3) satisfy a certain condition. In particular, if $w(T) \leq U$ for $T := S$ and $T := N \setminus S$, then (7) defines a facet of P^* . Otherwise (7) can be strengthened by adding y_k to the lefthand side (if $w(S) > U$), or deleting $-y_l$ (if $w(N \setminus S) > U$) or both.

The inequalities corresponding to the subtour elimination constraints of the ATSP can be used, along with

$$y(N) \leq n - 2 \tag{8}$$

to replace condition (5) in an integer programming representation of P_0 and P^* , namely:

$$P_0 := \text{conv} \left\{ (x, y) \in \mathbb{R}^{A \cup N} \mid \begin{array}{l} (x, y) \text{ satisfies (2), (4), (8) and (7) for all} \\ S \subset N, 2 \leq |S| \leq n - 2, k \in S, l \in N \setminus S \end{array} \right\}$$

and

$$P^* := \text{conv}(P_0 \cap \{(x, y) \in \mathbb{R}^{A \cup N} : y \text{ satisfies (3)}\}).$$

Next, we turn to other facet defining inequalities for the ATS polytope P and show how to derive from them facet defining inequalities for P_0 and P^* .

4. Lifting the Facets of the ATS Polytope

Since the ATS polytope P is the restriction of P_0 to the subspace defined by $y_i = 0$, $i \in N$, facet defining inequalities for P can be lifted to facet defining inequalities for P_0 . We use sequential lifting to obtain the coefficients of the loop variables. The basic theorem on sequential

lifting [624, 641] applies to full dimensional polyhedra, and neither P_0 nor P^* are full dimensional. Nevertheless, the theorem (to follow) is valid, since whenever a loop variable is added to the lifting sequence, the dimension of the polytope increases by exactly one.

Suppose that we introduce the loops into G , and the associated variables into P_0 , one by one in some arbitrary sequence i_1, \dots, i_n . Let $P_0(k)$ denote the polytope obtained from P by introducing the first k loop variables $y_{i_j}, j = 1, \dots, k$.

Theorem 3 *Let $\alpha x \leq \alpha_0$, with $\alpha_0 > 0$, be any facet defining inequality for P . For $k = 1, \dots, n$, define $\beta_{i_k} = \alpha_0 - z(P_0(k))$, where*

$$z(P_0(k)) := \max\{\alpha x + \sum_{j=1}^{k-1} \beta_{i_j} y_{i_j} : (x, y) \in P_0(k), y_{i_k} = 1\}.$$

Then the inequality

$$\alpha x + \sum_{j=1}^k \beta_{i_j} y_{i_j} \leq \alpha_0 \tag{9}$$

is valid and facet defining for $P_0(k)$.

The coefficients of a sequentially lifted inequality are, in general, sequence-dependent. However, the following property of the lifting coefficients will be helpful in detecting situations, quite frequent in our case, when the lifted inequality is unique.

Corollary 4 *Let $\bar{\beta}_j$ and $\underline{\beta}_j$ be the values of the coefficient β_j when lifted first and last, respectively. Then $\bar{\beta}_j \geq \beta_j \geq \underline{\beta}_j$.*

Next we state several properties of the lifting coefficients that are specific to our polytope (see [65] for details). These properties form the basis of the facet-lifting procedures described in the next three sections.

Theorem 5 *For any $k \in \{1, \dots, n\}$,*

$$\bar{\beta}_k \leq \min\{\alpha_{i_k} + \alpha_{k_j} - \alpha_{i_j} : \exists x \in P \text{ with } \alpha x = \alpha_0 \text{ and } x_{i_k} = x_{k_j} = 1\}.$$

Proof. We have

$$\begin{aligned} \bar{\beta}_k &:= \alpha_0 - \max\{\alpha' x' : x' \in P(G - \{k\})\} \\ &= \max\{\alpha x : x \in P(G)\} - \max\{\alpha' x' : x' \in P(G - \{k\})\} \\ &\leq \alpha_{i_k} + \alpha_{k_j} - \alpha_{i_j} \end{aligned}$$

for any $i, j \in N \setminus \{k\}$ for which there exists $x \in P$ with $\alpha x = \alpha_0$ and $x_{ik} = x_{kj} = 1$. Here x' and α' are the vectors obtained from x and α , respectively, by deleting their component indexed by k . Indeed, the first equality follows from the definition of $\bar{\beta}_k$, while the second one from $\alpha_0 = \max\{\alpha x : x \in P\}$. To see that the last inequality holds, let $\bar{x} \in P$ satisfy $\alpha \bar{x} = \alpha_0$ and $\bar{x}_{ik} = \bar{x}_{kj} = 1$. Then, \bar{x}' defined by $\bar{x}'_{ij} = 1$ and $\bar{x}'_{hl} = \bar{x}_{hl}$ for all $h, l \in N \setminus \{k\}$, $(h, l) \neq (i, j)$, belongs to $P(G - \{k\})$. \square

Corollary 6 *Let $\alpha x \leq \alpha_0$ be a valid inequality for P and for P_0 . If for some $k \in N$ there exist indices $i, j \in N \setminus \{k\}$, $i \neq j$, such that*

$$\alpha_{ik} = \alpha_{kj} = \alpha_{ij} = 0,$$

and a tour $x \in P$ such that

$$\alpha x = \alpha_0, \quad x_{ik} = x_{kj} = 1,$$

then $\beta_k = 0$ in all liftings of $\alpha x \leq \alpha_0$.

Proof. If the stipulated conditions hold, then from Theorem 5, $\bar{\beta}_k \leq 0$. Further, since $\alpha x \leq \alpha_0$ is valid for P_0 , $\underline{\beta}_k \geq 0$. Thus $\beta_k = 0$. \square

Theorem 7 *Let $\alpha x \leq \alpha_0$ be a facet defining inequality for P , and let the inequality*

$$\alpha x + \beta y \leq \alpha_0 \tag{10}$$

be valid for P_0 and P^ . If for every $k \in N$ there exists a pair $i, j \in N \setminus \{k\}$ such that*

$$\beta_k = \alpha_{ik} + \alpha_{kj} - \alpha_{ij} \tag{11}$$

and a tour $x \in P$ such that $\alpha x = \alpha_0$ and $x_{ik} = x_{kj} = 1$, then (10) defines a facet of P_0 and of P^ .*

Proof. If the stipulated condition holds, then from Theorem 5 for every $k \in N$

$$\begin{aligned} \beta_k \leq \bar{\beta}_k &\leq \min\{\alpha_{hk} + \alpha_{kl} - \alpha_{hl} : \exists x \in P \text{ with } \alpha x = \alpha_0 \\ &\quad \text{and } x_{hk} = x_{kl} = 1\} \\ &\leq \alpha_{ik} + \alpha_{kj} - \alpha_{ij}, \end{aligned}$$

where i, j is a pair for which (11) is satisfied. But then $\beta_k = \bar{\beta}_k$ for all $k \in N$, which in view of Theorem 3 implies that (10) defines a facet of P_0 .

One can show that (10) also defines a facet of P^* by exhibiting $\dim P^*$ affinely independent points $(x^i, y^i) \in P^*$ such that $\alpha x^i + \beta y^i = \alpha_0$. \square

Theorem 7 will be the main tool for lifting facets of P into facets of P_0 and P^* .

Corollary 8 *Let $\alpha x \leq \alpha_0$ be a facet defining inequality for P , valid for P_0 . Let $k \in N$ satisfy one of the following conditions:*

- (a) $\alpha_{ik} = 0$ for all $i \neq k$, and there exists $j_* \neq i, k$ such that $\alpha_{kj_*} = 0$ and $\alpha_{ij_*} \geq 0$ for all $i \neq j_*$; or
- (b) $\alpha_{kj} = 0$ for all $j \neq k$, and there exists $i_* \neq j, k$ such that $\alpha_{i_*k} = 0$ and $\alpha_{i_*j} \geq 0$ for all $j \neq i_*$.

Then $\beta_k = 0$ in all the liftings of $\alpha x \leq \alpha_0$.

Proof. Suppose that (a) holds for k [an analogous reasoning applies when (b) holds for k]. Since $\alpha x \leq \alpha_0$ defines a facet of P , there exists $\bar{x} \in P$ with $\alpha \bar{x} = \alpha_0$ and $\bar{x}_{kj_*} = 1$. Let $i_0 \neq k, j_*$ be the index for which $\bar{x}_{i_0k} = 1$. Then,

$$\bar{\beta}_k \leq \alpha_{i_0k} + \alpha_{kj_*} - \alpha_{i_0j_*} \leq 0.$$

Also, since $\alpha x \leq \alpha_0$ is valid for P_0 , $\underline{\beta}_k \geq 0$. Since $\bar{\beta}_k \geq \beta_k \geq \underline{\beta}_k$, $\beta_k = 0$ in all the liftings of $\alpha x \leq \alpha_0$. \square

Corollary 9 *Let $\alpha x \leq \alpha_0$ be a facet-defining inequality for P , valid for P_0 . Let $k \in N$ be such that $\alpha_{ik} = \alpha_{kj} = 0$ for all $i, j \in N \setminus \{k, l\}$, where $l \neq k$ is an arbitrary but fixed index. Then $\beta_k = 0$ in all the liftings of $\alpha x \leq \alpha_0$.*

Proof. We claim that there exists $x \in P$ such that $\alpha x = \alpha_0$ and $x_{ik} = x_{kj} = 1$ for some pair $i, j \in N \setminus \{l\}$, $i \neq j$. For suppose not, then $x \in P$ and $\alpha x = \alpha_0$ imply that $x_{lk} + x_{kl} = 1$, contrary to the assumption that $\alpha x \leq \alpha_0$ defines a facet of P . But then $\bar{\beta}_k \leq \alpha_{ik} + \alpha_{kj} - \alpha_{ij} \leq 0$ and $\underline{\beta}_k \geq 0$ imply $\beta_k = 0$ for all the liftings of $\alpha x \leq \alpha_0$. \square

5. Primitive Inequalities from the ATSP

Given a valid inequality $\alpha x \leq \alpha_0$ for the ATS polytope P , two nodes h and k are called *clones* with respect to $\alpha x \leq \alpha_0$ if

- (a) $\alpha_{ih} = \alpha_{ik}$ and $\alpha_{hi} = \alpha_{ki}$ for all $i \in N \setminus \{h, k\}$.
- (b) $\alpha_{hk} = \alpha_{kh} = \max\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij} : i, j \in N \setminus \{h, k\}, i \neq j\}$.
- (c) the inequality $\tilde{\alpha}x \leq \alpha_0 - \alpha_{hk}$, where $\tilde{\alpha}$ is the restriction of α to the arcs of $\tilde{G} := G - \{h\}$, is valid for the ATS polytope $P(\tilde{G})$ defined on \tilde{G} .

A valid inequality $\alpha x \leq \alpha_0$ for P is called *primitive* if G has no clones with respect to $\alpha x \leq \alpha_0$.

These concepts were introduced by Balas and Fischetti [73] (see also Chapter 3), who showed that if all the primitive members of a family \mathcal{F} of inequalities define regular facets of the ATS polytope P , then so do all the members of \mathcal{F} . (A facet is regular unless it is trivial or defined by an inequality of the form $x_{ij} + x_{ji} \leq 1$.) They also gave a cloning (clique lifting) procedure by which nonprimitive members of a family of inequalities can be derived from the primitive members by using the properties (a), (b) and (c) of clones.

The concept of a primitive inequality, as well as the associated cloning/clique lifting procedure, has been extended in [65] to the polytopes P_0 and P^* . Given any inequality $\alpha x + \beta y \leq \alpha_0$ for P_0 or P^* , the *restriction* of $\alpha x + \beta y \leq \alpha_0$ to P is the inequality $\alpha x \leq \alpha_0$ obtained by removing the term βy . A valid inequality $\alpha x + \beta y \leq \alpha_0$ for P_0 or P^* will be called *primitive* if its restriction to P is a primitive valid inequality for P .

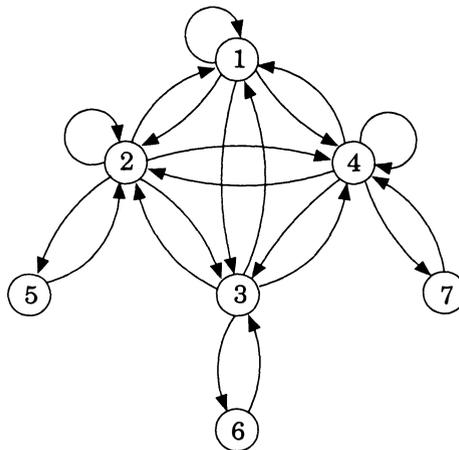


Figure 14.2. Primitive comb inequality for P^*

In this section, we give procedures for generating primitive facet defining inequalities for P_0 and P^* from primitive facet defining inequalities for P . Cloning and clique-lifting will be discussed in the next section. We start with the well known *comb* inequalities. A primitive comb inequality for the ATS polytope is of the form

$$x(H, H) + \sum_{j=1}^t x(T_j, T_j) \leq |H| + \frac{t-1}{2},$$

where H , the *handle*, and $T_j, j = 1, \dots, t$, the *teeth* of the comb, are node sets satisfying

$$|N \setminus (H \cup (\cup_{j=1}^t T_j))| \leq 1 \tag{12}$$

$$|H \cap T_j| = |T_j \setminus H| = 1, \quad j = 1, \dots, t, \text{ with } t \geq 3 \text{ odd}, \tag{13}$$

$$T_i \cap T_j = \emptyset, \quad i, j = 1, \dots, t, \tag{14}$$

and

$$|H \setminus (\cup_{j=1}^t T_j)| \leq 1. \tag{15}$$

These inequalities, analogous to the comb inequalities for the symmetric TS polytope [396, 402] known to be facet defining, define facets for the ATS polytope P for $n \geq 7$ [295]. Now we state the corresponding inequalities for the PCTS polytope.

Theorem 10 *Let H and $T_j, j = 1, \dots, t$, be node sets of G_L satisfying (12), (13), (14) and (15), and let $n \geq 7$. Then the primitive comb inequality*

$$x(H, H) + \sum_{j=1}^t x(T_j, T_j) + y(H) \leq |H| + \frac{t-1}{2} \tag{16}$$

defines a facet of P_0 and of P^ .*

We will defer the proof of this theorem until later in this section, as the family of comb inequalities is a subclass of a more general class to be introduced below. Figure 14.2 shows the support graph of a primitive comb inequality for P^* with $n = 8$ and $s = 3$.

Next we consider the class of *odd CAT* inequalities, known to be facet defining for P [62]. Two arcs $(i, j), (k, l)$ are called *incompatible* if $i = k$, or $j = l$, or $i = l$ and $j = k$, i.e. if they have common heads or common tails, or are antiparallel. A *closed alternating trail* (CAT) in G is a sequence of distinct arcs $T := \{a_1, \dots, a_t\}$ such that for $i, j \in \{1, \dots, t\}$, a_i and a_j are incompatible if and only if $|i - j| = 1$ or $t - 1$. An odd CAT is a CAT of odd length (number of arcs). In Figure 14.3(a), $T := \{(1, 2), (3, 2), (3, 4), (4, 3), (1, 3)\}$ represents an odd CAT with 5 arcs. Let $N(T)$ be the set of endpoints of the arcs of T . A node $i \in N(T)$ is a *source* if $\text{deg}^+(i) \geq 2$, a *sink* if $\text{deg}^-(i) \geq 2$, and *neutral* if $\text{deg}^+(i) = 1$ and $\text{deg}^-(i) = 1$, where the in- and out-degrees are those in G as opposed to G_L . A node can be a source and a sink at the same time. From the definition it is not hard to see that every odd CAT contains at least one 2-cycle with a neutral node.

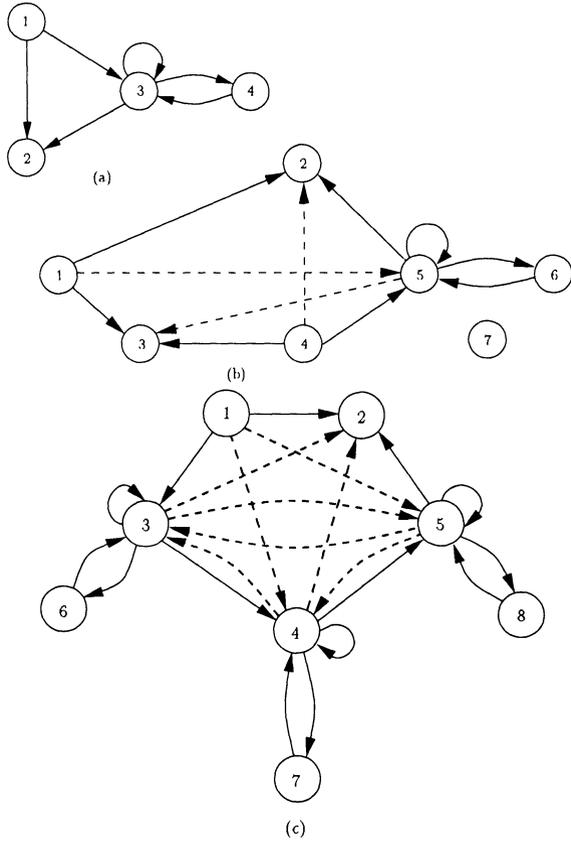


Figure 14.3. Primitive odd CAT inequalities for P^*

A chord of T is an arc in $A \setminus T$ joining two nodes of $N(T)$. A chord of type 1 is one that joins a source i to a sink $j \neq i$. Figures 14.3(b) and (c) show all chords of type 1 in dotted lines. Let T be an odd CAT in G and K the set of its chords of type 1. Then apart from two small pathological cases (with $5 \leq n \leq 6$), the inequality

$$x(T \cup K) \leq \frac{|T| - 1}{2} \tag{17}$$

defines a facet of the ATS polytope P [62].

Next we state the corresponding result for the PCTS polytope. Again, we will deal here with primitive odd CAT inequalities, and leave the general case to the next section. The odd CAT inequality (17) for P is primitive if every 2-cycle of T has a neutral node.

Theorem 11 *Let (17) be a primitive odd CAT inequality that defines a facet of P , and let $Z := \{i \in N(T) : i \text{ is both a source and a sink}\}$. Then the primitive odd CAT inequality*

$$x(T \cup K) + y(Z) \leq \frac{|T| - 1}{2} \tag{18}$$

defines a facet of P_0 and of P^ .*

Again, the proof of Theorem 11 will be deferred until later. Figures 14.3(a)-(c) show the support graphs of primitive odd CAT inequalities for P^* with $|T| = 5, 7$, and 11, respectively.

Next we consider the class of *source-destination (SD) inequalities*, introduced in [73]. This class generalizes the comb inequalities and the odd CAT inequalities. Its primitive members are defined as follows: Let (S, D, W, I, E, Q) be a partition of N with the following properties: S is the (possibly empty) set of *sources*, D is the (possibly empty) set of *destinations* (sinks), $H := W \cup I$ is the (nonempty) *handle*, with $0 \leq |W| \leq 1$ and $|I| \geq 1$, $I := \{i_1, \dots, i_t\}$, $E := \{e_1, \dots, e_t\}$, $T_j := \{i_j, e_j\}$, $j = 1, \dots, t$, are the *teeth*, and $0 \leq |Q| \leq 1$, with $|S| + |D| + t$ odd.

Then the primitive SD inequality defined on G

$$x(S \cup H, D \cup H) + \sum_{j=1}^t x(T_j, T_j) \leq \frac{1}{2}(|S| + |D| + 2|H| + t - 1) \tag{19}$$

is valid for P and, with the exception of three small pathological cases (with $4 \leq n \leq 6$), defines a facet of P if $||S| - |D|| \leq \max\{0, t - 3\}$ [73].

We now give the extension of this class of inequalities to P_0 and P^* .

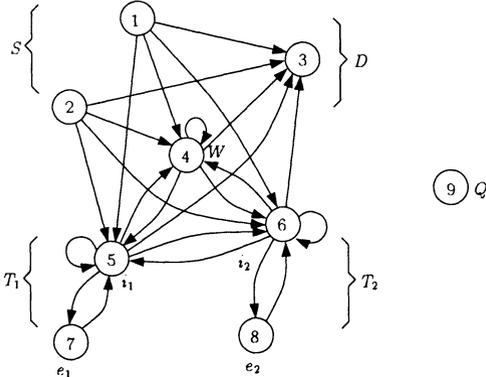


Figure 14.4. Primitive SD inequality for P^* .

Theorem 12 Let (19) be a primitive SD inequality that defines a facet of P , and let $|S| + |D| + t \geq 3$. Then the primitive SD inequality

$$x(S \cup H, D \cup H) + \sum_{j=1}^t x(T_j, T_j) + y(H) \leq \frac{1}{2}(|S| + |D| + 2|H| + t - 1) \tag{20}$$

defines a facet of P_0 and P^* .

Proof outline. Adding $\frac{1}{2}$ times the inequalities (2) for $i \in S \cup H$ and $j \in D \cup H$, and the inequalities (7) for $S = T_j, j = 1, \dots, t$, and rounding down the coefficients yields the inequality (20), which is therefore valid for P_0 and P^* .

To prove that (20) is facet defining, one can show that the conditions of Theorem 7 apply. This can be done largely by using Corollaries 8 and 9 (see [65] for details).□

Figure 14.4 shows the support graph of a primitive SD inequality for P^* , with $|S| = 2, |D| = 1, |W| = 1, |H| = 3$ and $t = 2$. The condition of the Theorem requiring $|S| + |D| + t \geq 3$ is needed to eliminate instances like the invalid inequality (with $S = D = \emptyset, t = 1$)

$$x(H, H) + x(T_1, T_1) + y(H) \leq |H|,$$

violated by (\bar{x}, \bar{y}) such that $\bar{y}_i = 1$ for $i \in H \setminus \{i_1\}, \bar{y}_i = 0$ for all $i \in (N \setminus H) \cup \{i_1\}, \bar{x}_{i_1 e_1} = \bar{x}_{e_1 i_1} = 1$.

Theorems 10 and 11 are now seen to be implied by Theorem 12. Indeed, primitive comb inequalities are the special case of SD inequalities defined by $S = D = \emptyset$ and $t \geq 3$ odd, while primitive odd CAT inequalities are the special case defined by $|S| = |D| \geq 1$, $t \geq 1$ odd.

A *clique tree* C of a digraph G is a strongly connected subgraph of G induced by two collections of cliques (maximal complete digraphs), H_i , $i = 1, \dots, h$ and T_j , $j = 1, \dots, t$, called *handles* and *teeth*, respectively, satisfying the following conditions: (i) the teeth are pairwise disjoint; (ii) the handles are pairwise disjoint; (iii) every handle intersects an odd number, greater than or equal to three, of teeth; (iv) every tooth has at least one handle-free node (i.e. node not contained in any handle); (v) every nonempty intersection of a handle with a tooth is an articulation set of C .

Given a clique tree C as defined above, the inequality

$$\sum_{i=1}^h x(H_i, H_i) + \sum_{j=1}^t x(T_j, T_j) \leq \sum_{i=1}^h |H_i| + \sum_{j=1}^t (|T_j| - t_j) - (t + 1)/2, \tag{21}$$

where t_j denotes the number of handles intersecting T_j , is called a *clique tree inequality*. These inequalities were introduced for the symmetric TS polytope by Grötschel and Pulleyblank [406], who showed that they are facet inducing for that polytope. Fischetti [295] showed that for $n \geq 7$ they are also facet inducing for the asymmetric TS polytope.

The class of clique tree inequalities has been generalized to P_0 and P^* [65]. Again, we will restrict ourselves here to the primitive members of the class. A clique tree C , and the corresponding inequality, is primitive if (vi) every tooth has exactly one handle-free node; (vii) every nonempty intersection of a handle and a tooth contains exactly one node; (viii) every handle has at most one node not contained in any tooth; (ix) G has at most one node not contained in C .

A clique tree in G_L , the digraph with loops, is defined in the same way as a clique tree in the corresponding digraph without loops, except that every node contained in a handle has a loop (see Figure 14.5). A clique tree in G_L with a single handle is a comb in G_L . It follows that a clique tree inequality for P_0 , just like a comb inequality for P_0 , differs from the corresponding inequality for P in that it contains the loop variables y_k with coefficient $\beta_k = 1$ for every node of every handle.

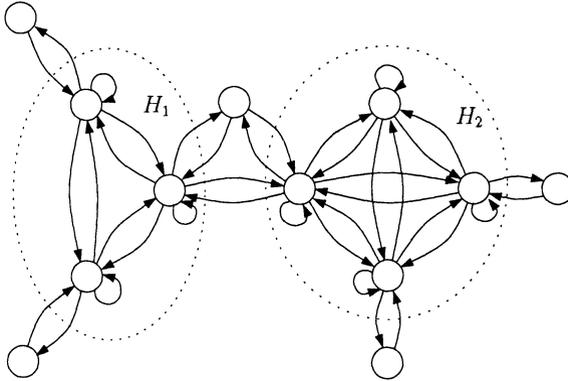


Figure 14.5. Primitive clique tree for P_0 with two handles.

Theorem 13 *The primitive clique tree inequality*

$$\sum_{i=1}^h x(H_i, H_i) + \sum_{j=1}^t x(T_j, T_j) + \sum_{i=1}^h y(H_i) \leq \sum_{i=1}^h |H_i| + \sum_{j=1}^t (|T_j| - t_j) - (t + 1)/2 \tag{22}$$

is valid and, for $n \geq 7$, facet defining for P_0 and P^* .

For the proof of this theorem, which uses Theorem 7 and its Corollaries, the reader is referred to [65].

Finally, we address the class of facet defining inequalities for the ATS polytope known as lifted cycle inequalities. For any directed cycle C , the cycle inequality

$$\sum_{(i,j) \in C} x_{ij} \leq |C| - 1$$

is valid for the ATS polytope P ; and its liftings, of the form

$$\sum_{(i,j) \in C} x_{ij} + \sum_{(i,j) \in K} \alpha_{ij} x_{ij} \leq |C| - 1,$$

where K is the set of chords of C , are often facet defining for P . Two known classes of lifted cycle inequalities for P are the D_k^+ and D_k^- in-

equalities [396, 402], where k is any integer satisfying $3 \leq k \leq n - 2$:

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2x(\{i_1\}, \{i_3, \dots, i_k\}) + \sum_{j=4}^k x(\{i_j\}, \{i_3, \dots, i_{j-1}\}) \leq k - 1 \tag{23}$$

and

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2x(\{i_2, \dots, i_{k-1}\}, \{i_1\}) + \sum_{j=3}^{k-1} x(\{i_j\}, \{i_2, \dots, i_{j-1}\}) \leq k - 1. \tag{24}$$

It is easy to see that the only clones with respect to (23) and (24) are the isolated nodes i_{k+1}, \dots, i_n . We will now show how to generalize these inequalities to the polytopes P_0 and P^* .

Theorem 14 *For any $k \in \{3, \dots, n - 2\}$ and any $l \in \{k + 1, \dots, n\}$, the lifted cycle inequality*

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2x(\{i_1\}, \{i_3, \dots, i_k\}) + \sum_{j=4}^k x(\{i_j\}, \{i_3, \dots, i_{j-1}\}) + y_{i_1} + \sum_{j=3}^k y_{i_j} - y_{i_l} \leq k - 1 \tag{25}$$

defines a facet of P_0 .

Furthermore, if $\sum_{i \in T} w_i \leq U$ for $T = \{i_1, \dots, i_k\}$ and for $T = \{i_{k+1}, \dots, i_n\}$, then (25) defines a facet of P^* .

We illustrate the inequalities (25) for $n = 5, k = 3$ and for $n = 7, k = 5$ through their support graphs shown in Figure 14.6(a) and (b). Here the single and double lines represent coefficients of 1 and 2, respectively, as before, and the shaded lines represent coefficients of -1. Clearly, for $k = n - 2$ the inequality (25) is primitive.

The reader may notice the similarity between the pattern of loop coefficients for these lifted cycle inequalities, and for the subtour elimination inequalities (7). In both cases, all but one of the cycle nodes has a loop coefficient of one (with the remaining node having a loop coefficient of 0), and exactly one non-cycle node has a loop coefficient of -1. But whereas

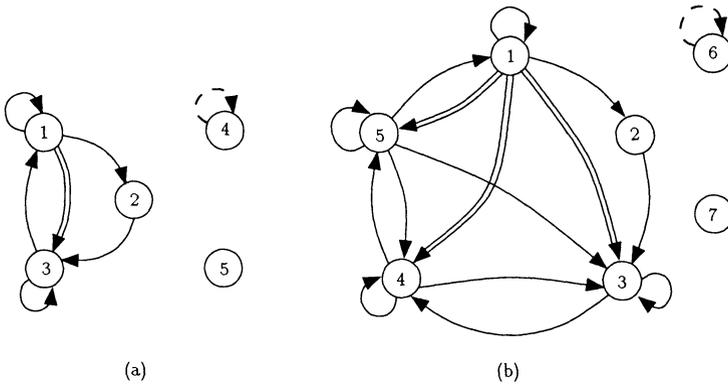


Figure 14.6. Primitive lifted cycle inequalities for P_0 .

in the case of the inequalities (7) the cycle node with loop coefficient of 0 can be any node of the cycle, in the case of the inequalities D_k^+ , D_k^- , the cycle node with loop coefficient equal to zero is the unique node whose indegree and outdegree in the support graph of the inequality is exactly one.

6. Cloning and Clique Lifting for the PCTSP.

In the previous sections we have examined the problem of extending to P_0 and P^* primitive facet defining inequalities for the ATS polytope P . In this section we extend to P_0 and P^* the cloning operation itself and use it to derive more general classes of facet defining inequalities for these polytopes.

When dealing with P , P_0 and P^* , we will have to specify the order of the digraphs G and G_L on which they are defined. So we will use G^n and G_L^n to denote the digraphs $G = (N, A)$ and $G_L = (N, A \cup L)$, respectively, on n nodes, and we will denote $P(G^n)$, $P_0(G_L^n)$ and $P^*(G_L^n)$ the corresponding polytopes. G^{n+1} and G_L^{n+1} will denote the digraphs obtained from G^n and G_L^n , respectively, by adding the node $n+1$, the arcs $(i, n+1)$, $(n+1, i)$, $i \in N$, and, in the case of G_L^{n+1} , the loop $(n+1, n+1)$. As mentioned earlier, a facet defining inequality for the ATS polytope is called regular unless it is of the form $x_{ij} \geq 0$ or $x_{ij} + x_{ji} \leq 1$.

Theorem 15 *Let $\alpha x + \beta y \leq \alpha_0$ be a facet defining inequality for $P_0(G_L^n)$, whose restriction to $P(G^n)$ is a regular facet defining inequality for $P(G^n)$. For an arbitrary but fixed $k \in N$, let*

$$\delta_k := \max\{\alpha_{ik} + \alpha_{kj} - \alpha_{ij} : i, j \in N \setminus \{k\}, i \neq j\}, \quad (26)$$

and define $(\tilde{\alpha}, \tilde{\beta}) \in \mathbb{R}^{(n+1)^2}$, $\tilde{\alpha}_0 \in \mathbb{R}$ by

$$\tilde{\alpha}_{ij} := \alpha_{ij} \text{ for all } (i, j) \in A, \tilde{\beta}_i := \beta_i \text{ for all } i \in N \tag{27}$$

$$\tilde{\alpha}_{i,n+1} := \alpha_{ik} \text{ and } \tilde{\alpha}_{n+1,i} := \alpha_{ki} \text{ for all } i \in N \setminus \{k\} \tag{28}$$

$$\tilde{\alpha}_{k,n+1} = \tilde{\alpha}_{n+1,k} := \delta_k \tag{29}$$

$$\tilde{\beta}_{n+1} := \delta_k \tag{30}$$

$$\tilde{\alpha}_0 := \alpha_0 + \delta_k. \tag{31}$$

Then the inequality $\tilde{\alpha}x + \tilde{\beta}y \leq \tilde{\alpha}_0$ defines a facet of $P_0(G_L^{n+1})$ and if $w_i \leq U - w_{n+1}$ for all $i \in N$, a facet of $P^*(G_L^{n+1})$.

Proof outline. In [65], the validity of the lifted inequality $\tilde{\alpha}x + \tilde{\beta}y \leq \tilde{\alpha}_0$ is shown by contradiction, through a complex case by case argument. As to the facet defining property, the proof uses the fact, not hard to show, that the restriction of $\tilde{\alpha}x + \tilde{\beta}y \leq \tilde{\alpha}_0$ to $P(G^{n+1})$ defines a facet of $P(G^{n+1})$.□

The cloning procedure defined in Theorem 15 differs from the corresponding procedure for the ATS polytope in one important respect: In the case of the PCTS polytope, when a node is cloned, the arcs incident with the clones get the same coefficients as the corresponding arcs incident with the original node, but the loops of the clones may get coefficients different from the loop of the original node. Thus while the clones of a given initial node are identical among themselves, they may differ from the original in the coefficient of their loop. It seems therefore more appropriate to call them *quasi-clones*. We will give a few examples. Consider the primitive odd CAT inequality shown in Figure 14.3(a), whose restriction to the ATS polytope P is known as T_2 :

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + x_{33} \leq 2$$

Applying the cloning procedure to node 4 produces the inequality

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + x_{3,4a} + x_{4a,3} + x_{4,4a} + x_{4a,4} + x_{33} + x_{4a,4a} \leq 3$$

whose support graph is shown in Figure 14.7(a). The loop of node 4 has a coefficient of 0, whereas the loop of its quasi-clone 4a has a coefficient of 1. Any additional quasi-clones of node 4 would also get a coefficient of 1 for their loop. On the other hand, cloning node 3 instead of 4 produces the inequality

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + x_{1,3a} + x_{3a,2} + x_{3a,4} + x_{4,3a} + 2x_{3,3a} + 2x_{3a,3} + x_{3,3} + 2x_{3a,3a} \leq 2 + 2 = 4,$$

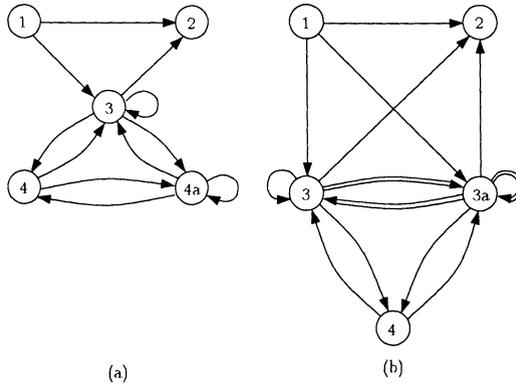


Figure 14.7. Cloning of two nodes of T_2

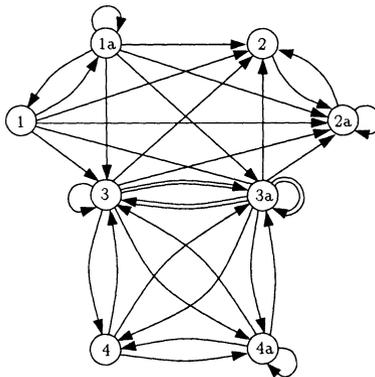


Figure 14.8. Cloning every node of T_2

whose support graph is shown in Figure 14.7(b). Here the loop of the original node 3 has coefficient 1, whereas the loop of its quasi-clone 3a has coefficient 2.

Figure 14.8 shows the result of cloning (once) every node with respect to the same inequality whose restriction to P is T_2 . Here the loops of nodes 1, 2 and 4 have coefficients equal to 0, but the loops of their quasi-clones 1a, 2a and 4a have coefficients equal to 1. Further, the loop of node 3 has a coefficient of 1, whereas that of its quasi-clone 3a has a coefficient of 2. The inequality is

$$\begin{aligned}
 &x_{12} + x_{1,2a} + x_{13} + x_{1,3a} + x_{1a,2} + x_{1a,2a} + x_{1a,3} + x_{1a,3a} + x_{1,1a} \\
 &\quad + x_{1a,1} + x_{1a,1a} + x_{2,2a} + x_{2a,2} + x_{2a,2a} + x_{3,2} + x_{3,2a} + x_{3,4} + x_{3,4a} \\
 &\quad + x_{3a,2} + x_{3a,2a} + x_{3a,4} + x_{3a,4a} + 2x_{3,3a} + 2x_{3a,3} + x_{3,3} + 2x_{3a,3a} \\
 &\quad + x_{4,3} + x_{4,3a} + x_{4a,3} + x_{4a,3a} + x_{4,4a} + x_{4a,4} + x_{4a,4a} \\
 &\leq 2 + 1 + 1 + 2 + 1 = 7.
 \end{aligned}$$

Cloning with respect to a given inequality can of course be applied repeatedly to the same node (or recursively, to its quasi-clones, which yields the same result). *Clique-lifting* consists of doing just that; but whereas in the case of the ATS polytope clique lifting is defined as replacing a node by a clique (of its clones), in the case of the PCTS polytope it seems more appropriate to talk about adding to a node a clique (of its quasi-clones), since the members of the added clique will be exact copies of each other with respect to the inequality in question, but may differ from the original node in the coefficient of their loop.

Next we derive the general form of an SD inequality for the PCTS polytope obtained by clique-lifting every node of the primitive SD inequality introduced in the previous section.

We denote by K_i^S and K_j^D the cliques added to nodes $i \in S$ and $j \in D$, respectively; by K_i^I , K_j^E , K^W and K^Q the cliques added to nodes $i \in I$, $j \in E$, $w \in W$ and $q \in Q$, respectively (with $|I| = |E| = t$); and, finally, by

$$H^* := (W \cup K^W) \cup (\cup_{k=1}^t (\{i_k\} \cup K_k^I)) \quad (\text{where } K^W = \emptyset \text{ if } W = \emptyset),$$

$$T_k^* := (\{i_k\} \cup K_{i_k}^I) \cup (\{e_k\} \cup K_{e_k}^E), \quad k = 1, \dots, t,$$

$$S^* := \cup_{i \in S} (\{i\} \cup K_i^S) \quad \text{and}$$

$$D^* := \cup_{j \in D} (\{j\} \cup K_j^D).$$

We then have the general SD inequality

$$\begin{aligned}
 &x(S^* \cup H^*, D^* \cup H^*) + \sum_{k=1}^t x(T_k^*, T_k^*) + \sum_{i=1}^{|S|} x(\{i\} \cup K_i^S, \{i\} \cup K_i^S) \\
 &+ \sum_{j=1}^{|D|} x(\{j\} \cup K_j^D, \{j\} \cup K_j^D) + y_w + y(K^W) + \sum_{k=1}^t y_{i_k} \\
 &+ 2 \sum_{k=1}^t y(K_{i_k}^I) + \sum_{k=1}^t y(K_{e_k}^E) + \sum_{i=1}^{|S|} y(K_i^S) + \sum_{j=1}^{|D|} y(K_j^D) \quad (32) \\
 &\leq |S^*| - |S| + |D^*| - |D| + |H^*| + \sum_{k=1}^t (|T_k^*| - 1) + \\
 &\qquad\qquad\qquad \frac{1}{2}(|S| + |D| - t - 1).
 \end{aligned}$$

Figure 14.9 shows the support graph of an SD inequality obtained from the primitive SD inequality of Figure 14.4 by clique-lifting nodes 1, 6 and 7. Here the arcs drawn inside the cliques apply to all pairs of nodes of the clique, and the loops drawn outside the cliques apply to every node of the clique.

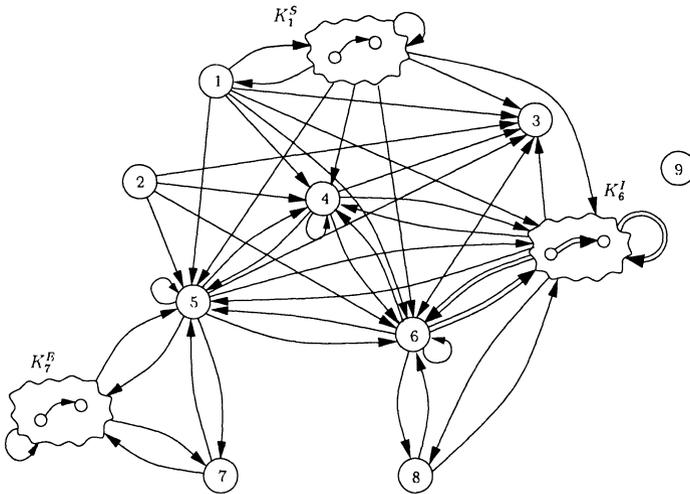


Figure 14.9. SD inequality obtained by clique-lifting

In Section 5 we mentioned that the primitive SD inequalities properly generalize the primitive comb and odd CAT inequalities. We can now add that the lifted SD inequality (31) properly generalizes not only the

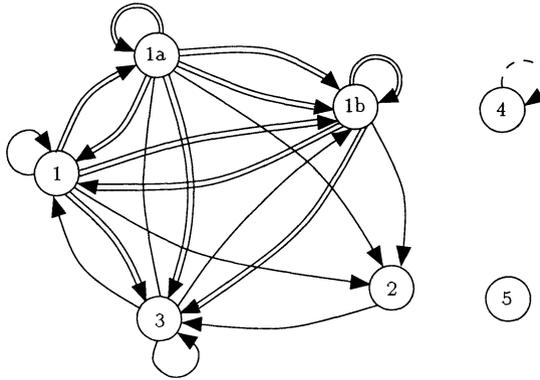


Figure 14.10. Lifted 3-cycle inequality with two quasi-clones

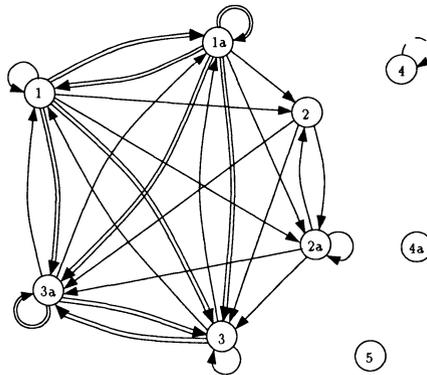


Figure 14.11. Lifted 3-cycle inequality with one quasi-clone for every node

lifted comb and odd CAT inequalities, but many other inequalities that define facets of P_0 and P^* , and that can be obtained from the corresponding facet defining inequalities for the ATS polytope P . Thus, inequalities for P_0 and P^* corresponding to the T_k inequalities for P [396] can be obtained from a primitive SD inequality with $|S| = |D| = t = 1$, $W = \emptyset$, by cloning (possibly repeatedly) the nodes $e_1 \in E$ and $q \in Q$. Inequalities corresponding to the simple FDA (fixed-outdegree one-arborescence) inequalities for P introduced in [73], can be derived by cloning from the odd CAT inequality with $|T| = 5$ shown in Figure 14.3(a), which is equivalent to the (unique) primitive simple FDA inequality. Inequalities for P_0 and P^* corresponding to the C_2 inequalities for P [396] can be derived from the simple SD inequality with $|S| = |D| = 1$ and $t \geq 3$, by cloning nodes in $N \setminus (S \cup D)$.

Cloning applied to clique tree inequalities produces structures similar to those illustrated on comb and SD inequalities. As to the lifted cycle inequalities, we illustrate the results of cloning on the lifted 3-cycle inequality of Figure 14.6(a). Adding two quasi-clones to node 1 gives rise to the inequality whose support is shown in Figure 14.10 and whose righthand side is 6, whereas adding just one quasi-clone to every node yields the inequality with the support shown in Figure 14.11 and with a righthand side of 7.

7. A Projection: The Cycle Polytope

Any problem that can be represented on the digraph with loops G_L , can also be represented on the simple digraph $G = (N, A)$, and this latter representation can be obtained from the former one by projecting out the loop variables. In this case the cycle and loops polytope P_0 gets replaced by the *cycle polytope* P_C of G , defined as the convex hull of incidence vectors of simple directed cycles of G . Notice that minimizing a linear function over the cycle polytope P_C , i.e. finding the shortest cycle in a directed graph with arbitrary arc lengths is an NP-complete task, as it subsumes the asymmetric traveling salesman (ATS) problem as a special case; but it can be solved in $O(n^3)$ time if the cost coefficients are restricted to exclude the occurrence of negative-length dicycles. Indeed, for cost functions satisfying the above condition, finding a shortest path from j to i and adding to it the length of (i, j) yields the shortest cycle containing (i, j) . Doing this for every arc in the role of (i, j) and choosing the shortest among the cycles found solves the problem.

The cycle polytope of a directed graph was studied in [79], whose main results are summarized in this section. The symmetric counterpart of

P_C , i.e. the cycle polytope of an undirected graph, has been studied in [227] and [92].

In the previous sections we exploited the relationship between the cycle and loops polytope P_0 and the ATS polytope P . There is also a direct relationship between the cycle polytope P_C and the ATS polytope, i.e. a connection in the space of arc variables (without recourse to loop variables and hence to P_0). Indeed, if $P_{C\parallel}$ denotes the convex hull of incidence vectors of simple directed k -cycles (cycles of length k), then the ATS polytope is $P_{C\setminus}$, and the cycle polytope is $P_C = P_{\cup_{k=2}^n C\parallel}$. Thus the ATS polytope is contained in P_C ; it is in fact the restriction of P_C to the hyperplane defined by $x(N, N) = n$. In the sequel we assume that the digraph G is complete.

Theorem 16 *The cycle polytope P_C of G is the convex hull of points $x \in \{0, 1\}^A$ satisfying the system*

$$x(i, N) \leq 1 \quad i \in N \tag{33}$$

$$x(N, i) - x(i, N) = 0 \quad i \in N \tag{34}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n x_{ij} \geq 1 \tag{35}$$

$$x(k, N) + x(\ell, N) - x(S, N \setminus S) \leq 1 \quad \forall S \subset N, 2 \leq |S| \leq n - 2, k \in S, \ell \in N \setminus S \tag{36}$$

Proof. Constraints (33) and (34) are satisfied by the incidence vectors of all unions of node-disjoint simple dicycles and the zero vector. Constraint (35) excludes the zero vector by requiring the solution-subgraph to contain at least one arc (i, j) such that $i < j$. Clearly, any dicycle and therefore any union of dicycles, has this property. Conversely, any $x \in \{0, 1\}^A$ that satisfies (33), (34), (35) defines a nonempty union of simple dicycles in G .

Constraints (36) are satisfied by all simple dicycles, but violated by unions of dicycles with more than one member. Indeed, let C be a simple dicycle and x its incidence vector. If C contains both k and ℓ , it must also contain an arc of the cutset $(S, N \setminus S)$ that separates k from ℓ , and thus x satisfies the corresponding inequality (36); while the remaining inequalities (36) are trivially satisfied. Now let x be the incidence vector of a union of several disjoint cycles. Let C_1 and C_2 be among these. Choose S such that $C_1 \subseteq (S, S)$ and $C_2 \subseteq (N \setminus S, N \setminus S)$. Then there exists $k \in S$ and $\ell \in N \setminus S$ such that $x(k, N) = 1$, $x(\ell, N) = 1$ and $x(S, N \setminus S) = 0$, i.e. x violates (36). \square

It is not hard to see that the polyhedron defined by (33), (34), (36) and $x \geq 0$ is the projection into the x -space of the polyhedron defined by (2), (7) and $x \geq 0, y \geq 0$. We then have

Corollary 17 P_C is the projection of P_0 into the x -space.

The following two results of Balas and Oosten [78], relating the dimension of polyhedra to that of their projections, were instrumental in analyzing the structure of P_C .

Consider a polyhedron

$$Q := \{(u, x) \in \mathbb{R}^p \times \mathbb{R}^q : Au + Bx \leq b\},$$

where (A, B, b) has m rows, and the projection of Q into the x -space (the subspace of x):

$$\text{Proj}_x(Q) := \{x \in \mathbb{R}^q : \exists u \in \mathbb{R}^p \text{ with } (u, x) \in Q\}.$$

Let $A^=u + B^=x = b^=$ denote the equality subsystem of Q , i.e. the set of equations corresponding to those inequalities of $Au + Bx \leq b$ satisfied at equality by every $x \in Q$, and for a matrix M , let $\text{rank}(M)$ denote the rank of M .

Dimension Theorem [78].

$$\dim(\text{Proj}_x(Q)) = \dim(Q) - p + \text{rank}(A^=). \quad \square$$

Now let $\alpha u + \beta x \leq \beta_0$ be a valid inequality defining a facet F of Q , i.e.

$$F := \{(u, x) \in Q : \alpha u + \beta x = \beta_0\},$$

and let $(\alpha_A)^=u + (\beta_B)^=x = (\beta_0)^=$ be the equality subsystem of F .

Facet Theorem [78]. $\text{Proj}_x(F)$ is a facet of $\text{Proj}_x(Q)$ if and only if $\text{rank}((\alpha_A)^=) = \text{rank}(A^=)$. \square

Since in the case of P_0 $\text{rank}(A^=) = p$, it follows immediately that $\dim P_C = \dim P_0 = (n - 1)^2$. Further, the inequalities (33), (35), (36), as well as $x \geq 0$, can all be shown to be facet defining for P_C . Moreover, in the case of the inequalities (35), this is true for any permutation of N ; i.e., in fact there are $n!$ inequalities of the form (35) defining facets of P_C , although any one of them suffices for a valid formulation.

More generally, unlike in the case of a general polyhedron projected onto an arbitrary subspace, in our case the facets of P_0 project into facets of P_C .

Theorem 18 *Let $\alpha x + \beta y \leq \alpha_0$ be a facet defining inequality for the cycle-and-loops polytope P_0 . Then the inequality*

$$\sum_{i \in N} \sum_{j \in N \setminus \{i\}} (\alpha_{ij} - \beta_i) x_{ij} \leq \alpha_0 - \sum_{i \in N} \beta_i, \tag{37}$$

which is the projection of $\alpha x + \beta y \leq \alpha_0$ onto the subspace of x , defines a facet of P_C .

Notice that the facets of P_0 are derived from facets of the ATS polytope by specialized lifting procedures for the loop variables. Thus P_0 constitutes a “bridge” through which every facet defining inequality for the ATS polytope has its counterpart for the cycle polytope.

Thus, substituting for the loop variables y_i one of the expressions obtained from (2) into the inequality (7) yields (36). Substituting the same expression into, say, the primitive SD-inequality (20), yields the inequality

$$x(S, D \cup H) + \sum_{j=1}^s x(T_j, T_j) - x(H, N \setminus (D \cup H)) \leq (|S| + |D| + s - 1) / 2, \tag{38}$$

valid and facet defining for the cycle polytope P_C . By the same procedure one can obtain the counterparts in the x -space of all the other facet defining inequalities for P_0 , which all define facets of P_C .

In section 6 we have shown how the cloning (clique lifting) procedure for the ATS polytope can be generalized to a corresponding procedure for the PCTS polytope P_0 . As to the cycle polytope P_C , in order to find the facet inducing inequality that corresponds to a given (arbitrary) facet defining inequality $\alpha x \leq \alpha_0$ for the ATS polytope $P(G)$ defined on the digraph G , one can proceed as follows:

- 1 Find the primitive inequality $\tilde{\alpha} x \leq \tilde{\alpha}_0$ for $P(\tilde{G})$, the ATS polytope defined on some induced subgraph \tilde{G} of G , from which $\alpha x \leq \alpha_0$ can be derived by cloning (clique lifting).
- 2 Derive from $\tilde{\alpha} x \leq \tilde{\alpha}_0$ the corresponding primitive facet inducing inequality $\tilde{\alpha} x + \tilde{\beta} y \leq \tilde{\alpha}_0$ for $P_0(\tilde{G}_L)$ defined on the digraph \tilde{G}_L with loops.
- 3 Apply the cloning procedure to $\tilde{\alpha} x + \tilde{\beta} y \leq \tilde{\alpha}_0$ to obtain a facet inducing inequality $\alpha x + \beta y \leq \alpha_0$ for $P_0(G_L)$.
- 4 Substitute for y to obtain the corresponding facet inducing inequality $\hat{\alpha} x \leq \hat{\alpha}_0$ for $P_C(G)$, the cycle polytope of G .

We illustrate the procedure on an example.

Example 19 Consider the inequality

$$x_{12} + x_{13} + x_{15} + x_{32} + x_{34} + x_{43} + x_{45} + x_{52} + x_{54} + 2x_{35} + 2x_{53} \leq 4 \quad (E.1)$$

whose support graph is shown in Figure 14.12(a). This graph has an arc (i, j) drawn in a plain line if $\alpha_{ij} = 1$, an arc (i, j) drawn in a double line if $\alpha_{ij} = 2$, and no arc (i, j) if $\alpha_{ij} = 0$. (E.1) is valid for the ATS polytope P on $n \geq 5$ nodes.

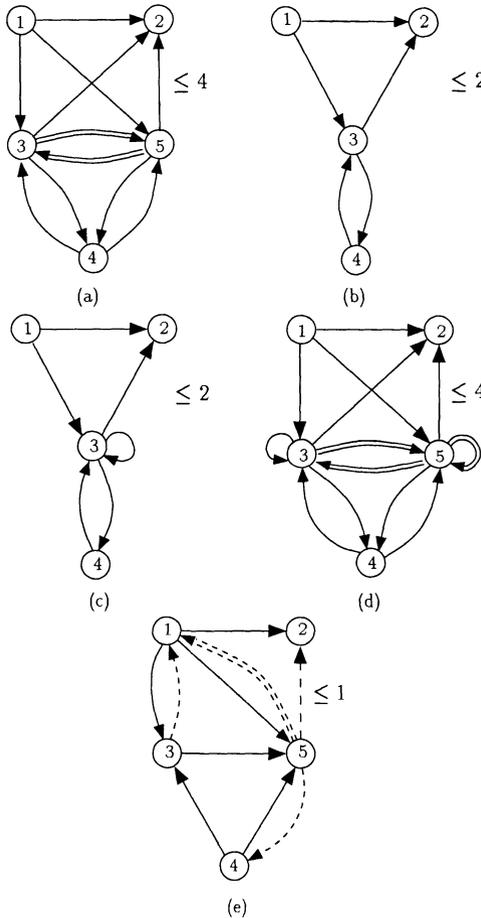


Figure 14.12. Illustration of Example 19

It is not hard to see that nodes 3 and 5 are clones, in that $\alpha_{i3} = \alpha_{i5}$ and $\alpha_{3i} = \alpha_{5i}$ for all i , $\alpha_{35} = \alpha_{53} = \max \{ \alpha_{i3} + \alpha_{3j} - \alpha_{ij} : i, j \in$

$N \setminus \{3, 5\} = 2$, and deleting one of the two nodes, say 5, yields a valid inequality. Indeed, the primitive inequality from which (E.1) can be obtained by cloning is the odd CAT inequality with $|T| = 5$, also known as T_2 , whose support graph is shown in Figure 14.12(b):

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} \leq 2 \tag{E.2}$$

This inequality is known to be facet defining for the ATS polytope P defined on the complete digraph with 4 nodes. The corresponding inequality for P_0 defined on the complete digraph with loops on 4 nodes, illustrated in Figure 14.12(c), is

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + y_3 \leq 2. \tag{E.3}$$

Applying the cloning procedure of section 6 to node 3 yields the inequality whose support graph is shown in Figure 14.12(d):

$$x_{12} + x_{13} + x_{15} + x_{32} + x_{34} + x_{43} + x_{45} + x_{52} + x_{54} + 2x_{35} + 2x_{53} + y_3 + 2y_5 \leq 4 \tag{E.4}$$

Note that node 3 has loop coefficient 1, whereas its quasi-clone 5 has loop coefficient 2. Finally, substituting for y_3 and y_5 we obtain the inequality whose support graph is shown in Figure 14.12(e), with negative coefficients drawn in dotted lines:

$$x_{12} + x_{13} + x_{15} + x_{35} + x_{43} + x_{45} - x_{31} - x_{52} - x_{54} - 2x_{51} \leq 1 \tag{E.5}$$

Next we describe a more general lifting procedure for the cycle polytope P_C that is not based on its connection with P_0 .

Let $\alpha x \leq \alpha_0$ be a valid inequality that defines a proper face F_α of $P_C(G^n)$. For our purposes here, a lifting of $\alpha x \leq \alpha_0$ is an inequality

$$\alpha x + \sum_{i=1}^n \beta_i x_{i,n+1} + \sum_{j=1}^n \gamma_j x_{n+1,j} \leq \alpha_0 \tag{39}$$

which is valid. Notice that (39) defines a face F of $P_C(G^{n+1})$ such that $F_\alpha \subseteq F$. The set of all vectors $(\beta, \gamma) \in \mathbb{R}^{n+n}$ for which (39) is valid will be called the *lifting set* of $\alpha x \leq \alpha_0$, and denoted L_α . The following theorem, which is a special case of a more general result by Oosten [634], introduces a relatively simple explicit representation of L_α for the case when $\alpha x \leq \alpha_0$ defines a nontrivial facet of $P_C(G^n)$. (We call trivial the facets of $P_C(G^n)$ defined by the inequalities $x_{ij} \geq 0$; all other facets are nontrivial.)

Theorem 20 *Let*

$$L = \left\{ (\beta, \gamma) \in \mathbb{R}^{n+n} \left| \begin{array}{l} \beta_i + \gamma_j \leq \alpha_{ij} \quad \forall (i, j) \in A \\ \beta_i + \gamma_i \leq \alpha_0 \quad \forall i \in N \end{array} \right. \right\}$$

Then we have:

- (i) *if $\alpha x \leq \alpha_0$ is valid for $P_C(G^n)$, then $L \subseteq L_\alpha$; and*
- (ii) *if, in addition, $\alpha x \leq \alpha_0$ defines a nontrivial facet of $P_C(G^n)$, then $L = L_\alpha$. Furthermore, in this case*
 - (ii a) *L_α is full dimensional*
 - (ii b) *If $(\beta, \gamma) \in L_\alpha$, then $(\beta + ce, \gamma - ce) \in L_\alpha$ for all $c \in \mathbb{R}$ with $e := (1, \dots, 1)$; and*
 - (ii c) *The extreme faces (faces of lowest dimension) of L_α have dimension 1.*

The usefulness of the concept of a lifting set becomes clear in light of the following very general result.

Theorem 21 *Let $\alpha x \leq \alpha_0$ be valid and facet defining for $P_C(G^n)$. Then a lifting (39) of $\alpha x \leq \alpha_0$ defines a facet of $P_C(G^{n+1})$ if and only if (β, γ) is a one-dimensional face of the lifting set L_α .*

This characterization leads to an efficient procedure for generating all the facet defining liftings of an inequality $\alpha x \leq \alpha_0$. Unlike in the general case, where some lifted facets cannot be generated sequentially, in the case of the cycle polytope all facet defining inequalities of the form (39) can be obtained from $\alpha x \leq \alpha_0$ by sequential lifting, at a computational cost of $O(n^2)$ for each inequality. The cloning procedure discussed in the previous section is one special instance of this more general sequential lifting procedure, which subsumes all valid liftings.

Theorem 22 *Let $\alpha x \leq \alpha_0$ be valid and facet defining for $P_C(G^n)$. Then a lifting (39) of $\alpha x \leq \alpha_0$, with $\gamma_p = 0$ for some arbitrary but fixed p , defines a facet of $P_C(G^{n+1})$ if and only if there exists an ordering $\pi := (\pi_1, \dots, \pi_{2n})$ of the components of (β, γ) , with $\pi_1 = \gamma_p$, $\pi_2 = \beta_i$ for some i , and for $k = 2, \dots, 2n$,*

- (i) *if $\pi_k = \beta_i$, then*

$$\beta_i = \min_j \{ \alpha_{ij} - \gamma_j : \gamma_j = \pi_\ell \text{ for some } \ell < k \}$$

(ii) if $\pi_k = \gamma_j$, then

$$\gamma_j = \min_i \{ \alpha_{ij} - \beta_i : \beta_i = \pi_\ell \text{ for some } \ell < k \}$$

where $\alpha_{ii} = \alpha_{jj} = \alpha_0$.

Proof. The system defining L_α is the dual of the constraint set of an assignment problem, whose bases are known to be triangular modulo row and column permutations. Since the rank of the system is $2n - 1$ and there are $2n$ variables, we can fix the value of an arbitrarily chosen variable; this justifies our rule of setting $\gamma_p = 0$. The procedure for generating a basic solution of the system then consists of the sequence of steps described in the theorem. Each such sequence is easily seen to produce a basic solution satisfying at equality $2n - 1$ linearly independent inequalities. Different sequences may yield different subsystems that are satisfied at equality, hence different basic solutions, and all basic solutions are obtainable this way. Since (39) defines a facet of $P_C(G^{n+1})$ if and only if (β, γ) lies on a 1-dimensional face of L_α , i.e. if and only if (β, γ) is a basic solution of the system defining L_α , the theorem follows. \square

We illustrate Theorem 22 on the following

Example 23 Consider the inequality

$$x_{12} + x_{32} + x_{34} - x_{23} \leq 1 \tag{E.6}$$

which defines a facet of the cycle polytope on the digraph with 4 nodes. We want to lift this inequality into one for the cycle polytope on the digraph with 5 nodes. There are eight coefficients to be lifted, corresponding to the arcs $(i, 5)$, $(5, i)$, $i = 1, 2, 3, 4$; choosing $p = 1$ and setting the coefficient γ_1 of $(5, 1)$ to 0 leaves seven.

Suppose we choose the sequence $\beta_1, \dots, \beta_4, \gamma_2, \dots, \gamma_4$, corresponding to the arcs $(1, 5)$, \dots , $(4, 5)$, $(5, 2)$, \dots , $(5, 4)$ (since arc $(5, 1)$ has coefficient $\gamma_1 = 0$). Then $\pi_1 = \gamma_1 = 0$,

$$\begin{aligned} \pi_2 &= \beta_1 = \min_j \{ \alpha_{1j} - \gamma_j : \gamma_j = \pi_1 = 0 \} = \alpha_{11} (= \alpha_0) = 1. \\ \pi_3 &= \beta_2 = \min_j \{ \alpha_{2j} - \gamma_j : \gamma_j = \pi_1 = 0 \} = \alpha_{21} = 0 \\ \pi_4 &= \beta_3 = \alpha_{31} = 0 \\ \pi_5 &= \beta_4 = \alpha_{41} = 0 \\ \pi_6 &= \gamma_2 = \min_i \{ \alpha_{i2} - \beta_i : \beta_i = \pi_\ell \text{ for some } \ell < 6 \} \\ &= \min \{ 1 - 1, 1 - 0, 1 - 0, 0 - 0 \} = 0 \\ \pi_7 &= \gamma_3 = \min \{ 0 - 1, -1 - 0, 1 - 0, 0 - 0 \} = -1 \\ \pi_8 &= \gamma_4 = \min \{ 0 - 1, 0 - 0, 1 - 0, 1 - 0 \} = -1. \end{aligned}$$

So the resulting lifted inequality is

$$x_{12} + x_{32} + x_{34} - x_{23} + x_{15} - x_{53} - x_{54} \leq 1. \tag{E.7}$$

The inequality (E.7) can also be obtained by cloning, as follows:

The starting inequality (E.6), which is facet inducing for the cycle polytope P_C on G^4 , corresponds to the inequality

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + y_3 \leq 2 \tag{E.8}$$

which is facet inducing for the cycle-and-loops polytope P_0 on G^4 . (Indeed, (E.6) is obtained by setting $y_3 = 1 - x_{13} - x_{23} - x_{43}$ in (E.8).)

Cloning node 1 relative to (E.8) yields the inequality

$$x_{12} + x_{13} + x_{32} + x_{34} + x_{43} + x_{15} + x_{51} + x_{52} + x_{53} + y_3 + y_5 \leq 3, \tag{E.9}$$

facet inducing for P_0 defined on G^5 . Substituting now for $y_3 = 1 - \sum_i x_{i3}$ and $y_5 = 1 - \sum_j x_{5j}$ yields (E.7).

The remaining sequences yield the additional inequalities:

$$\begin{aligned} x_{12} + x_{32} + x_{34} - x_{23} - x_{45} + x_{52} - x_{53} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{53} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{15} - 2x_{25} - x_{45} + x_{52} + x_{53} + x_{54} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{15} - x_{25} - x_{45} + x_{52} + x_{54} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{25} - x_{45} + x_{52} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{15} - x_{25} + x_{54} &\leq 1 \\ x_{12} + x_{32} + x_{34} - x_{23} - x_{25} &\leq 1 \end{aligned}$$

These eight inequalities are all facet defining, and they define eight distinct facets. Each of them can be obtained by using many different sequences, their numbers for various inequalities being far from equal. Another property worth pointing out is that all these liftings yield the same sum for the lifted coefficients, namely -1.

We have seen how every regular facet defining inequality for the ATS polytope gives rise to one or several facet defining inequalities for the cycle polytope P_C . We now address the question as to what other inequalities are valid and facet defining for P_C , besides those that have their counterpart for the ATS polytope P . For one thing, all the inequalities coming from the ATS polytope are of the form $\alpha x \leq \alpha_0$, with $\alpha_0 > 0$. The relevant question to ask then, is: which facets of P_C are defined by inequalities of the form $\alpha x \leq \alpha_0$ with $\alpha_0 \leq 0$? The concluding result of this survey provides the somewhat surprising answer:

Theorem 24 Any inequality $\alpha x \leq \alpha_0$ with $\alpha_0 = 0$ that is valid and facet defining for P_C is equivalent to one of the inequalities $x_{ij} \geq 0$, $(i, j) \in A$.

Theorem 25 Any inequality $\alpha x \leq \alpha_0$ with $\alpha_0 < 0$ that is valid and facet defining for P_C is equivalent to one of the inequalities

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n x_{\pi(i),\pi(j)} \geq 1$$

where π is some permutation of the node set N .

For proofs and details, see [79].

Acknowledgements. This research was supported by the National Science Foundation through grant DMI-9802773 and by the Office of Naval Research through contract N00014-97-1-0196.

Chapter 15

THE BOTTLENECK TSP

Santosh N. Kabadi

Faculty of Administration

University of New Brunswick-Fredericton

New Brunswick, Canada

kabadi@unb.ca

Abraham P. Punnen

Department of Mathematical Sciences

University of New Brunswick-Saint John

New Brunswick, Canada

punnen@unbsj.ca

1. Introduction

In this chapter we study the *bottleneck traveling salesman problem* (BTSP) introduced in Chapter 1. BTSP is a variation of the classical traveling salesman problem (TSP) that differs from the TSP only in the objective function. Let us first restate the problem.

Let $G = (N, E)$ be a (directed or undirected) graph on node set $N = \{1, 2, \dots, n\}$ and let \mathbb{F} be the family of all Hamiltonian cycles (tours) in G . For each edge $e \in E$, a cost c_e is prescribed. For any $\mathcal{H} \in \mathbb{F}$, let $c_{\max}(\mathcal{H}) = \max\{c_e : e \in \mathcal{H}\}$. Then the BTSP is to find a Hamiltonian cycle $\mathcal{H} \in \mathbb{F}$ such that $c_{\max}(\mathcal{H})$ is as small as possible.

Without loss of generality we replace G by K_n in the undirected case and \vec{K}_n in the directed case by adding the missing edges with cost ∞ . Thus, the edge costs will be given in the form of an $n \times n$ matrix C , called the cost matrix, where any non-diagonal entry c_{ij} corresponds to the cost c_e of the edge $e = (i, j)$. As indicated in Chapter 1 and Chapter 11, we can also represent a tour in \vec{K}_n as a cyclic permutation γ on $N = \{1, 2, \dots, n\}$. Let $c_{\max}(\gamma) = \max\{c_{i, \gamma(i)} : i \in N\}$. Then the BTSP is to find a tour γ^* on N such that $c_{\max}(\gamma^*) = \min\{c_{\max}(\gamma) : \gamma \text{ is}$

a tour on N }. When the underlying cost matrix needs to be emphasized, we sometimes denote the BTSP with cost matrix C as BTSP(C). The BTSP can also be formulated as an integer programming problem:

$$\begin{aligned} \text{Minimize} \quad & \max\{c_{ij}x_{ij}, 1 \leq i, j \leq n, i \neq j\} & (1) \\ \text{Subject to} \quad & & (2) \\ & \sum_{i=1}^n x_{ij} = 1, j \in N & (3) \\ & \sum_{j=1}^n x_{ij} = 1, i \in N & (4) \\ & x_{ij} = 0 \text{ or } 1 & (5) \\ & \sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 1 \quad \forall S \subset N, & (6) \end{aligned}$$

where $\bar{S} = N \setminus S$.

In fact each of the (mixed) integer programming formulations of TSP discussed in Chapter 1 leads to a corresponding (mixed) integer programming formulation of the BTSP.

To the best of our knowledge, the bottleneck TSP was introduced by Gilmore and Gomory [360] assuming special structure of elements of C . Garfinkel and Gilbert [349] considered the general BTSP model and discussed an application of the problem in the context of machine scheduling. Meaningful interpretations of the BTSP model and its variations can be given in the context of some route planning problems and transportation of goods perishable in time.

Another problem closely related to the BTSP is the *maximum scatter traveling salesman problem* (MSTSP) [33]. Here, each edge e in G is assigned a weight w_e and we seek a tour \mathcal{H} in G such that the smallest edge weight in \mathcal{H} is as large as possible. Let $w_{\min}(\mathcal{H}) = \min\{w_e : e \in \mathcal{H}\}$. Then, the MSTSP is to find a Hamiltonian cycle $\mathcal{H} \in \mathbb{F}$ such that $w_{\min}(\mathcal{H})$ is as large as possible.

Applications of MSTSP and its variations include medical image processing [666], obtaining retvetting sequence in joining metals in the aircraft industry [745, 746] etc.

The problems MSTSP and BTSP are equivalent in the sense that an optimal solution to the MSTSP with weight matrix W can be obtained by solving the BTSP with cost matrix $C = -W$ and vice versa. If we require the edge costs/weights to be positive, a large enough constant could be added to each of the edge costs/weights. However, this trans-

formation may not be useful for some approximation algorithms that use special structure of C or W , since the addition of a constant and/or multiplication of edge costs/weights by -1 may violate key properties of the matrix that are used by these algorithms. Thus in such cases, the characteristics of the two problems are different and warrant separate treatment.

When the cost matrix C is symmetric, (equivalently the underlying graph is symmetric), we refer to the BTSP as *symmetric bottleneck traveling salesman problem* (SBTSP); otherwise it is referred to as *asymmetric bottleneck traveling salesman problem* (ABTSP). A special case of SBTSP, where the vertices of G correspond to points in the Euclidean plane and edge costs are Euclidean distances, is referred to as *Euclidean bottleneck traveling salesman problem* (EBTSP). Similarly we have symmetric, asymmetric, and Euclidean versions of the MSTSP.

It is well known that the Hamiltonian cycle problem on a grid graph is NP-complete [466]. As an immediate consequence we have that EBTSP is NP-hard and hence BTSP is NP-hard [466]. In fact BTSP is NP-hard even if we restrict the graph G to a grid graph or a planar bipartite graph in which degree of each node is 3 or less. As in the case of TSP, this follows immediately by a reduction from the Hamiltonian cycle problem on these graphs which is known to be NP-complete [466] (see Appendix B). Similarly, MSTSP can be shown to be NP-hard on grid graphs and planar bipartite graphs in which degree of each node is 3 or less. Fekete [284] recently proved that MSTSP under Euclidean distances in \mathbb{R}^d is NP-hard for any fixed $d \geq 3$. More complexity results are discussed in Section 3.

2. Exact Algorithms

Recall that an exact algorithm for an optimization problem is guaranteed to produce an optimal solution for any instance of the problem or declare that a feasible solution does not exist. Since BTSP and MSTSP are NP-hard, such algorithms are generally of implicit enumeration type. Exact algorithms for MSTSP have not been discussed explicitly in literature. However, the transformation discussed in Section 1 can be used to solve MSTSP as BTSP.

2.1. BTSP as a TSP

In the preceding chapters, we have seen several interesting properties of and solution approaches for the TSP. Let us now examine how the TSP is related to the BTSP. We will first show that BTSP can be formulated

as a TSP in the sense that an optimal solution to this TSP gives an optimal solution to the BTSP.

Note that in solving BTSP to optimality, the numerical values of the edge costs are unimportant; only the ordering of edge costs matters. Let the edges of G be labeled as $\{e_1, e_2, \dots, e_m\}$ such that $c_{e_1} \leq c_{e_2} \leq \dots \leq c_{e_m}$. Let $\{d_e : e \in E\}$ be another set of costs for the edges of G . Let us denote by $\text{BTSP}(C)$ and $\text{BTSP}(D)$ the instances of BTSP with edge costs c_e 's and d_e 's respectively. Let L and U respectively be known lower and upper bounds on the optimal objective function value of $\text{BTSP}(C)$.

Lemma 1 *If $d_{e_1} \leq d_{e_2} \leq \dots \leq d_{e_m}$ with $d_{e_i} < d_{e_{i+1}}$ whenever $c_{e_i} < c_{e_{i+1}}$ for all i such that $L \leq c_{e_i} \leq U$, then every optimal solution to $\text{BTSP}(D)$ is also an optimal solution to $\text{BTSP}(C)$.*

The proof of the above lemma is straightforward.

Let $\alpha_1 < \alpha_2 < \dots < \alpha_t$ be an ascending arrangement of distinct costs c_e of edges of G such that $L \leq c_e \leq U$. Define $F_r = \{\mathcal{H} \in \mathbb{F} : c_{\max}(\mathcal{H}) = \max\{c_e : e \in \mathcal{H}\} = \alpha_r\}$ for $r = 1, \dots, t$, $F_{t+1} = \{\mathcal{H} \in \mathbb{F} : c_{\max}(\mathcal{H}) > \alpha_t\}$, and $U_r = \cup_{i=1}^r F_i$, $r = 1, \dots, t+1$. Consider new edge weights $\{d_e : e \in E\}$ satisfying,

$$\min\left\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in F_r\right\} > \min\left\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in U_{r-1}\right\} \tag{7}$$

for all $2 \leq r \leq (t + 1)$. Here, minimum over empty set is taken as $-\infty$. Let $\text{TSP}(D)$ denote the TSP with edge weights d_e for $e \in E$.

Theorem 2 *Every optimal solution to $\text{TSP}(D)$ is also an optimal solution to $\text{BTSP}(C)$.*

Proof. Clearly, if k is the smallest index such that F_k is non-empty then any $\mathcal{H} \in F_k$ is an optimal solution to $\text{BTSP}(C)$ with the optimal objective function value α_k . If \mathcal{H}' is an optimal solution to $\text{TSP}(D)$ and $\mathcal{H}' \in F_p$, then,

$$\sum_{e \in \mathcal{H}'} d_e = \min\left\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in F_p\right\} > \min\left\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in U_{p-1}\right\}.$$

Thus, U_{p-1} must be empty and hence \mathcal{H}' is optimal to $\text{BTSP}(C)$. ■

Corollary 3 *Let $b_1 = 0$ and for $j = 2, \dots, t + 1$, let $b_j = nb_{j-1} + 1$. Let the edge costs d_e 's be defined as :*

$$d_e = \begin{cases} 0 & \text{if } c_e \leq \alpha_1 \\ b_j & \text{if } c_e = \alpha_j, \quad j = 2, \dots, t \\ b_{t+1} & \text{if } c_e > \alpha_t \end{cases}$$

Then every optimal solution to $TSP(D)$ is also an optimal solution to $BTSP(C)$.

Corollary 4 Let p be the largest index such that $c_{e_p} \leq U$ and $m = |E|$. Define the edge costs $d_{e_j} = 2^{j-1}$ for $j = 1, 2, \dots, p$ and $d_{e_j} = 2^{p+1}$ for $j = p + 1, p + 2, \dots, m$. Then every optimal solution $TSP(D)$ is also an optimal solution to $BTSP(C)$.

Let k_i be the number of edges e with $c_e = \alpha_i$, $i = 1, \dots, t$.

Corollary 5 Let $b_1 = 0$. For $j = 2, \dots, t + 1$, let u_j be the smallest positive integer such that $\sum_{i=u_j}^{j-1} k_i < n$. Define $b_j = \sum_{i=u_j}^{j-1} k_i b_i + (n - \sum_{i=u_j}^{j-1} k_i) b_{u_j-1} + 1$. Let the edge costs d_e 's be defined as :

$$d_e = \begin{cases} 0 & \text{if } c_e \leq \alpha_1 \\ b_j & \text{if } c_e = \alpha_j, j = 2, \dots, t \\ b_{t+1} & \text{if } c_e > \alpha_t \end{cases}$$

Then every optimal solution to $TSP(D)$ is also an optimal solution to $BTSP(C)$.

The proofs of corollaries 3, 4 and 5 follow from the fact that the special edge costs defined satisfy the condition (7).

Results analogous to Theorem 2 in the context of various bottleneck problems including the BTSP are well known [146, 410, 472, 691]. Although, Theorem 2 shows that BTSP can be solved as a TSP, it is not of much practical value since the costs d_e used in $TSP(D)$ grow exponentially. However, when the number of distinct edge costs between given lower and upper bounds on the optimal objective function value is relatively small (say ≤ 10), then the edge costs defined in corollaries 3 and 5 are sometimes useful. This is exploited in the generalized threshold algorithm given in the next section.

A result similar to Theorem 2 can be obtained for the case of MSTSP showing that MSTSP can be solved as a MAX TSP or as a TSP.

2.2. Generalized Threshold Algorithm

The generalized threshold algorithm [691] is a modification of the well known threshold algorithm for solving bottleneck problems [269]. It solves BTSP as a sequence of TSP's with relatively smaller edge costs utilizing Corollary 3. Without loss of generality, we assume the edge costs are positive. Thus $L > 0$. Let S_1, S_2, \dots, S_r be an ordered partition of the index set $\{1, 2, \dots, t\}$, (that is, $p \in S_i, q \in S_j, i < j$ implies $\alpha_p <$

α_q). We say that an edge e of G corresponds to S_i if $c_e = \alpha_p$ for some $p \in S_i$. For each edge e that corresponds to S_i , define $c'_e = i$, $1 \leq i \leq r$. Define $c'_e = 0$ if $c_e < L$ and $c'_e = r + 1$ if $c_e > U$. Let $\text{BTSP}(C')$ represent the BTSP with edge costs $\{c'_e : e \in E\}$. Let \mathcal{H}' be an optimal solution to $\text{BTSP}(C')$ with optimal objective function value k . (Note that $k \in \{1, \dots, r\}$.) Let \mathcal{H}^* be an optimal solution to BTSP.

Theorem 6 [691] $\min_{i \in S_k} \{\alpha_i\} \leq c_{\max}(H^*) \leq \max_{i \in S_k} \{\alpha_i\}$.

Note that $\text{BTSP}(C')$ is an approximation to the problem $\text{BTSP}(C)$. If r is small (say ≤ 10), then $\text{BTSP}(C')$ could be solved as a TSP with edge costs of moderate size using Corollary 3 or 5. Further, the solution to $\text{BTSP}(C')$ provides new lower and/or upper bounds for the BTSP as guaranteed by Theorem 6. Using these new bounds, a ‘better’ approximation to BTSP can be constructed. Continuing this process, we eventually get an optimal solution to the BTSP. Our generalized threshold algorithm is precisely this. A formal description of the algorithm is given below.

The Generalized Threshold Algorithm

- Step 1:** Construct a lower bound L and an upper bound U for the optimal objective function value. Set $q = 1$, (q is the iteration counter).
- Step 2:** Let $\alpha_1 < \alpha_2 < \dots < \alpha_{t_q}$ be an ascending arrangement of distinct edge costs c_e such that $L \leq c_e \leq U$.
- Step 3:** Construct the ordered partition $S_1, S_2, \dots, S_{r(q)}$ of $\{1, 2, \dots, t_q\}$. Let every edge e with cost $c_e < L$ correspond to S_0 and every edge e with $c_e > U$ correspond to $S_{r(q)+1}$. Let $c'_e = i$ for edges e that corresponds to S_i , $0 \leq i \leq r(q) + 1$.
- Step 4:** Solve the BTSP with costs c'_e . (This is done by solving an equivalent TSP indicated in Theorem 2.) Let \mathcal{H}' be the optimal solution produced and let the optimal objective function value be k . (Note that $k \in \{1, \dots, r(q)\}$.)
- Step 5:** If $|S_k| = 1$, then output \mathcal{H}' and stop. Else update the lower and upper bounds
 $L = \min_{i \in S_k} \{\alpha_i\}$ and
 $U = \max_{e \in \mathcal{H}'} \{c_e\}$.
 Set $q = q + 1$ and go to Step 2.

The validity of the generalized threshold algorithm follows from the preceding discussions. Note that the number of partitions, r (indicated as $r(q)$ in the algorithm), is a function of the iteration counter and can

be changed from iteration to iteration. For a large value of r , the TSP in Step 4 will have very large edge costs, leading to overflow errors. A reasonable choice of r is a number less than or equal to 10. The complexity of the algorithm depends on the number of iterations, which in turn depends on the way the partitions S_1, S_2, \dots, S_r are constructed. If S_1, S_2, \dots, S_{r-1} are selected as singletons and S_r contains all the remaining edges with cost c_e between L and U , the algorithm could take $O(n^2/r)$ iterations in worst case, since $t = O(n^2)$. We call this the *incremental search version* of the generalized threshold algorithm. If $|S_i|$ is approximately equal to $|S_j|$ for all $1 \leq i, j \leq r$, the number of iterations of the generalized threshold algorithm is $O(\log_r n)$. We call this the *r-section version* of the generalized threshold algorithm. If the lower bound L is expected to be tight, the incremental search version may work well in practice and often terminates in one iteration, although its worst case complexity is too high. If $r = 2$ (which is desirable for very large n), Step 4 of the generalized threshold algorithm can be implemented as testing hamiltonicity of an appropriate spanning subgraph of G . This special case of the generalized threshold algorithm is precisely the adaptation of the well known *threshold algorithm* to the BTSP.

The generalized threshold algorithm could take advantage of existing powerful TSP codes to get a reasonably fast algorithm for the BTSP without much programming effort. It is easy to develop a corresponding generalized threshold algorithm for the MSTSP which solves the problem as a sequence of MAX TSP's. We could also use the generalized threshold algorithm for BTSP to solve MSTSP using the transformation discussed in the Section 1.

2.3. Branch and Bound Algorithms

Branch and bound algorithms are classical approaches for solving 'hard' combinatorial optimization problems. The power of a branch and bound algorithm depends on the ability to generate good lower and upper bounds on the optimal objective function value and establishing an efficient branching strategy to generate the search tree. Garfinkel and Gilbert [349], Carpaneto et al [165], and Sergeev and Chernyshenko [757] developed specialized branch and bound algorithms to solve BTSP. Computational results based on problems of size less than or equal 200 are reported in [165, 349]. There is no recent experimental study published on the branch and bound algorithms for BTSP. The branching strategies used by Carpaneto et al [165] and Garfinkel and Gilbert [349] are similar to those studied for the case of TSP and will not be discussed here. For details we refer to the original papers. (See also Chapter 4.)

2.3.1 Lower Bounds. We now discuss some good lower bounds for the BTSP that can be obtained efficiently.

2-max Bound: This lower bound is very simple and easy to compute. It is valid for the symmetric version of BTSP only. For each node i of G , let $\Delta(i)$ be the set of edges incident on i and μ_i be the second smallest cost (counting multiplicity) of edges in $\Delta(i)$. Then $\max_{i \in N} \{\mu_i\}$ is a lower bound on the optimal objective function value of the BTSP. In a graph with m edges, this bound can be identified in $O(m)$ time. An asymmetric version of 2-max bound is introduced by Carpaneto et al [165].

Biconnected Spanning Subgraph Bound: We denote this lower bound as *BSS bound* and it is defined for the symmetric version of BTSP. Since every tour is biconnected, the optimum objective function value of a Bottleneck biconnected spanning subgraph problem (BBSSP) on the graph G is a lower bound for the BTSP. Several algorithms are available to solve the BBSSP. For example, in a graph with m edges and n nodes, Manku [578] proposed an $O(m)$ algorithm, Punnen and Nair [685] an $O(m + n \log n)$ algorithm, Timofeev [793] an $O(n^2)$ algorithm, and Parker and Rardin [661] an $O(n^2 \log n)$ algorithm to solve BBSSP.

Strongly Connected Spanning Subgraph Bound: We denote this bound as *SCSS bound* and is used for the asymmetric version of the BTSP. Since every directed Hamiltonian cycle is strongly connected, the optimal objective function value of a bottleneck strongly connected spanning subgraph problem (BSSSP) on the digraph G is a lower bound for the ABTSP on G . In a digraph with m arcs, BSSSP can be solved in $O(m)$ time [677].

Assignment Bound: The assignment problem is used to compute lower bounds for the traveling salesman problem. In the same way, the bottleneck assignment problem (BAP) can be used to compute a lower bound for the BTSP. Carpaneto et al [165] used the assignment bound, among other lower bounds, in their branch and bound algorithm for BTSP. They also provided a heuristic search scheme to find alternate optimal solutions for the BAP that correspond to cyclic permutations (tours). If the heuristic is successful in getting such a tour, it is indeed an optimal tour. BAP can be solved in $O(n^{2.5})$ time using an algorithm of Punnen and Nair [686]. For the case of sparse cost matrix (several edge costs are very large) an algorithm due to Gabow and Tarjan [343] runs faster. Other algorithms for BAP include Derigs and Zimmerman [253],

and Carpaneto and Toth [167].

2.4. Branch and Cut Algorithms

Branch and cut algorithms are the state-of-the-art exact algorithms for the TSP (see Chapters 4 and 2) that could solve reasonably large TSPs to optimality. A branch and cut algorithm for the TSP is based on a partial linear programming representation of the TSP. The success of the algorithm depends on the ability to generate facets or high dimensional faces of the TSP polytope that are violated by a ‘current’ infeasible solution. The BTSP can be formulated as a bottleneck linear programming problem (BLP) [410]

$$\begin{array}{ll} \text{Minimize} & \max\{c_e : x_e > 0\} \\ \text{Subject to} & X \in T_n, \end{array}$$

where T_n is the TSP polytope (symmetric or asymmetric depending on whether BTSP is symmetric or asymmetric) and entries of $X = (x_1, x_2, \dots, x_m)$ correspond to edges of G . Since the objective function of the BLP is concave and quasi-convex, it can be shown that there exists an optimal solution to this BLP that is an extreme point of T_n and a local minimum is a global minimum. Thus branch and cut algorithms similar to those for TSP can be developed for the BTSP using cutting planes and a BLP solver. However, no implementation of such an algorithm is available in literature.

A dynamic programming approach for the BTSP was proposed by Sergeev [756]. As in the case of TSP, this algorithm is primarily of theoretical importance only.

3. Approximation Algorithms

In Chapters 5 and 6 we have studied approximation algorithms for the TSP where a *priori* mathematical guarantee could be obtained on the performance of an approximation algorithm. Chapters 8, 9, and 10 discussed implementation aspects of various practical approximation algorithms (heuristics) for TSP. The literature on approximation algorithms for BTSP is not as extensive as that of the TSP. In this section we study approximation algorithms for BTSP and MSTSP. We assume throughout this section that the edge costs are positive.

Recall that an algorithm yields a factor δ approximation for a minimization problem, if the algorithm is guaranteed to produce a solution whose objective function value is at most δ times the optimal objective function value. An algorithm yields a factor δ -approximation for a max-

imization problem if the algorithm is guaranteed to produce a solution whose objective function value is at least $1/\delta$ times the optimal objective function value.

Note that transformation used earlier between BTSP and MSTSP may not map a δ -approximate solution of BTSP to a δ -approximate solution of MSTSP. Thus, for worst case analysis of approximation algorithms we treat these two problems separately.

Theorem 7 [661, 33] *Unless $P = NP$, there is no polynomial time δ -approximation algorithm for the BTSP or MSTSP for any constant δ , $1 \leq \delta < \infty$.*

Proof. Let us first consider the case of symmetric BTSP. We prove the theorem by showing that any such approximation algorithm \mathcal{A} , if exists, can be used to solve the Hamiltonian cycle problem in polynomial time, implying $P = NP$. Suppose that we are given a graph G^* and we wish to test if G^* contains a Hamiltonian cycle. Convert the graph G^* to a complete graph K_n by adding the missing edges and assign a cost of $\delta + 1$ to each of these new edges. Assign a cost 1 to each of the original edges. Now we have an instance of SBTSP on K_n . If G^* contains a Hamiltonian cycle, the optimal objective function value, OPT , of our SBTSP is 1 and if the algorithm \mathcal{A} is applied to this instance, it must produce a tour of value δ or less (in fact exactly equal to 1). If G^* has no Hamiltonian cycle, then OPT is $\delta + 1$. Thus the objective function value of the solution produced by \mathcal{A} is less than or equal to δ precisely when G^* contains a Hamiltonian cycle. The result now follows from the NP-completeness of Hamiltonicity testing [347]. The proof for the case of ABTSP or MSTSP (symmetric and asymmetric) can be obtained in a similar way. ■

In view of Theorem 7, it is not very likely that we shall succeed in obtaining meaningful performance bound for polynomial time approximation algorithms for BTSP or MSTSP with arbitrary edge costs. However, by assuming special properties of edge costs, heuristics with guaranteed performance bounds can be obtained.

3.1. Worst Case Analysis of Heuristics for BTSP

Recall that for any $\tau \geq 1/2$, the edge costs c_e of a complete graph K_n satisfy the τ -triangle inequality [22], if for any three nodes i, j, k of K_n , $c_{ij} \leq \tau(c_{ik} + c_{kj})$. If $\tau = 1$, τ -triangle inequality reduces to the triangle inequality. $\tau = 1/2$ forces all the edges of K_n to be of same cost. If $\tau > 1$, τ -triangle inequality can be viewed as a relaxation of the triangle inequality, where as for $1/2 < \tau < 1$, it is a restriction on the

triangle inequality. The following lemma is an immediate consequence of the τ -triangle inequality.

Lemma 8 *Suppose the edge costs c_e of K_n satisfy the τ -triangle inequality for some $\tau \geq 1/2$. Let $P(i, j) = (e_1, e_2, \dots, e_r)$ be a path in K_n joining vertices i and j . Then, $c_{ij} \leq \min\{S_1, S_2\}$, where $S_1 = \tau^{r-1}c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k}$ and $S_2 = \tau^{r-1}c_{e_1} + \sum_{k=1}^{r-1} \tau^k c_{e_{r+1-k}}$.*

The concept called the *power of a graph* plays a central role in our approximation algorithms for SBTSP

Definition 9 *Let $G = (N, E)$ be a graph (not necessarily complete) and t be a positive integer. The t^{th} power of G is the graph $G^t = (N, E^t)$, where there is an edge $(u, v) \in E^t$ whenever there is a path from u to v in G with at most t edges.*

For any graph $G = (N, E)$ with edge costs c_e for $e \in E$, $c_{\max}(G)$ denotes $\max\{c_e : e \in E\}$. Similarly, $c_{\min}(G) = \min\{c_e : e \in E\}$. A similar notation will be used if G is replaced by a collection S of edges of G . We use the phrase $e \in G$ and $e \in E$ interchangeably.

G^2 is called the square of the graph G and G^3 is the cube of G .

Lemma 10 *Suppose the edge costs c_e of K_n satisfy the τ -triangle inequality for some $\tau \geq 1/2$. Let G be a subgraph of K_n . Then*

$$c_{\max}(G^t) \leq \begin{cases} t c_{\max}(G) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1)c_{\max}(G) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^{t-1} + \tau - 2)c_{\max}(G) & \text{if } \tau < 1 \end{cases}$$

Proof. Let (i, j) be an arbitrary edge of G^t . By definition of G^t , there exists a path $P(i, j) = (e_1, e_2, \dots, e_r)$ in G^t from i to j of length at most t . By Lemma 8,

$$c_{ij} \leq \tau^{r-1}c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k} \leq t c_{\max}(G) \text{ for } \tau = 1.$$

Similarly, if $\tau > 1$, then by Lemma 8 we have,

$$\begin{aligned}
 c_{ij} &\leq \tau^{r-1}c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k} \\
 &\leq (\tau^{r-1} + \sum_{k=1}^{r-1} \tau^k)c_{\max}(G) \\
 &\leq (\tau^{t-1} + \sum_{k=1}^{t-1} \tau^k)c_{\max}(G) \\
 &= \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1)c_{\max}(G)
 \end{aligned}$$

The case $\tau < 1$ can be proved in a similar way. ■

Theorem 11 Suppose the edge costs c_e of K_n satisfy the τ -triangle inequality for some $\tau \geq 1/2$. Let S be a spanning subgraph of K_n such that $c_{\max}(S)$ is a lower bound for the optimal objective function value of BTSP and let \mathcal{H}^* be an optimal solution to BTSP on K_n . If S^t , $1 \leq t < n$ and integer t , contains a Hamiltonian cycle \mathcal{H} , then

$$c_{\max}(\mathcal{H}) \leq \begin{cases} t c_{\max}(\mathcal{H}^*) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1)c_{\max}(\mathcal{H}^*) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^{t-1} + \tau - 2)c_{\max}(\mathcal{H}^*) & \text{if } \tau < 1 \end{cases} .$$

Proof. Assume that $\tau = 1$. By definition of power of a graph, $c_{\max}(\mathcal{H}) \leq \max(S^t) \leq t c_{\max}(S)$. The last inequality follows from Lemma 10. From the optimality of \mathcal{H}^* , $c_{\max}(S) \leq c_{\max}(\mathcal{H}^*)$. The result now follows immediately. The case $1/2 \leq \tau < 1$ and $\tau > 1$ can be proved in a similar way. ■

Theorem 11 and Lemma 10 are generalizations of corresponding results by Hochbaum and Shmoys [449] proved for the case $\tau = 1$.

Let us now discuss a simple heuristic for BTSP called *the bottleneck double tree heuristic*, which is an adaptation of the double tree heuristic for the TSP.

Bottleneck Double Tree Heuristic

Step 1: Compute a bottleneck spanning tree T of K_n .

Step 2: Duplicate the edges of T to form an Eulerian multigraph T^* .

Step 3: Identify an Eulerian tour in T^* . Traverse T^* along this Eulerian tour and introduce shortcuts whenever a previously visited node is encountered, to produce a tour in K_n .

Note that Step 3 of the algorithm allows a lot of flexibility. It is possible to construct examples where the double tree heuristic produces the worst solution. The quality of the solution produced however depends on the order in which the edges are traversed in the Eulerian tour. The solution generated by different orders of traversal may be different. Among all such solutions, identifying the best one can be shown to be a NP-hard problem. It may be noted that the cube of any connected graph is Hamiltonian connected (and hence Hamiltonian) [448, 449]. It has been shown by Hobbs [448] that a tour can be generated in Step 3 of the algorithm by introducing shortcuts to only paths of T^* of length 3 or less and such a tour, say \mathcal{H}' , belongs to T^3 . (See also [100].) By Theorem 11, if the edge costs satisfy τ -triangle inequality, then

$$c_{\max}(\mathcal{H}') \leq \begin{cases} 3 c_{\max}(\mathcal{H}^*) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^2 - \tau - 1)c_{\max}(\mathcal{H}^*) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^2 + \tau - 2)c_{\max}(\mathcal{H}^*) & \text{if } 1/2 \leq \tau < 1 \end{cases}$$

where \mathcal{H}^* is an optimal solution to SBTSP. The short-cutting phase as described above can be done in $O(n)$ time, (see for example Hobbs [448]). Also, the bottleneck spanning tree in Step 1 can be obtained in $O(m)$ time, where m is the number of edges in G [153]. Thus we have the following theorem.

Theorem 12 *If the edge costs satisfy the τ -triangle inequality, then the double tree algorithm produces a solution to the SBTSP in $O(n^2)$ time with a performance bound δ , where*

$$\delta = \begin{cases} 3 & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^2 - \tau - 1) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^2 + \tau - 2) & \text{if } 1/2 \leq \tau < 1 \end{cases}$$

Let us now consider a general heuristic algorithm for BTSP based on the concept of power of a graph.

Algorithm Power(S,t)

- Step 1:** Construct a connected spanning subgraph S of K_n such that $c_{\max}(S)$ is a lower bound on the optimal objective function value of the BTSP.
- Step 2:** Find the smallest positive integer t such that S^t is Hamiltonian.
- Step 3:** Output any Hamiltonian cycle in S^t .

When S is a bottleneck spanning tree, and $t = 3$, Power(S,t) is identical to the double tree algorithm if Step 3 of power(S,t) is implemented

using Steps 2 and 3 of the double tree algorithm. The complexity and performance bound in this case are the same as those of the double tree algorithm. We now show that by investing the same amount of time a better performance bound can be achieved.

Recall that the objective function value of the bottleneck biconnected spanning subgraph problem (BBSSP) is a lower bound on the optimal objective function value of BTSP. Thus, we could use a bottleneck biconnected spanning subgraph for S in Step 1 of algorithm $Power(S, t)$. As we have seen earlier, BBSSP on K_n can be solved in $O(n^2)$ time, (see Timofeev [793], Parker and Rardin [661], Punnen and Nair [685], and Manku [578]).

Now, for a biconnected graph, S , what is the smallest value of t such that S^t is Hamiltonian? Fleischner [312] proved that this value of t is 2. We state this elegant result in the following theorem.

Theorem 13 *The square of every biconnected graph is Hamiltonian.*

We are now left with the task of generating a tour in the square of a biconnected graph. Lau [542, 543] suggested an $O(n^2)$ algorithm to accomplish this. (See also Rardin and Parker [697].) Thus, by Theorem 11, the performance bound δ of this algorithm is 2τ (using $t = 2$) for $\tau \geq 1/2$. We thus have the following theorem.

Theorem 14 *If the edge costs of K_n satisfy τ -triangle inequality, then a 2τ -approximate solution to the BTSP can be obtained in $O(n^2)$ time.*

Algorithm $Power(S, t)$ for $t = 2$, and 3, with $\tau = 1$ was published first by Doroshko and Sarvanov [260] in a Russian paper in 1981. The case $t = 2$, $\tau = 1$ was obtained independently by Parker and Rardin [661]. Hochbaum and Shmoys [449] considered the case for general t with $\tau = 1$. We now show that improving the performance bound of Theorem 14 for any polynomial time algorithm and any $\tau > 1/2$ amounts to P=NP. This result was proved by Doroshko and Sarvanov [260], Parker and Rardin [661], and Hochbaum and Shmoys [449] for the case $\tau = 1$.

Theorem 15 *Unless $P = NP$, there is no polynomial time $2\tau - \epsilon$ approximation algorithm for BTSP on K_n , with edge costs satisfying τ -triangle inequality, for any $\epsilon > 0$, $\tau > 1/2$.*

Proof. We prove the theorem by showing that for any $\epsilon > 0$, $\tau > 1/2$, a polynomial time $2\tau - \epsilon$ approximation algorithm \mathcal{A} for the BTSP on K_n , with edge costs satisfying τ -triangle inequality, can be used to test Hamiltonicity of an arbitrary graph (digraph), establishing P=NP. Let

G be an arbitrary graph (digraph) with each edge of cost one. Convert G into a complete graph (digraph) by adding new edges of cost 2τ each. Clearly the costs of edges of this complete graph (digraph) satisfy τ -triangle inequality. It can be verified that \mathcal{A} produces a tour \mathcal{H} with worst case error bound $2\tau - \epsilon$ precisely when \mathcal{H} is a tour in G . This completes the proof. ■

Bollobás, Fenner, and Frieze [128] discussed an approximation algorithm for BTSP along with its probabilistic analysis. This algorithm uses a variation of the algorithm HAM discussed in Chapter 7.

3.2. Approximation Algorithms for MSTSP

Let us now consider an approximation algorithm for the MSTSP. Although algorithms discussed in the previous subsection could be easily modified to generate corresponding algorithms for MSTSP, the performance guarantees do not translate in the same way.

The basic idea of this approximation algorithm is to solve a ‘relaxation’ of the MSTSP, which provides a lower bound on the optimal objective function value, and generate a Hamiltonian cycle from the optimal solution of this relaxation. Sufficient conditions for existence of a Hamiltonian cycle in a graph form the key to our algorithm. Some well known such conditions are summarized in the following theorem.

Theorem 16 *If a graph $G = (N, E)$ on n nodes satisfies any of the following properties $\mathcal{P}1, \mathcal{P}2, \dots, \mathcal{P}6$, then G is Hamiltonian.*

$\mathcal{P}1$: $\deg(v) \geq \lceil n/2 \rceil$ for all $v \in N$ (Dirac [256]).

$\mathcal{P}2$: $\deg(u) + \deg(v) \geq n$ for all pairs of non-adjacent vertices u and v of G . (Ore [636]).

$\mathcal{P}3$: $i < n/2$ implies $\deg(i) > i$ or $\deg(n - 1) > n - i$, where the nodes $1, 2, \dots, n$ of G are labeled in the ascending sequence of their degrees. (Chvátal [194]).

$\mathcal{P}4$: $\kappa(G) \geq \alpha(G)$, where $\kappa(G)$ is the connectivity number and $\alpha(G)$ is the independence number of G . (Chvátal-Erdos [199]).

$\mathcal{P}5$: $\theta(G)n \geq \alpha(G)$ where $\theta(G) = \min\{t(S) : S \subset N, S \neq \emptyset\}$ with $t(S) = \frac{|\bar{S} \cap N(S)|}{|S|}$ and $N(S)$ is the neighborhood of S (Lu [571]).

$\mathcal{P}6$: G has minimum degree k , $n \geq 6k$ and $|E| > \binom{n-k}{2} + k^2$ (Erdos [276]).

Let $\mathcal{F}(\mathcal{P})$ be the family of all spanning subgraphs of K_n satisfying a prescribed property \mathcal{P} which guarantees that elements of $\mathcal{F}(\mathcal{P})$ are Hamiltonian. Consider the *maxmin* problem (MMP(\mathcal{P}))

$$\begin{aligned} &\text{Maximize} && w_{\min}(S) \\ &\text{Subject to} && S \in \mathcal{F}(\mathcal{P}) \end{aligned}$$

Let $S(\mathcal{P})$ be an optimal solution to MMP(\mathcal{P}) with optimal objective function value $z(\mathcal{P})$ and \mathcal{H}^* be an optimal solution to the MSTSP. Clearly $z(\mathcal{P}) \leq w_{\min}(\mathcal{H}^*)$. Consider the following approximation algorithm for MSTSP.

The \mathcal{P} -relaxation Algorithm

Step 1: Solve MMP(\mathcal{P}). Let $S(\mathcal{P})$ be an optimal solution generated.

Step 2: Output any Hamiltonian cycle $\mathcal{H}(\mathcal{P})$ in $S(\mathcal{P})$.

The complexity of this algorithm depends on the ability to solve MMP(\mathcal{P}) and to generate $\mathcal{H}(\mathcal{P})$ from $S(\mathcal{P})$, which in turn depends on the property \mathcal{P} . We will study the case when $\mathcal{P} = \mathcal{P}1$ or $\mathcal{P}2$. The \mathcal{P} -relaxation algorithm is based on the original ideas of Arkin et al [33] where the case $\mathcal{P} = \mathcal{P}1$ was considered.

Let us first consider the case $\mathcal{P} = \mathcal{P}2$. In this case, MMP(\mathcal{P}) can be solved using the binary search version of the threshold algorithm [269] for bottleneck problems. The complexity of the algorithm, in this case, would be $O(n^2 \log n)$. Since $\mathcal{P}1$ implies $\mathcal{P}2$, the case $\mathcal{P} = \mathcal{P}1$ can be solved by the algorithm for $\mathcal{P} = \mathcal{P}2$. However, this special case can be solved more efficiently in $O(n^2)$ time as observed by Arkin et al [33]. For each node i of K_n , compute the $\lceil n/2 \rceil^{th}$ largest weight of edges that are adjacent i (counting multiplicity). This can be done in $O(n)$ time [5]. Let δ be the smallest of all these values. Remove all edges with weight less than δ from K_n . The resulting graph is an optimal solution to MMP($\mathcal{P}1$).

A Hamiltonian cycle $\mathcal{H}(\mathcal{P}1)$ in $S(\mathcal{P}1)$ can be generated in $O(n^2)$ time. (see Arkin et al[33].) This algorithm is based on a constructive proof of sufficiency of $\mathcal{P}1$ for the existence of a Hamiltonian cycle in a graph. The same algorithm works for generating a Hamiltonian cycle in $S(\mathcal{P}2)$.

The following lemma is useful in establishing a performance bound for the \mathcal{P} -relaxation algorithm.

Lemma 17 [33] *Let R be a subset of nodes of K_n such that $|R| > \lfloor n/2 \rfloor$. Then for any Hamiltonian cycle of K_n , there exists an edge joining two nodes of R .*

Let $e^* = (u, v)$ be an edge in $S(\mathcal{P}2)$ of smallest weight. Without loss of generality we assume $S(\mathcal{P}2)$ contains all edges of weight more than w_{e^*} and it is ‘minimal’ in the sense that removal of any more edges of weight w_{e^*} from $S(\mathcal{P}2)$ violates the condition of $\mathcal{P}2$. Hence, there exists a node i in $S(\mathcal{P}2)$ such that degree of i is less than or equal to $\lfloor n/2 \rfloor$. Consider the set

$$R = \{i\} \cup \{j : (i, j) \notin S(\mathcal{P}2)\}.$$

Since $\deg(i) \leq \lfloor n/2 \rfloor$, we have $|R| > \lfloor n/2 \rfloor$. Let \mathcal{H}^* be an optimal solution to MSTSP. By Lemma 17 \mathcal{H}^* contains an edge f joining two nodes of R . Thus

$$w_{\min}(\mathcal{H}^*) \leq w_f.$$

Now, suppose the edge weights of K_n satisfy τ -triangle inequality for some $\tau \geq 1/2$. Since for any $j \in R, j \neq i$, weight of the edge (i, j) is no more than w_{e^*} , it follows that $w_f \leq 2w_{e^*}$. Hence, for any Hamiltonian cycle $\mathcal{H}(\mathcal{P}2)$ in $S(\mathcal{P}2)$,

$$w_{\min}(\mathcal{H}^*) = w_f \leq 2\tau w_{e^*} \leq 2\tau w_{\min}(\mathcal{H}(\mathcal{P}2)).$$

The same performance bound holds for the case $\mathcal{P} = \mathcal{P}1$ since $\mathcal{P}1$ implies $\mathcal{P}2$. The forgoing discussion is summarized as follows.

Theorem 18 *The \mathcal{P} -relaxation algorithm produces a solution to MSTSP on K_n with objective function value at least $\frac{1}{2\tau}$ times the optimal objective function value. When $\mathcal{P} = \mathcal{P}1$ (or $\mathcal{P}2$), the complexity of the algorithm is $O(n^2)$ (or $O(n^2 \log n)$).*

It can be shown that the performance bound given above is the best possible for the MSTSP whenever the edge weights satisfy the τ -triangle inequality for $\tau > 1/2$.

3.3. Experimental Analysis of Heuristics

Unlike the TSP, not much literature exists on experimental analysis of heuristics for the BTSP. The algorithms discussed in the previous subsections, without further modifications, may not work well in practice although they are best possible from the worst case analysis point of view. Some quick heuristics for BTSP can be obtained by straightforward modifications of well known construction and/or improvement heuristics for the TSP, such as arbitrary insertion heuristic and its variations, nearest neighbor algorithm, patching algorithms (especially in the case of asymmetric version of BTSP), k -opt heuristic, etc. However, no computational results are reported in literature on any such adaptations. Because of the close similarity of these heuristics with their TSP

counterparts, we shall not elaborate on them. Similar heuristics can be constructed for the case of MSTSP. Garfinkel and Gilbert [349] proposed a heuristic for BTSP which is based on the threshold algorithm.

We now discuss a heuristic based on the generalized threshold algorithm, called the *approximate generalized threshold algorithm* which can be considered as an extension of the heuristic by Garfinkel and Gilbert [349]. This algorithm differs from the generalized threshold algorithm only in the way we solve the TSP in Step 4. In the approximation version, we simply use an approximation algorithm for TSP. Preliminary computational results discussed in [691], using Helsgaun's implementation [446] of the Lin-Kernighan Heuristic as the approximate TSP solver produced optimal solutions for many problems from the TSPLIB. For this implementation, the 2-max lower bound was used as the initial lower bound. For many Euclidean problems, this lower-bound turned out to be the optimal objective function value. The incremental search version produced optimal solutions for almost all these problems in one iteration. Various refinements and hybrids of the approximate generalized threshold algorithm are discussed in [691] along with computational results.

4. Polynomially Solvable Cases of BTSP

In this section we use the permutation definition of BTSP mentioned in Section 1. This will keep the section consistent with analogous results for TSP discussed in Chapter 11. Though the diagonal elements of the cost matrix C do not play any role in the definition of BTSP, various results in this section require specific values of the elements c_{ii} .

The following result will be useful in what follows.

Lemma 19 [361] *Let L be a lower bound on the optimal objective function value of BTSP(C). Define matrix C' as $c'_{ij} = \max\{c_{ij} - L, 0\}$. Then a tour γ is an optimal solution to BTSP(C) if and only if it is an optimal solution to BTSP(C').*

We start with a minor generalization of the classical result of Lawler [546] (see also [361]). We call a cost matrix C a *max-Lawler matrix* if and only if it satisfies the following conditions.

- (i) For all $1 < i \leq j < k \leq n$, $\max\{c_{k1}, c_{ji}\} \geq \max\{c_{ki}, c_{j1}\}$.
- (ii) For all $1 \leq i < j \leq k < n$, $\max\{c_{ni}, c_{kj}\} \leq \max\{c_{nj}, c_{ki}\}$.

It may be noted that every upper triangular matrix C is a max-Lawler matrix.

Theorem 20 *If C is a max-Lawler matrix then the problem BTSP(C) can be solved in $O(n^{2.5})$ time.*

Proof. A modification of the Lawler’s algorithm for TSP with upper triangular cost matrix (see Chapter 11, Section 4.5), with the initial permutation π chosen as an optimal bottleneck assignment solution on C with $\pi(n) = 1$, produces an optimal solution to this subclass of BTSP. The proof is similar to the one in the case of TSP with upper triangular cost matrix. Since the bottleneck assignment problem can be solved in $O(n^{2.5})$ time [686], the result follows. ■

If C is an upper triangular matrix, then the optimal objective function value of BTSP(C) is clearly non-negative. Therefore, using Lemma 19, we can assume that C is a non-negative, upper triangular matrix with diagonal entries as zeros. This allows us to convert the problem of finding an optimal bottleneck assignment solution π with $\pi(n) = 1$, to the problem of finding an optimal bottleneck path from node 1 to node n in \vec{K}_n [361] and the latter can be solved in $O(n^2)$ time.

4.1. Pyramidally Solvable Cases of BTSP

In this section, we use extensively various notations and definitions introduced in Chapter 11. An instance BTSP(C) is said to be *pyramidally solvable* if and only if there exists an optimal tour that is pyramidal. Given any cost matrix C , the problem of finding an optimal pyramidal tour for BTSP(C) can be solved in $O(n^2)$ time by transforming it into the problem of finding an optimal bottleneck path of a certain type in a directed, acyclic multigraph using idea similar to that used in the case of pyramidal TSP in Chapter 11. We give below the details.

Define a directed, acyclic multigraph $\hat{G} = (\hat{N}, E_u \cup E_l)$ where, $\hat{N} = \{1, 2, \dots, n - 1\}$ and for each $1 \leq i < j \leq n - 1$, we have two arcs e_{ij}^u and e_{ij}^l , each from node i to node j , with $e_{ij}^u \in E_u$ and $e_{ij}^l \in E_l$. We call a path in \hat{G} with arcs alternately in E_u and E_l an *alternating path* (*a-path*). We associate with each arc in \hat{G} a set of arcs in \vec{K}_n in such a way that every *a-path* in \hat{G} from node 1 to node $n - 1$ corresponds to a unique pyramidal tour on N and vice versa. We give below an example.

Let $n = 12$ and consider the pyramidal tour $\gamma = (1, 4, 5, 6, 9, 10, 12, 11, 8, 7, 3, 2, 1)$.

The fact that $\gamma(1) = 4$ implies $\gamma(3) = 2$ and $\gamma(2) = 1$. Hence, we associate with arc $e_{1,3}^u$ the set of arcs $\{(1, 4), (3, 2), (2, 1)\}$ in \vec{K}_{12} and we assign to $e_{1,3}^u$ weight $w_{1,3}^u = \max\{c_{1,4}, c_{3,2}, c_{2,1}\}$. Similarly, $\gamma(6) = 9$ implies that $\gamma(8) = 7$. So we associate with $e_{6,8}^u$ the set of arcs $\{(6, 9), (8, 7)\}$ and assign it weight $w_{6,8}^u = \max\{c_{6,9}, c_{8,7}\}$. Using the

same argument, we associate with arcs $e_{10,11}^u, e_{3,6}^l$ and $e_{8,10}^l$ arc sets $\{(10, 12), (12, 11)\}, \{(7, 3), (4, 5), (5, 6)\}$ and $\{(11, 8), (9, 10)\}$, respectively, and assign them weights $w_{10,11}^u = \max\{c_{10,12}, c_{12,11}\}$, $w_{3,6}^l = \max\{c_{7,3}, c_{4,5}, c_{5,6}\}$ and $w_{8,10}^l = \max\{c_{11,8}, c_{9,10}\}$. Then γ corresponds to the a -path $e_{1,3}^u - e_{3,6}^l - e_{6,8}^u - e_{8,10}^l - e_{10,11}^u$ in \hat{G} . It is not difficult to verify that if we assign the following weights to the arcs of \hat{G} , then the bottleneck weight of each a -path equals the bottleneck cost of the corresponding pyramidal tour.

For each $1 \leq i < j \leq n - 1$, assign to e_{ij}^u and e_{ij}^l weights

$$w_{ij}^u = \max\{c_{i,j+1}, \max\{c_{k,k-1} : k \in U_{ij}\}\} \text{ and}$$

$$w_{ij}^l = \max\{c_{j+1,i}, \max\{c_{k,k+1} : k \in L_{ij}\}\},$$

respectively. Here, for $1 < i < j < n - 1$, $U_{ij} = \{i + 2, \dots, j\}$ and $L_{ij} = \{i + 1, \dots, j - 1\}$; for $1 < j < n - 1$, $U_{1j} = \{2, \dots, j\}$ and $L_{1j} = \{1, \dots, j - 1\}$; for $1 < i < n - 1$, $U_{i,n-1} = \{i + 2, \dots, n\}$ and $L_{i,n-1} = \{i + 1, \dots, n - 1\}$ and $U_{1,n-1} = \{2, \dots, n\}$ and $L_{1,n-1} = \{1, \dots, n - 1\}$.

Our problem is thus, equivalent to the problem of finding an optimal bottleneck a -path (an a -path with largest edge weight as small as possible) in \hat{G} from node 1 to node $(n - 1)$. The weights of all the arcs in $E_u \cup E_l$ can be calculated in $O(n^2)$ time and an optimal bottleneck a -path in \hat{G} can be obtained in $O(n^2)$ time by simple modifications of standard shortest path algorithms.

We now present results on polynomially testable sufficiency conditions on C for $BTSP(C)$ to be pyramidally solvable. Lemma 21 will be useful in proving the sufficiency conditions that follow. Here we use the concepts of upper minor $C^{(U,i,j,k)}$ and lower minor $C^{(L,i,j,k)}$ of a matrix C , hereditary matrix properties and a permutation $\pi^{(i)}$ corresponding to any permutation π on N and $i \in N$. These are defined in Chapter 11. Also, we associate with any permutation π on N a digraph $G_\pi = (N, E_\pi)$ where $E_\pi = \{(i, \pi(i)) : i \in N\}$. For any matrix property \mathcal{P} , let $MBVI(\mathcal{P})$ (minimum bottleneck violator index of \mathcal{P}) be the smallest integer n , such that there exists an $n \times n$ matrix C , satisfying property \mathcal{P} , for which $BTSP(C)$ does not have any optimal tour that is pyramidal. Let \mathcal{P} be a hereditary matrix property with $MBVI(\mathcal{P}) < \infty$. Let $n = MBVI(\mathcal{P})$.

Lemma 21 *Suppose an $n \times n$ cost matrix C satisfies property \mathcal{P} and none of the optimal tours to $BTSP(C)$ is pyramidal. Then the following statements are true.*

- (i) If for all $p, q, 3 \leq p < q - 1 \leq n - 1$, each of the upper minors $C^{(U,p+1,q,p)}$ and $C^{(U,q,p+1,p)}$ satisfies property \mathcal{P} , then every optimal tour to $BTSP(C)$ has node $(n - 1)$ as one of its peaks.
- (ii) If for all $p, q, 1 \leq q < p - 1 \leq n - 4$, each of the lower minors $C^{(L,p-1,q,p)}$ and $C^{(L,q,p-1,p)}$ satisfies property \mathcal{P} , then every optimal tour to $BTSP(C)$ has node 2 as one of its valleys.

Proof. We prove the result by contradiction. Thus, if possible, let C be an $n \times n$ matrix satisfying \mathcal{P} such that,

- (i) none of the optimal tours for $BTSP(C)$ is pyramidal;
- (ii) for all $p, q, 3 \leq p < q - 1 \leq n - 1$, each of the upper minors $C^{(U,p+1,q,p)}$ and $C^{(U,q,p+1,p)}$ satisfies \mathcal{P} ; and
- (iii) there exists an optimal tour γ to $BTSP(C)$ with the second largest peak $p < n - 1$.

(The case when the lower minors rather than the upper minors of C satisfy property \mathcal{P} will follow similarly.) Then G_γ contains a pyramidal subpath P_{uv} on node set $\{p + 1, \dots, n\}$, where either $u = p + 1 < v \leq n$ or $v = p + 1 < u \leq n$. Let γ' be the unique, non-trivial subtour of $\gamma^{(p+1)}$ on node set $\{1, 2, \dots, p + 1\}$ and let $C' = C^{(U,v,u,p)}$. By the choice of the matrix C , there exists an optimal tour ψ' for $BTSP(C')$ that is pyramidal. If we replace node $(p + 1)$ in $G_{\psi'}$ by the path P_{uv} in G_γ , the resultant digraph corresponds to a pyramidal tour ψ on N where

$$\psi(i) = \begin{cases} \psi'(i) & \text{for all } i \in \{1, 2, \dots, p\} - \{\psi'^{-1}(p + 1)\} \\ u & \text{for } i = \psi'^{-1}(p + 1) \\ \psi'(p + 1) & \text{for } i = v \\ \gamma(i) & \text{for all other } i \end{cases}$$

Let $a = \max\{c_{ij} : (i, j) \text{ is an arc in } P_{u,v}\}$. Then,

$$c_{\max}(\psi) = \max\{a, c'_{\max}(\psi')\} \text{ and } c_{\max}(\gamma) = \max\{a, c'_{\max}(\gamma')\}.$$

Hence, $c_{\max}(\psi) - c_{\max}(\gamma) \leq 0$. Thus, ψ is an optimal solution to $BTSP(C)$, contradicting the choice of C . ■

As an immediate consequence of the above lemma we have the following.

Let \mathcal{P} be a hereditary matrix property. Let $n = MBVI(\mathcal{P}) < \infty$.

Corollary 22 *Suppose property \mathcal{P} is an upper hereditary (a lower hereditary) matrix property. Let C be an $n \times n$ matrix satisfying property \mathcal{P} for which none of the optimal tours to $BTSP(C)$ is pyramidal. Then*

every optimal tour to $BTSP(C)$ has node $(n - 1)$ as one of its peaks (node 2 as one of its valleys).

The property defined below is the ‘max-version’ of the property BK-II(k) discussed in Chapter 11.

Definition 23 For any integer $k \geq 4$, an $n \times n$ matrix C satisfies property max-BK-II(k) if and only if the following two conditions hold.

1 For any $p, q, k - 1 \leq p < q \leq n$, consider any two sets $\{i_1, i_2, \dots, i_{(k-2)}\}$ and $\{j_1, j_2, \dots, j_{(k-2)}\}$ of $(k - 2)$ distinct integers each such that,

- (i) $1 \leq i_1, i_2, \dots, i_{(k-2)} < p$; and $1 \leq j_1, j_2, \dots, j_{(k-2)} < p$;
- (ii) $i_u \neq j_v$ for all $u \neq v$; and
- (iii) $|\{i_1, i_2, \dots, i_{k-2}\} \cup \{j_1, j_2, \dots, j_{k-2}\}| \geq k - 1$.

Then for the $k \times k$ submatrix C' of C corresponding to the rows and columns $\{i_1, i_2, \dots, i_{(k-2)}, p, p + 1\}$ and $\{j_1, j_2, \dots, j_{(k-2)}, p, q\}$ or $\{i_1, i_2, \dots, i_{(k-2)}, p, q\}$ and $\{j_1, j_2, \dots, j_{(k-2)}, p, p + 1\}$, respectively, there exists an optimal solution γ to $BTSP(C')$ in which node $(k - 1)$ is adjacent to node k , (that is, either $\gamma(k) = k - 1$ or $\gamma(k - 1) = k$).

2 For any $r \leq k$, $BTSP(C^{(U,r)})$ is pyramiddally solvable.

Theorem 24 If C is a max-BK-II(k) matrix for some integer $4 \leq k \leq n$, then $BTSP(C)$ is pyramiddally solvable.

Proof. It is easy to see that the property max-BK-II(k) is upper hereditary. If the result is not true then there exist $k, n, 4 \leq k < n < \infty$, such that $MBVI(max - BK - II(k)) = n$. Let C be an $n \times n$ cost matrix satisfying the property max-BK-II(k) such that there does not exist any optimal solution to $BTSP(C)$ that is pyramidal. Let γ be an optimal solution to $BTSP(C)$. By Lemma 21, $(n - 1)$ is a peak of γ . Choose a set $\{i_1, i_2, \dots, i_{k-2}\}$ of distinct nodes in $\{1, 2, \dots, n - 2\}$ including the nodes $\gamma^{-1}(n - 1)$ and $\gamma^{-1}(n)$. Remove the arcs $\{(i_j, \gamma(i_j)) : 1 \leq j \leq k - 2\} \cup \{(n, \gamma(n)), (n - 1, \gamma(n - 1))\}$ from G_γ to get $k - 2$ node-disjoint, paths $\{P_{u_i, v_i} : 1 \leq i \leq k - 2\}$, in addition to isolated nodes n and $(n - 1)$. If we contract the paths $\{P_{u_i, v_i} : 1 \leq i \leq k - 2\}$ in G_γ to nodes $\{1', 2', \dots, (k - 2)'\}$, respectively, the resultant digraph defines a tour γ' on the node set $\{1', 2', \dots, (k - 2)', n - 1, n\}$. Let C' be the submatrix of C corresponding to the rows $\{v_1, v_2, \dots, v_{(k-1)}, n - 1, n\}$ and columns $\{u_1, u_2, \dots, u_{(k-1)}, n - 1, n\}$. Let us index the rows and columns of C' by

$\{1', 2', (k-2)', n-1, n\}$, in that order. Then, by property max-BK-II(k), there exists an optimal solution ψ' for BTSP(C') such that node n is adjacent to node $(n-1)$ in ψ' . In $G_{\psi'}$, replace the nodes $\{1', 2', \dots, (k-2)'\}$ by the respective paths $\{P_{u_i, v_i} : 1 \leq i \leq k-2\}$ to get the digraph corresponding to a tour ψ on N . Let $a = \max\{c_{rs} : (r, s) \text{ is an arc in } P_{u_i, v_i} \text{ for some } i, 1 \leq i \leq k-2\}$. Then $c_{\max}(\gamma) = \max\{a, c'_{\max}(\gamma')\}$ and $c_{\max}(\psi) = \max\{a, c'_{\max}(\psi')\}$. Hence, $c_{\max}(\gamma) \geq c_{\max}(\psi)$ and ψ is an optimal solution to BTSP(C). But node $(n-1)$ is not a peak node of ψ . We thus have a contradiction. ■

The diagonal elements of the cost matrix C do not form part of any tour. However, solutions produced by certain algorithms for TSP and BTSP such as the Gilmore-Gomory type algorithms, and some sufficiency conditions for validity of these algorithms depend on the values of diagonal elements. The sufficiency condition below falls in this category.

A cost matrix C satisfies property \mathcal{P}_I if and only if

- (i) for any i, j, k such that $1 \leq i, j < k < n$ and $i \neq j$, $\max\{c_{ik}, c_{kj}\} \geq \max\{c_{ij}, c_{kk}\}$;
- (ii) for any i, j such that $1 < i < j \leq n$, $\max\{c_{j, i-1}, c_{ii}\} \geq \max\{c_{ji}, c_{i, i-1}\}$;
- (iii) C^T satisfies conditions (i) and (ii) where C^T is the transpose of C .

Theorem 25 *If matrix C satisfies property \mathcal{P}_I then BTSP(C) is pyramidally solvable.*

Proof. If the result is not true, then there exists some $n < \infty$ such that $MBVI(\mathcal{P}_I) = n$. Let C be an $n \times n$ matrix satisfying the property \mathcal{P}_I such that none of the optimal tours for BTSP(C) is pyramidal. Let γ be an optimal solution to BTSP(C). Let $\gamma^{-1}(n) = a$ and $\gamma(n) = b$. We assume that $a < b$. Since the property \mathcal{P}_I is an upper hereditary matrix property, it follows by Lemma 21 that node $(n-1)$ is a peak node of γ and hence, $b < n-1$. Define a tour γ' on N as follows:

$\gamma'(i) = \gamma^{(b)}(i) \ \forall i \in \{1, 2, \dots, b\} - \{a\}$; $\gamma'(a) = n$; $\gamma'(i) = i-1 \ \forall i \in \{b+1, \dots, n\}$. The digraph $G_{\gamma'}$ can be obtained from G_{γ} by adding and removing arcs in the following sequence. For $i = n-1, \dots, b+1$, in that order, remove arcs (u_i, i) , (i, v_i) and add arcs (u_i, v_i) and (i, i) ; then for $i = b+1, \dots, n-1$, in that order, remove arcs $(n, i-1)$ and (i, i) and add arcs $(i, i-1)$ and (n, i) . By property \mathcal{P}_I , each time the maximum of the costs of the arcs in the modified digraph either decreases or remains the same. Hence, $c_{\max}(\gamma') \leq c_{\max}(\gamma)$ and thus, γ' is an optimal solution to BTSP(C). But $(n-1)$ is not a peak of γ' and this contradicts

Lemma 21. ■

A cost matrix C is called a *max-distribution matrix* if and only if

$$\max\{c_{iv}, c_{ju}\} \geq \max\{c_{iu}, c_{jv}\} \text{ for all } i < j \text{ and } u < v.$$

Every max-distribution matrix satisfies both the properties \mathcal{P}_I and max-BK-II(4). Hence, Corollary 26 below follows from theorems 24 and 25.

Corollary 26 [147] *If C is a max-distribution matrix, then $BTSP(C)$ is pyramidally solvable.*

A cost matrix C is said to be *upward (downward) graded on its rows* if and only if $c_{ij} \leq c_{i,j+1}$ ($c_{ij} \geq c_{i,j+1}$) for all i, j . It is said to be *upward (downward) graded on its columns* if and only if C^T is downward (upward) graded on its rows. A matrix is said to be *doubly graded* if and only if it is graded (upward or downward) on both its rows and its columns. We call a matrix graded upward on its rows and downward on its columns a *(UD)-graded matrix* and we similarly define *(UU)-graded*, *(DU)-graded* and *(DD)-graded matrices*.

A matrix C is *diagonally outward graded* if and only if

- (i) $c_{ij} \leq c_{i,j+1}$ for all $i \in N$ and $j = i, \dots, n-1$;
- (ii) $c_{ij} \geq c_{i,j+1}$ for all $i \in N$ and $j = 1, \dots, i-1$; and
- (iii) C^T satisfies conditions (i) and (ii).

C is *diagonally inward graded* if and only if $-C$ is diagonally outward graded.

Every doubly graded matrix C can be transformed into a (UU)-graded matrix or a (UD)-graded matrix by renumbering the elements of N and reordering the rows and columns of C accordingly. Every (UU)-graded matrix is a max-distribution matrix. Also, if C is diagonally outward graded, then it is a max-distribution matrix [801]. Hence by Corollary 26, in each of these cases $BTSP(C)$ is pyramidally solvable. In the former case, the following stronger result is proved in [361].

Theorem 27 [361] *If C is a (UU)-graded matrix, then $\gamma = (1, 2, \dots, n, 1)$ is an optimal solution to $BTSP(C)$.*

Proof. Since C is a (UU)-graded matrix, $c_{\max}(\gamma) = \max\{c_{i,i+1} : i \in \{1, 2, \dots, n-1\}\} = c_{u,u+1}$ for some u . Now for any tour ψ on N , there exist i, v such that $i \leq u < v$ and $\psi(i) = v$. But then $c_{\max}(\psi) \geq c_{iv} \geq c_{u,u+1}$. ■

Using the same arguments as in the proof of Theorem 24, we can prove the following result.

Theorem 28 *Suppose matrix C satisfies the following conditions:*

- (i) *for any distinct i, j, k such that $1 \leq i, j < k < n$, $\max\{c_{ik}, c_{kj}\} \geq c_{ij}$;*
- (ii) *for any i, j such that $1 < i < j \leq n$, $c_{j,i-1} \geq \max\{c_{ji}, c_{i,i-1}\}$;*
- (iii) *C^T satisfies (i) and (ii).*

Then, $BTSP(C)$ is pyramidally solvable.

A matrix C is a max-Klyaus matrix if and only if for any i, j, k such that $i < j < k$,

$$c_{ki} \geq \max\{c_{kj}, c_{ji}\} \text{ and } c_{ik} \geq \max\{c_{ij}, c_{jk}\}.$$

A max-Klyaus matrix satisfies both property max-BK-II(4) and the conditions of Theorem 28. Hence, as a corollary to theorem 28, we get the following.

Corollary 29 [147] *If C is a max-Klyaus matrix, then $BTSP(C)$ is pyramidally solvable.*

For other pyramidally solvable cases of BTSP, we refer the reader to [801].

4.2. Subclass of BTSP Solvable Using GG-Patchings

We now modify the GG scheme for TSP discussed in Chapter 11 in the context of BTSP. We call it BTGG scheme.

For any $S \subseteq \{1, 2, \dots, n - 1\}$, let $P(S)$ be the set of all permutations of elements of S . We denote

$$d[C, S] = \min\{c_{\max}(\psi) : \psi = \beta_{i_1} \circ \dots \circ \beta_{i_u}, (i_1, \dots, i_u) \in P(S)\}.$$

(See Appendix A for the definitions of permutation β_i and operation \circ .) For any cost matrix C and any two permutations μ and σ on N , the permuted cost matrix $C^{\mu, \sigma}$ is defined as $c_{ij}^{\mu, \sigma} = c_{\mu(i), \sigma(j)}$. We denote $C^{\xi, \sigma}$ by C^σ , where ξ is the identity permutation.

The BTGG scheme works as follows. We start with a given permutation π . If π is a tour, the algorithm outputs this tour. Otherwise, we patch the subtours of π by constructing a patching pseudograph $G_p^\pi = (N_p^\pi, E_p^\pi)$, finding an optimal spanning tree $E_p^\pi[T^*]$ of G_p^π , finding an optimal ordering of elements of $E_p^\pi[T^*]$ and performing patching operations corresponding to the edges in $E_p^\pi[T^*]$ in the optimal order. For details of various terms used above and insight into the basics of the

algorithm we refer to Chapter 11. A formal description of the algorithm is given below.

Algorithm : BTGG Scheme

Input: An $n \times n$ cost matrix C and a permutation π on N .

Step 1: If π is a tour then output π and stop. Suppose π has m subtours, for some $m > 1$, on node sets N_1, \dots, N_m .

Step 2: Construct the patching pseudograph $G_p^\pi = (N_p^\pi, E_p^\pi)$ where, $N_p^\pi = \{1, \dots, m\}$ and $E_p^\pi = \{e_i = (u, v) : 1 \leq i < n; i \in N_u; (i+1) \in N_v\}$.

Step 3: Find a spanning tree in G_p^π with edge set $E_p^\pi[T^*]$ such that $d[C^\pi, T^*]$ is minimum. Let $(i_1, i_2, \dots, i_{m-1})$ be an ordering of the elements of T^* such that $d[C^\pi, T^*] = c_{\max}(\pi \circ \beta_{i_1} \circ \dots \circ \beta_{i_{m-1}})$. The tour $\gamma^* = \pi \circ \beta_{i_1} \circ \dots \circ \beta_{i_{m-1}}$ is the desired output. Stop.

4.2.1 Sufficiency Conditions for Validity of BTGG Scheme.

Now we give sets of sufficiency conditions on matrix C' under which for any C and π such that $C^\pi = C'$, the BTGG Scheme, with π as the starting permutation, produces an optimal solution to BTSP(C). For any set $Y \subseteq N$ with at least two elements, we define the range of Y as the set $\{i, i+1, \dots, j\}$ such that $i \leq j$, $Y \subseteq \{i, i+1, \dots, j+1\}$ and i and $j+1$ are in Y and we denote it by $[i, j]$. For definitions of various other terms used below, we refer to Chapter 11.

Definition 30 An $n \times n$ matrix C satisfies max patching property I (MPP-I) if and only if the following condition holds.

Let ψ be an arbitrary permutation on N with ℓ non-trivial subtours on the vertex sets N^1, N^2, \dots, N^ℓ . Let $\{S_1, S_2, \dots, S_\ell\}$ be pairwise disjoint subsets of N and $S = \bigcup_{i=1}^\ell S_i$ such that

(i) for each i , each element of S_i lies in the range of N^i and each region of N^i contains at the most one element of S_i ; and

(ii) $|S| \equiv (\sum_{i=1}^\ell (|N^i| - 1)) \pmod{2}$.

Then, $c_{\max}(\psi) \geq d[C, S]$.

Theorem 31 If an $n \times n$ matrix C' satisfies the property MPP-I, then for any cost matrix C and any permutation π on N such that $C' = C^\pi$, the BTGG scheme with π as the starting permutation produces an optimal solution to BTSP(C).

Proof. Suppose C' satisfies the property MPP-I and let a cost matrix C and a permutation π on N be such that $C' = C^\pi$. Let η be an optimal

solution to BTSP(C) and let $\psi = \pi^{-1} \circ \eta$. Suppose ψ has ℓ non-trivial subtours on node sets N^1, N^2, \dots, N^ℓ . By Corollary 42 of Chapter 11, there exists a subset S of N with partition $\{S_1, S_2, \dots, S_\ell\}$ such that (i) for each i , every element of S_i lies in the range N^i and every region of N^i contains at the most one element of S_i ; (ii) $E_p^\pi[S]$ is the edge set of a spanning tree of the patching pseudograph G_p^π , and therefore, (iii) $|S| \equiv ((\sum_{i=1}^\ell (|N^i| - 1) \pmod{2})$. Let (i_1, i_2, \dots, i_m) be an ordering of the elements of S for which $c_{\max}^\pi(\beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_m}) = d[C^\pi, S]$. Then, $\gamma^* = \pi \circ \beta_{i_1} \circ \beta_{i_2} \circ \dots \circ \beta_{i_m}$ is a tour on N , $c_{\max}(\eta) = c_{\max}^\pi(\psi)$ and $c_{\max}(\gamma^*) = d[C^\pi, S]$. Hence, $c_{\max}(\eta) \geq c_{\max}(\gamma^*)$ and γ^* is an optimal solution to BTSP(C). ■

Definition 32 A matrix C satisfies max patching property II (MPP-II) if and only if the following condition holds.

Let ψ be an arbitrary permutation on N . Suppose ψ has ℓ non-trivial subtours and G_ψ^I (for definition, see Chapter 11, Section 4.2) has r connected components of sizes v_1, v_2, \dots, v_r . Let $X_\psi^1, X_\psi^2, \dots, X_\psi^r$ be the unions of the node sets of the non-trivial subtours of ψ corresponding, respectively, to nodes of the r connected components of G_ψ^I . Let S be any subset of N with a partition $\{S_1, S_2, \dots, S_r\}$, such that

- (i) for each i , every element of S_i lies in the range X_ψ^i , every region of X_ψ^i contains at the most one element of S_i and $|S_i| \leq (|X_\psi^i| - v_i)$; and
- (ii) $|S| \equiv ((\sum_{i=1}^r |X_\psi^i|) - \ell) \pmod{2}$. Then $c_{\max}(\psi) \geq d[C, S]$.

Theorem 33 If a matrix C' satisfies property MPP-II then for any cost matrix C and any permutation π on N such that $C' = C^\pi$, the BTGG scheme with π as the starting permutation produces an optimal solution to BTSP(C).

Theorem 33 can be proved along the same lines as Theorem 31 using Corollary 43 of Chapter 11.

As a corollary, we get the following result.

Corollary 34 Suppose C^π is a max-distribution matrix. Then the BTGG scheme with π as the starting permutation produces an optimal solution to BTSP(C).

Outline of Proof. It will be sufficient to show that C^π satisfies the property MMP-II. Thus, consider an arbitrary permutation ψ on N . Suppose ψ has ℓ non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_\ell$ and G_ψ^I has r

connected components of sizes v_1, v_2, \dots, v_r . Let $X_\psi^1, X_\psi^2, \dots, X_\psi^r$ be the unions of the node sets of the non-trivial subtours of ψ corresponding, respectively, to nodes of the r connected components of G_ψ^I and let $[p_1, q_1], \dots, [p_r, q_r]$ be the ranges of the sets $X_\psi^1, \dots, X_\psi^r$, respectively. Let S be any subset of N with a partition $\{S_1, S_2, \dots, S_r\}$, such that for any $1 \leq i \leq r$, every element of S_i lies in the range X_ψ^i , every region of X_ψ^i contains at the most one element of S_i and $|S_i| \leq (|X_\psi^i| - v_i)$.

For any distinct i, j belonging to the same connected component of G_ψ^I , there exist nodes k, u of \mathcal{C}_i and \mathcal{C}_j , respectively, such that the ranges of $\{k, \psi(k)\}$ and $\{u, \psi(u)\}$ intersect and $c_{\max}^\pi(\psi \circ \alpha_{k,u}) \leq c_{\max}^\pi(\psi)$. Let $\psi' = \psi \circ \alpha_{k,u}$. By repeating this process, we get a permutation η such that $c_{\max}^\pi(\eta) \leq c_{\max}^\pi(\psi)$ and the node sets of the non-trivial subtours of η are $X_\psi^1, X_\psi^2, \dots, X_\psi^r$.

For any $i, 1 \leq i \leq r$, and any $j \in [p_i, q_i] - X_\psi^i$, there exists $u \in X_\psi^i$ such that $u < j < \eta(u)$. Let $\eta' = \eta \circ \alpha_{u,j}$. Then $c_{\max}^\pi(\eta') \leq c_{\max}^\pi(\eta)$. By repeating this process, we get a permutation ϕ such that $c_{\max}^\pi(\phi) \leq c_{\max}^\pi(\eta)$ and ϕ is dense on the node set $X = \cup_{i=1}^r \{p_i, \dots, q_i\}$. Since C^π is a max-distribution matrix, it follows by Corollary 26 that $d[C^\pi, X] \leq c_{\max}^\pi(\phi)$. The result now follows from the fact that since $S \subseteq X$, $d[C^\pi, S] \leq d[C^\pi, X]$.

We do not know the complexity of implementation of the BTGG scheme when C^π is a max-distribution matrix. However when the patching pseudograph G_p^π is a multi-path or a multi-tree, BTGG-scheme can be implemented in polynomial time using the same idea as in the case of GG-scheme discussed in Chapter 11.

For an arbitrary permutation π on N , let N^1, \dots, N^k be the node sets of the non-trivial subtours, (that is, subtours with at least two nodes) of π . Let

$$d^{in}(C, \pi) = \max\{c_{i, \pi(i)} : i \in \cup_{j=1}^k N^j\}.$$

For any $S \subseteq \{1, 2, \dots, n-1\}$, let $P(S)$ be the set of all permutations of elements of S . Let

$$d^{in}[C, S] = \min\{d^{in}(C, \psi) : \psi = \beta_{i_1} \circ \dots \circ \beta_{i_u} \text{ and } (i_1, \dots, i_u) \in P(S)\}.$$

Suppose π is an optimal solution to the bottleneck assignment problem on C (and therefore, the identity permutation ξ is an optimal solution to the bottleneck assignment problem on C^π). Then, if we modify the properties MPP-I and MPP-II as well as the BTGG scheme by replacing the functions $c_{\max}(\pi)$ and $d[C, S]$ by functions $d^{in}(C, \pi)$ and $d^{in}[C, S]$, respectively, the modified properties can be shown to be sufficient for the modified BTGG scheme to produce an optimal solution to BTSP(C). It

may be noted that if C^π is a max-distribution matrix, then π is an optimal solution to the bottleneck assignment problem on C . If C^π is a (UU)-graded matrix, then by Theorem 27, for $S = \{u, u + 1, \dots, v\}$ for any $1 \leq u < v < n$, $d^{in}[C^\pi, S] = \max\{c_{i,i+1}^\pi : i \in S\}$. Thus, Step 3 of the modified BTGG scheme reduces to a bottleneck spanning tree problem, where each edge $e_i \in E_p^\pi$ is assigned a weight $c_{i,i+1}^\pi$, and this problem can be solved in $O(n)$ [361]. We thus have the following result.

Corollary 35 [361] *If C^π is a (UU)-graded matrix, then the BTGG-scheme with π as the starting permutation and with the function $d[C^\pi, T^*]$ replaced by the function $d^{in}[C^\pi, T^*]$ produces an optimal solution to BTSP(C) in $O(n)$ time.*

If C^π is a (UD)-graded matrix then for permutation ψ defined as $\psi(i) = n - i + 1$, $C^{\pi \circ \psi}$ is a (UU)-graded matrix. Thus, BTSP when C^π is a (UD)-graded matrix can be solved in $O(n)$ time. Suppose C is a Gilmore-Gomory type matrix (see Chapter 11) with $f(x) \geq 0$ and $g(x) = 0$ for all x and let us assume, without loss of generality, that $b_1 \leq \dots \leq b_n$. If we define a permutation π on N such that $a_{\pi(1)} \leq \dots \leq a_{\pi(n)}$, then C^π is a (UU)-graded matrix. Finding such a permutation π takes $O(n \log n)$ time. Thus, in this case BTSP(C) can be solved in $O(n \log n)$ time [360].

If C is a sum matrix, (that is, $c_{ij} = a_i + b_j$ for all i, j), or a small matrix, (that is, $c_{ij} = \min\{a_i, b_j\}$), then let us assume without loss of generality that $a_1 \geq \dots \geq a_n$. If we define π such that $b_{\pi(1)} \leq \dots \leq b_{\pi(n)}$, then in each of these cases, C^π is a (UU)-graded matrix. Again, finding such a permutation π takes $O(n \log n)$ time. Hence, in each of these cases BTSP(C) can be solved in $O(n \log n)$ time. For the small matrix case, BTGG scheme can be implemented in $O(n)$ time using the same idea as in the case of TSP on such matrices, as discussed in Chapter 11, Section 4.3. An alternate $O(n)$ scheme for the small matrix case is given in [798] that is similar to the scheme by Gabovich [340] for TSP on such matrices. If C is a large matrix, then every tour on N has the same bottleneck cost and thus the problem is trivial.

In [361] it is shown that if the cost matrix C is a graded matrix, then BTSP(C) can be solved in $O(n^2)$ time using a modification of the BTGG scheme that allows patchings that are not GG-patchings.

4.3. BTSP on Ordered Product Matrices

A cost matrix C is called an *ordered product matrix* if and only if there exist $\{a_i, b_i : i \in N\}$ such that $a_1 \leq a_2 \leq \dots \leq a_n$, $b_1 \geq b_2 \geq \dots \geq b_n$ and $c_{ij} = a_i b_j$.

The following result by Van Der Veen [801] improved upon the previous $O(n^2)$ algorithm of [147] for BTSP on ordered product matrices.

Theorem 36 [801] *BTSP on an ordered product cost matrix can be solved in $O(n)$ time.*

Proof. If all the a_i 's have the same sign or all the b_j 's have the same sign then the cost matrix C is a doubly graded matrix and an $O(n)$ algorithm follows from Corollary 35. Suppose there exist $1 \leq u, v < n$ such that $a_u < 0 \leq a_{u+1}$ and $b_v \geq 0 > b_{v+1}$.

Case (i): $u = v$. Let $m = n - v$. In this case, we can partition the matrix C as

$$C = \begin{bmatrix} D^{1,1} & D^{1,2} \\ D^{2,1} & D^{2,2} \end{bmatrix},$$

where matrices $D^{1,1}, D^{1,2}, D^{2,1}$ and $D^{2,2}$ are of sizes $u \times u$, $u \times m$, $m \times u$ and $m \times m$, respectively, and $D^{1,1} \leq 0$, $D^{2,2} \leq 0$, $D^{1,2} > 0$ and $D^{2,1} \geq 0$. Obviously, for every tour γ on N , $c_{\max}(\gamma) \geq 0$ and hence, using Lemma 19, we consider matrix C' where, $c'_{ij} = \max\{C_{i,j}, 0\}$. The matrix C' has the form:

$$C' = \begin{bmatrix} 0 & A^{1,2} \\ A^{2,1} & 0 \end{bmatrix},$$

where $A^{1,2}$ is a positive (UU)-graded matrix and $A^{2,1}$ is a non-negative (DD)-graded matrix and therefore, C' is a max-distribution matrix. Hence, BTSP(C') is pyramidally solvable. For any pyramidal tour γ on N there is exactly one pair $\{i, j\} \subseteq N$ such that $i \leq u < \gamma(i)$ and $j > u \geq \gamma(j)$ and $c'_{\max}(\gamma) = \max\{c'_{i,\gamma(i)}, c'_{j,\gamma(j)}\}$. Since $A^{1,2}$ is a (UU)-graded matrix and $A^{2,1}$ is a (DD)-graded matrix, an optimal choice of such a pair of arcs $\{(i, \gamma(i)), (j, \gamma(j))\}$ belongs to the set

$$S = \{ \{(u, u + 2), (u + 1, u - 1)\}, \{(u - 1, u + 1), (u + 2, u)\}, \\ \{(u - 1, u + 2), (u + 1, u)\}, \{(u, u + 1), (u + 2, u - 1)\} \}.$$

An optimal tour can thus be obtained by finding $\{(x, y), (s, t)\} \in S$ with minimum value of $\max\{c_{xy}, c_{st}\}$ and constructing a pyramidal tour containing these arcs.

Case (ii): $u < v$. Let $\ell = v - u$ and $m = n - v$. We can partition the matrix C as

$$C = \begin{bmatrix} D^{1,1} & D^{1,2} & D^{1,3} \\ D^{2,1} & D^{2,2} & D^{2,3} \\ D^{3,1} & D^{3,2} & D^{3,3} \end{bmatrix},$$

where $D^{1,1} \leq 0$, $D^{1,2} \leq 0$, $D^{1,3} \geq 0$, $D^{2,1} \geq 0$, $D^{2,2} \geq 0$, $D^{2,3} \leq 0$, $D^{3,1} \geq 0$, $D^{3,2} \geq 0$ and $D^{3,3} \leq 0$. In this case too, it is easy to see that $c_{\max}(\gamma) \geq$

0 for every tour γ on N and hence, using Lemma 19, we consider matrix C' , where for all $1 \leq i, j \leq n$, $c'_{ij} = \max\{c_{ij}, 0\}$. We can partition C' as

$$C' = \begin{bmatrix} 0 & 0 & A^{1,3} \\ A^{2,1} & A^{2,2} & 0 \\ A^{3,1} & A^{3,2} & 0 \end{bmatrix},$$

where $A^{1,3}$ is a non-negative (UU)-graded matrix and each of $A^{2,1}, A^{2,2}, A^{3,1}$ and $A^{3,2}$ is a non-negative (DD)-graded matrix. Therefore, C' is a max-distribution matrix and hence, $\text{BTSP}(C')$ is pyramidally solvable. The bottleneck cost of any tour on N will be decided by the arcs in the tour corresponding to the entries of the matrices $A^{1,3}, A^{2,1}, A^{2,2}, A^{3,1}$ and $A^{3,2}$. Using the structure of these matrices, it has been shown in [801] that there exists an optimal pyramidal tour for which the set of the arcs of the tour corresponding to the entries of these five matrices belongs to the set

$$S = \{\{(x, x - 2)\}, \{(u - 1, v + 2)\}, \{(v + 2, v), (u - 1, v + 1)\}, \\ \{(u + 1, u - 1), (u, v + 2)\}, \{(u + 1, u - 1), (v + 2, v), (u, v + 1)\}\},$$

where $x \in \{u + 2, \dots, v + 1\}$ is such that $c'_{x, x-2} = \min\{c'_{i, i-2} : i \in \{u + 2, \dots, v + 1\}\}$. Thus, an optimal pyramidal tour can be obtained in $O(n)$ time.

Case (iii): $u > v$. In this case, the result can be proved along the same lines as in case (ii). ■

4.4. BTSP on Symmetric, Diagonally Circular Inward Graded Matrix

In [33], $O(n)$ algorithms are presented for two subclasses of MSTSP. In the first case, each node in N is associated with a point on a line, while in the second case, each node is associated with a point on a circle. In each case, for any $i, j \in N$, weight w_{ij} is the Euclidean distance between the corresponding points. We consider this as instances of BTSP with edge costs $c_{ij} = -w_{ij}$. The case corresponding to points on a line is obviously a special case of the case of points on a circle. In the case of points on a line, the cost matrix is a symmetric, diagonally inward graded matrix and it is a Gilmore-Gomory type matrix (see Chapter 11) with $a_i = b_i$ for all $i \in N$ and $f(x) = g(x) \leq 0$ for all x . In the case of points on a circle, the cost matrix is a special case of what we call a symmetric, diagonally circular inward graded matrix (SDCI-matrix), and which we define below. We assume here that all indices and node labels are taken *modulo* n .

A symmetric cost matrix C is an *SDCI-matrix* if only if there exist $1 \leq \ell(1) \leq \ell(2) \leq \dots \leq \ell(n) < 2n$ such that for all $i \in N$,

- (i) $i \leq \ell(i) \leq n + i$;
- (ii) $c_{i,x-1} \geq c_{ix}$ for all $x \in \{i + 1, \dots, \ell(i)\}$ and $c_{i,x+1} \geq c_{ix}$ for all $x \in \{\ell(i) + 1, \dots, n + i - 1\}$;
- (iii) $c_{x+1,y} \leq c_{xy}$ for all $y \geq \ell(i) + 1$ and $x = i - 1, i - 2, \dots, y - n$ and $c_{x+1,y} \geq c_{xy}$ for all $y \leq \ell(i)$ and $x = i, i + 1, \dots, y - 1$.

We say that C is an SDCI-matrix with respect to $\{\ell(1), \ell(2), \dots, \ell(n)\}$.

It should be noted that a symmetric, diagonally inward graded matrix is an SDCI-matrix with $\ell(i) = n$ for all i ; and the negative of a symmetric diagonally inward graded matrix is a max-Klyaus matrix. In the case of points on a circle, if for each point i , we define $\ell(i)$ such that each of the sets of points $\{i, i + 1, \dots, \ell(i)\}$ and $\{i, i - 1, \dots, \ell(i) + 1\}$ lies on an arc subtending an angle less than 180° at the centre of the circle, then the cost matrix is an SDCI-matrix with respect to these values of $\{\ell(1), \ell(2), \dots, \ell(n)\}$.

For any set $S \subset N$, let us define $\lambda_S = \min\{c_{ij} : i, j \in S\}$.

The following facts given in [33] will be useful. Facts 37 and 38 are easy to verify and Fact 39 follows from Fact 38.

Fact 37 *If $n = 2k + 1$, for some positive integer k , then for any subset S of N of cardinality $k + 1$ and any tour γ on N , there exist u, v in S such that $\gamma(u) = v$. Therefore, λ_S is a lower bound on the optimal objective function value of BTSP(C).*

Fact 38 *If $n = 2k$, for some positive integer k , then for any pair of subsets S_1 and S_2 of N of cardinality k each, such that $S_1 \cap S_2 \neq \emptyset$, and any tour γ on N , either $\{u, \gamma(u)\} \subseteq S_1$ for some u or $\{v, \gamma(v)\} \subseteq S_2$ for some v . Hence, $\min\{\lambda_{S_1}, \lambda_{S_2}\}$ is a lower bound on the optimal objective function value of BTSP(C).*

Fact 39 *Suppose $n = 2k$, for some positive integer k . Let S_1, S_2 and S_3 be subsets of N of cardinality k each such that $\lambda_{S_1} \geq \lambda_{S_2} \geq \lambda_{S_3}$ and $\lambda_{S_3} \geq \lambda_S$ for any other subset S of N of cardinality k . Then, the following are true.*

- (i) λ_{S_3} is a lower bound on the optimal objective function value of BTSP(C).
- (ii) If $S_1 \cap S_2 \neq \emptyset$, then λ_{S_2} is a lower bound on the optimal objective function value of BTSP(C).

Theorem 40 *If C is an SDCI-matrix then BTSP(C) can be solved in $O(n)$ time.*

Proof. We follow the proof technique in [33] for the case of points on a circle.

Case (i) $n = 2k + 1$ for some k : For any $i \in N$, let $F_i = \{i, i + 1, \dots, i + k\}$ and $B_i = \{i, i - 1, \dots, i - k\}$. It follows from Fact 37 that $\lambda = \max\{\lambda_{F_i}, \lambda_{B_i} : 1 \leq i \leq n\}$ is a lower bound to the optimal objective function value of BTSP(C). Since, C is an SDCI-matrix, $\max\{\lambda_{F_i}, \lambda_{B_i}\} = \max\{c_{i,i+k}, c_{i,i-k}\}$ for all $i \in N$ and hence, $\lambda = \max\{c_{i,i+k}, c_{i,i-k} : i \in N\}$. Let $\gamma^* = (1, k + 2, 2k + 3, \dots, 2k(k + 1) + 1, 1)$. Then, $c_{\max}(\gamma^*) = \lambda$ and, it follows from the fact that $\gcd\{n, k + 1\} = 1$ that γ^* is a tour on N . Hence, γ^* is an optimal solution to BTSP(C).

Case (ii) $n = 2k$: In this case, let us define for any $i \in N$, $F_i = \{i, i + 1, \dots, i + k - 1\}$. Since C is an SDCI-matrix, we have the following. For any $i \in N$,

(i) if $\ell(i) \geq i + k - 1$, then $c_{i,i+k-1} = \lambda_{F_i}$;

(ii) if $\ell(i) < i + k - 1$, then $c_{i,i+k-1} \leq \min\{c_{i,i+k+1}, c_{i-1,i+k}, c_{i-2,i+k-1}\}$, and from (i) we have that $c_{i,i+k+1} = \lambda_{F_{i+k+1}}$, $c_{i-1,i+k} = \lambda_{F_{i+k}}$ and $c_{i-2,i+k-1} = \lambda_{F_{i+k-1}}$. Hence, if i_1, i_2, i_3 in N are such that $\lambda_{F_{i_1}} \geq \lambda_{F_{i_2}} \geq \lambda_{F_{i_3}}$ and $\lambda_{F_{i_3}} \geq \lambda_{F_i}$ for every other i , then $\lambda_{F_{i_j}} = c_{i_j, i_j+k-1}$ for $j = 1, 2, 3$. It now follows from Fact 39 that if $\{i_1, \dots, i_1 + k - 1\} \cap \{i_2, \dots, i_2 + k - 1\} \neq \emptyset$, then c_{i_2, i_2+k-1} is a lower bound on the optimal objective function value of BTSP(C); else, c_{i_3, i_3+k-1} is a lower bound. Consider the tour $\gamma = (i_1, (k + 1 + i_1), (2(k + 1) + i_1), \dots, (2k(k + 1) + i_1), i_1) = (i_1, u_1, \dots, u_{2k}, i_1)$. Let j be minimum index such that $u_j \in \{i_1 + k - 1, i_1 + 2k - 1\}$. Let γ^* be the tour $(i_1, u_1, \dots, u_j, u_{2k}, u_{2k-1}, u_{j+1}, i_1)$. The tour γ^* attains the lower bound stated above. ■

4.5. BTSP on Symmetric, Circulant Digraph

The complexities of both TSP and BTSP on a circulant digraph (See definition in Chapter 11 ,Section 10.5) are open. An $O(n \log n)$ scheme for BTSP on symmetric circulant digraphs is given in [147] and we discuss it next.

Theorem 41 [147] *Let $\hat{G} = (N, \hat{E})$ be the undirected graph obtained from a symmetric circulant digraph $G(n, a_1, -a_1, a_2, -a_2, \dots, a_m, -a_m)$, with $m < n$, by replacing every pair of arcs $\{(i, j), (j, i)\}$ by edge (i, j) . Let $\gcd\{n, a_1, \dots, a_m\} = g$. Then \hat{G} has precisely g connected components $\hat{G}_1, \dots, \hat{G}_g$ on node sets $N_i = \{i + kg : k \in \{0, \dots, (n/g - 1)\}\}$, respectively, and each connected component is Hamiltonian.*

Proof. Let us prove the result by induction on m . For $m = 1$, let $\gcd\{n, a_1\} = g_1$. In this case, the result follows from elementary results in number theory [434] and for each $i \in \{1, \dots, g\}$, the connected component \hat{G}_i has a Hamiltonian tour $\gamma_i^1 = (i, (i + a_1) \bmod n, \dots, (i + (n/g - 1)a_1) \bmod n, i)$.

Suppose now that the result is true for all $m \leq u$ for some $u < n$. Let us prove the result for $m = u$. Let $\gcd\{n, a_1, \dots, a_u\} = g_u$ and let $\gcd\{n, a_1, \dots, a_{u-1}\} = g_{u-1}$. Let $n = rg_{u-1}$ and $g_{u-1} = kg_u$. By induction, \hat{G}' , the undirected graph corresponding to $G(n, a_1, -a_1, \dots, a_{u-1}, -a_{u-1})$, has g_{u-1} connected components, each having a Hamiltonian tour. It follows from elementary number theory [434] that the nodes $x_1 = 1, x_2 = 1 + a_u, \dots, x_k = 1 + (k-1)a_u$ belong to different connected components of \hat{G}' which we denote by $\hat{G}'_1, \hat{G}'_2, \dots, \hat{G}'_k$, respectively. Also, edges $\{(x_i, x_{i+1}) : i \in \{1, \dots, k-1\}\}$ are in \hat{G} and the k components $\hat{G}'_1, \dots, \hat{G}'_k$ combine to form a connected component H_1 of H . Let $((1 =)i_0, i_1, \dots, i_s, i_0)$ be a tour in \hat{G}'_1 . Then, by symmetry, it follows that $(x_j, i_1 + x_j - 1, \dots, i_s + x_j - 1, x_j)$ is a tour in \hat{G}'_j for $j \in \{1, \dots, k\}$ and edges $\{(x_j + i_v - 1, x_{j+1} + i_v - 1) : j \in \{1, \dots, k-1\}, v \in \{0, \dots, s\}\}$ are in \hat{G} . It is now easy to verify that the following process produces a tour in the \hat{G}_1 . Remove edges $(i_s, 1)$ and $(i_s + x_2 - 1, x_2)$ and add edges $(1, x_2)$ and $(i_s, i_s + x_2 - 1)$; recursively, for $j = 2, \dots, k-1$, remove edges $(x_j + i_{j-2} - 1, x_j + i_{(j-1)} - 1)$ and $(x_{j+1} + i_{j-2} - 1, x_{j+1} + i_{j-1} - 1)$ and add edges $(x_j + i_{j-2} - 1, x_{j+1} + i_{j-2} - 1)$ and $(x_j + i_{j-1} - 1, x_{j+1} + i_{j-1} - 1)$, where for any $j > s$, $i_j = i_{j \bmod s}$. Because of symmetry, similar method can be used to obtain a tour in every other connected component of H .

■

Now consider an instance BTSP(C) where for some symmetric circulant digraph $G(n, a_1, -a_1, a_2, -a_2, \dots, a_m, -a_m)$ and numbers d_1, \dots, d_m , the cost matrix C is defined as follows: for any $1 \leq i, j \leq n$, if $|i - j| = a_u \bmod n$ for some $u \in \{1, \dots, m\}$, then $c_{ij} = d_u$; else, $c_{ij} = \infty$. The following $O(n \log n)$ scheme for this problem, based on Theorem 41, is given in [147]. Without loss of generality, let us assume that $d_1 \leq d_2 \leq \dots \leq d_m$. For each $i = 1, \dots, m$, find $\gcd\{n, a_1, \dots, a_i\}$. All these gcd values can be calculated in $O(n \log n)$ time. Find minimum $u \in \{1, \dots, m\}$ such that $\gcd\{n, a_1, \dots, a_u\} = 1$. Find a tour γ^* in $G(n, a_1, -a_1, a_2, -a_2, \dots, a_u, -a_u)$ as in the proof of Theorem 41. Then γ^* is the desired optimal solution.

4.6. BTSP on a Halin Graph

In this section we consider the BTSP restricted to a Halin graph. In Chapter 11, we have seen that TSP on a Halin graph with n nodes can be solved in $O(n)$ time. Using this algorithm within the binary search version of the threshold algorithm [269] for bottleneck problems, BTSP on a Halin graph can be solved in $O(n \log n)$ time. We now discuss an $O(n)$ algorithm to solve BTSP on a Halin graph [668].

Recall that a Halin graph G is obtained by embedding a tree T with no node of degree two in the plane and joining the leaf nodes by a cycle so that the resulting graph is planar. (see appendix A.) The nodes of G corresponding to the pendant nodes of T are called *outer nodes*. All other nodes are *internal nodes*. A Halin graph with exactly one internal node is called a *wheel*. It is easy to see that BTSP on a wheel can be solved in $O(n)$ time. As in the case of TSP on a Halin graph discussed in Chapter 11, we solve BTSP on a Halin graph by recursively collapsing fans. However, unlike the TSP, for BTSP we need an auxiliary objective function to record information generated during the iterations, and to maintain the invariant property of the objective function value in each iteration.

Let \mathbb{F} be the family of all the Hamiltonian cycles (tours) in G . For each outer node i of G , let $e_{i,1}$, $e_{i,2}$ and $e_{i,3}$ be the edges incident to i . Any tour \mathcal{H} in \mathbb{F} uses precisely two of these three edges. Let $p(e_{i,1}, e_{i,2})$ be the ‘penalty’ incurred if \mathcal{H} uses the edges $e_{i,1}$ and $e_{i,2}$; $p(e_{i,2}, e_{i,3})$ be the ‘penalty’ incurred if \mathcal{H} uses the edges $e_{i,2}$ and $e_{i,3}$; and $p(e_{i,1}, e_{i,3})$ be the ‘penalty’ incurred if \mathcal{H} uses the edges $e_{i,1}$ and $e_{i,3}$. Thus, for the tour \mathcal{H} and outer node i , define

$$P_i(\mathcal{H}) = \begin{cases} p(e_{i,1}, e_{i,2}) & \text{if } e_{i,1} \text{ and } e_{i,2} \in \mathcal{H} \\ p(e_{i,2}, e_{i,3}) & \text{if } e_{i,2} \text{ and } e_{i,3} \in \mathcal{H} \\ p(e_{i,1}, e_{i,3}) & \text{if } e_{i,1} \text{ and } e_{i,3} \in \mathcal{H} \end{cases} \quad (8)$$

Consider the modified bottleneck TSP on a Halin graph defined as follows:

$$\begin{array}{ll} \text{MBTSP:} & \text{Minimize } f(\mathcal{H}) \\ & \text{Subject to} \\ & \mathcal{H} \in \mathbb{F}, \end{array}$$

where

$$f(\mathcal{H}) = \max\{\max\{c_e | e \in \mathcal{H}\}, \max\{P_i(\mathcal{H}) | i \text{ is an outer node of } G\}\}$$

Note that for each outer node i and tour \mathcal{H} in \mathbb{F} , $P_i(\mathcal{H})$ can be identified in $O(1)$ time using the triplet $[p(e_{i,1}, e_{i,2}), p(e_{i,1}, e_{i,3}), p(e_{i,2}, e_{i,3})]$, called *penalty triplet*. At the beginning of the algorithm, we may choose $p(e_{i,1}, e_{i,2}) = p(e_{i,1}, e_{i,3}) = p(e_{i,2}, e_{i,3}) = 0$. Thus, with this choice of the penalty triplets, MBTSP is equivalent to BTSP. In our algorithm, we successively collapse fans into a pseudo-nodes and then update the values of the penalty triplets. Note that this fan collapsing scheme will eventually result in a wheel and hence we have to solve MBTSP on a wheel.

4.6.1 Algorithm for MBTSP on a wheel. Consider a wheel $W = (V, E)$ with u as its internal node. Let $1, 2, \dots, n$ be the outer nodes such that $e_i = (i, i + 1) \in E$, $i = 1, 2, \dots, n$, where $n + 1 \equiv 1$. Consider the cycle $\Delta = (1, 2, \dots, n, 1)$. Let e^* and e^{**} be edges of Δ such that

$$c_{e^*} = \max\{c_e \mid e \in \Delta\}$$

and

$$c_{e^{**}} = \max\{c_e \mid e \in \Delta - \{e^*\}\}.$$

Every tour on W is obtained by deleting from Δ some edge e_r and introducing edges (u, r) and $(u, r + 1)$. We denote this tour by \mathcal{H}_r . Thus, $P_r(\mathcal{H}_r) = p(e_{r-1}, (u, r))$, $P_{r+1}(\mathcal{H}_r) = p(e_{r+1}, (u, r + 1))$, and for every other $i : i \neq r, r + 1$, $P_i(\mathcal{H}_r) = p(e_i, e_{i+1})$. Define

$$\delta_i = \max\{c_{(u,i)}, c_{(u,i+1)}, p(e_{i+1}, (u, i + 1)), p(e_{i-1}, (u, i))\}.$$

Choose r, s, t such that

$$\begin{aligned} p(e_r, e_{r+1}) &= \max\{p(e_i, e_{i+1}) \mid e_i \in \Delta\} \\ p(e_s, e_{s+1}) &= \max\{p(e_i, e_{i+1}) \mid e_i \in \Delta, e_i \neq e_r\} \\ p(e_t, e_{t+1}) &= \max\{p(e_i, e_{i+1}) \mid e_i \in \Delta, e_i \neq e_r, e_s\}. \end{aligned}$$

(Here, $e_{n+1} = e_1$.) Now, given the values of $\delta_i, p(e_r, e_{r+1}), p(e_s, e_{s+1})$ and $p(e_t, e_{t+1})$, we can compute $f(\mathcal{H}_i)$, the objective function value of the tour $\mathcal{H}_i = \Delta - e_i + \{(u, i), (u, i + 1)\}$ for MBTSP on W , in $O(1)$ time by considering 10 different cases [668].

Choose q such that $f(\mathcal{H}_q) = \min\{f(\mathcal{H}_i) \mid 1 \leq i \leq n\}$. Then the tour \mathcal{H}_q is an optimal solution to MBTSP on W . Now

$$e^*, e^{**}, p(e_r, e_{r+1}), p(e_s, e_{s+1}), \text{ and } p(e_t, e_{t+1})$$

can be identified in $O(n)$ time. The quantity δ_i can be identified in $O(1)$ time for each $i = 1, 2, \dots, n$. Thus, MBTSP on W can be solved in $O(n)$ time.

4.6.2 MBTSP on a Halin Graph. We now consider MBTSP on a Halin graph $G = (N, E)$ which is not a wheel. Note that such a graph has at least two fans. Let S be a fan of G with center u and outer nodes u_1, u_2, \dots, u_m , where (u_1, u_2, \dots, u_m) is a path in S . The cutset $\{S, N - S\}$ contains precisely three edges $e_{S,1}, e_{S,2}$, and $e_{S,3}$ incident with nodes u_1, u , and u_m , respectively.

Any tour \mathcal{H} in G uses exactly two of the three edges $e_{S,1}, e_{S,2}$, and $e_{S,3}$. If \mathcal{H} uses $e_{S,1}$ and $e_{S,3}$, then $\mathcal{H} \cap S$ has the structure

$$(u_1, u_2, \dots, u_j, u, u_{j+1}, \dots, u_m) \tag{9}$$

If \mathcal{H} uses $e_{S,1}$ and $e_{S,2}$, then $\mathcal{H} \cap S$ is of the form

$$(u, u_m, u_{m-1}, \dots, u_2, u_1) \tag{10}$$

and if \mathcal{H} uses $e_{S,2}$ and $e_{S,3}$, then $\mathcal{H} \cap S$ will be of the form

$$(u_m, u_{m-1}, \dots, u_2, u_1, u) \tag{11}$$

Let $g(e_{S,2}, e_{S,3})$ be the contribution of S to the MBTSP objective function value of an optimal tour in G using edges $e_{S,2}$ and $e_{S,3}$. Then

$$g(e_{S,2}, e_{S,3}) = \max\{x_1, y_1\}$$

where

$$x_1 = \max\{\{c_{u_i, u_{i+1}} \mid 1 \leq i \leq m - 1\}, c_{u, u_1}\}$$

and

$$y_1 = \max\{\{p((u_i, u_{i+1}), (u_{i+1}, u_{i+2})) \mid 1 \leq i \leq m - 2\}, p((u_{m-1}, u_m), e_{S,3}), p((u_1, u_2), (u, u_1)), p((u, u_1), e_{S,2})\}.$$

Similarly, let $g(e_{S,1}, e_{S,2})$ be the contribution of S to the objective function value of an optimal tour in G using edges $e_{S,1}$ and $e_{S,2}$. Then

$$g(e_{S,1}, e_{S,2}) = \max\{x_2, y_2\}$$

where,

$$x_2 = \max\{\{c_{u_i, u_{i+1}} \mid 1 \leq i \leq m - 1\}, c_{u, u_m}\}$$

and

$$y_2 = \max\{\{p((u_i, u_{i+1}), (u_{i+1}, u_{i+2})), 1 \leq i \leq m - 2\}, p((u_{m-1}, u_m), (u_m, u)), p((u_1, u_2), e_{S,1}), p((u, u_m), e_{S,2})\}.$$

We similarly define $g(e_{S,1}, e_{S,3})$, the contribution of S to the objective function value of an optimal tour in G using edges $e_{S,1}$ and $e_{S,3}$. The quantities $g(e_{S,1}, e_{S,2})$, $g(e_{S,1}, e_{S,3})$, and $g(e_{S,2}, e_{S,3})$ can be computed in $O(p)$ time by appropriate modifications of the procedure for solving MBTSP on a wheel.

Consider the Halin graph $G \times S$ obtained from G by collapsing S into a pseudo-node \hat{v} . Let $e_{\hat{v},1}, e_{\hat{v},2}, e_{\hat{v},3}$ be the edges in $G \times S$ incident on node \hat{v} which correspond to edges $e_{S,1}, e_{S,2}, e_{S,3}$, respectively.

Assign to the penalty triplet $[p(e_{\hat{v},1}, e_{\hat{v},2}), p(e_{\hat{v},1}, e_{\hat{v},3}), p(e_{\hat{v},2}, e_{\hat{v},3})]$ the values $[g(e_{S,1}, e_{S,2}), g(e_{S,1}, e_{S,3}), g(e_{S,2}, e_{S,3})]$, respectively. Further, let $c_{e_{\hat{v},1}} = c_{e_{S,1}}, c_{e_{\hat{v},2}} = c_{e_{S,2}},$ and $c_{e_{\hat{v},3}} = c_{e_{S,3}}.$

The optimal objective function value of MBTSP on G and the optimal objective function value of the corresponding instance of MBTSP on $G \times S$ are the same. Note that $G \times S$ is also a Halin graph. If it is a wheel then MBTSP on the wheel can be solved using the algorithm discussed earlier. If it is not a wheel, then it contains at least two fans and the fan collapsing scheme can be continued and eventually we reach a wheel. Backtracking from the optimal solution on this wheel, an optimal solution to MBTSP (and hence to BTSP) on G can be constructed by expanding the pseudo-nodes. We leave it to the reader to verify that the procedure can be implemented in $O(n)$ time.

5. Variations of the Bottleneck TSP

Let us now discuss some variations of the BTSP that subsume both TSP and BTSP. We first consider traveling salesman problem under categorization [678]. Let S_1, S_2, \dots, S_p be a partition of the edge set E of the graph (digraph) $G = (N, E)$ and let \mathbb{F} be the family of all Hamiltonian cycles in G . Let c_e be the cost of edge $e \in E$. For any $\mathcal{H} \in \mathbb{F}$, define

$$f_1(\mathcal{H}, C) = \max_{1 \leq i \leq p} \sum_{e \in \mathcal{H} \cap S_i} c_e$$

and

$$f_2(\mathcal{H}, C) = \sum_1^p \max_{e \in \mathcal{H} \cap S_i} c_e$$

where summation over the empty set yields a value $-\infty$ and maximum over the empty set yields zero.

There are two versions of the traveling salesman problem under categorization which are denoted by TSPC-1 and TSPC-2. TSPC-1 seeks a Hamiltonian cycle \mathcal{H}' in G such that $f_1(\mathcal{H}', C)$ is as small as possible, whereas TSPC-2 attempts find a Hamiltonian cycle of G that minimizes $f_2(\mathcal{H}, C)$. For $p = 1$, TSPC-1 reduces to the TSP and TSPC-2 reduces to the BTSP. For $p = |E|$, TSPC-1 reduces to the BTSP and TSPC-2 reduces to the TSP. Thus TSPC-1 and TSPC-2 are proper generalizations of both TSP and BTSP and hence are NP-hard. We have seen that BTSP and TSP can be solved in linear time on Halin graph. However, it is possible to show that both TSPC-1 and TSPC-2 are strongly NP-hard on Halin graphs [678]. However, for fixed p , TSPC-2 is solvable in polynomial time on a Halin graph whereas TSPC-1 remains NP-hard

on Halin graphs even for $p = 2$ [678]. Similar generalizations of MAX TSP and MSTSP can be constructed.

Suppose that a Hamiltonian cycle represents the order in which n jobs are to be processed sequentially in an assembly line. Then the objective function of the TSP measures the total processing time. Under this model, each of TSPC-1 and TSPC-2 can be interpreted as the problem of minimizing the completion time of jobs in an assembly line when there is some series-parallel order relation for processing the jobs. More precisely, if jobs corresponding to the same subset are to be processed sequentially, where as different subsets can be processed in parallel, then the TSPC-1 objective function measures the total processing time. If jobs corresponding to the same subsets can be done in parallel, but a new subset of jobs can be taken only after processing all jobs from the current set, TSPC-2 objective function measures the total processing time.

Another problem that simultaneously generalizes BTSP and TSP is the k -sum traveling salesman problem (k -sum TSP) [411, 681, 802]. In a k -sum TSP, one seeks a Hamiltonian cycle \mathcal{H} of a graph (digraph) G on n nodes where the sum of the k largest edge-weights of \mathcal{H} is as small as possible. For $k = n$, k -sum TSP reduces to the TSP and for $k = 1$ it reduces to the BTSP. The k -sum TSP can be solved by solving $O(m)$ TSP's where m is the number of edges in G .

Acknowledgements: We are thankful to P. Sharma and P. Toth for their comments on an earlier version of this chapter. This work was partially supported by the NSERC grant OPG0008085 awarded to S.N. Kabadi and the NSERC grant OPG0170381 awarded to A.P. Punnen.

Chapter 16

TSP SOFTWARE

Andrea Lodi

D.E.I.S, University of Bologna

Viale Risorgimento 2, 40136 Bologna, Italy

alodi@deis.unibo.it

Abraham P. Punnen

Department of Mathematical Sciences

University of New Brunswick-Saint John

New Brunswick, Canada

punnen@unbsj.ca

1. Introduction

In the preceding chapters we have seen various algorithms for solving the *Traveling Salesman Problem* (TSP) and other related problems either by computing a provably optimal solution or by computing an approximate (heuristic) solution. Results of extensive comparative studies of various competitive heuristic algorithms for the symmetric and asymmetric traveling salesman problems are presented in Chapters 9 and 10. Most often these comparative studies focus on two important criteria - the quality of the solution produced and the computational time. While computational performance is one of the most important criteria, several other desirable criteria should to be considered in evaluating a software for any computational problem. An academician may be interested in a software with the aim to illustrate various features of an algorithm and a practitioner may want to solve quickly a small problem without bothering to work with a large and sophisticated source code. Moreover, due to the increasing use of *Internet* and *World Wide Web* in the academia, *Java applets* illustrating various steps of an algorithm have pedagogical advantages. Even in the case of source codes, one may prefer a specific programming language to another, whether it is a general-purpose language such as FORTRAN, Pascal, C, C++, etc. or

it is a special-purpose language such as AMPL, MAPLE, Mathematica, etc. Availability of programs in the form of a “callable library” will be of interest to many users. One may also be interested in a software with user friendly input/output formats or even a Graphical User Interface (GUI). Thus, our review of TSP software takes into account a broad range of characteristics, in addition to computational performance. We are not attempting to provide a comparative study of all available TSP software. Instead, our aim is to collect information available in various contexts and present them in an organized framework so that the readers of this chapter will have the most up-to-date information on available TSP software.

For most of the software that is available through the web there is a high probability that the current URL may become obsolete over a period of time and some times it may become difficult trace new URL's, if any, for the information. In order to minimize such difficulties we maintain the web page

http://www.or.deis.unibo.it/research_pages/tspsoft.html

which is mirrored at

<http://or.unbsj.ca/~punnen/tspsoft.html>

where latest information on software discussed in this chapter can be accessed in addition to information on updates and on new software.

Problem Transformations: The *Asymmetric* Traveling Salesman Problem (ATSP), i.e., a TSP in which the cost matrix is not necessarily symmetric, is clearly more general than the *Symmetric* Traveling Salesman Problem (STSP) where the cost matrix is always assumed to be symmetric. Often these two versions of the TSP are investigated independently.

On one hand, a code for the ATSP can in principle handle symmetric instances, but the effectiveness of many of the solution procedures for the ATSP depends on the asymmetric structure of the cost matrix (e.g., methods based on the Assignment Problem (AP) relaxation) and these procedures become less powerful in the symmetric case. On the other hand, substantial effort has been invested in developing efficient algorithms for solving the STSP. As a result very effective solution procedures and computer codes are available to solve the STSP. Chapter 1 and the computational section of Chapter 4 discuss transformations that reduce an ATSP into a STSP by doubling the number of nodes. Thus,

any code for the STSP can be used to solve the ATSP using these transformations. Further, Chapter 1 contains transformations that reduce several variations of TSP to the TSP. Thus, under these transformations, the TSP software discussed here can also be used to solve these variations. Our discussion also includes information on some general-purpose codes that can be used in developing software for TSP and other combinatorial optimization problems.

2. Exact algorithms for TSP

In this section we consider various software to compute an optimal solution to the TSP.

- 1 **Concorde**: This is an ANSI C code for the exact solution of STSP and is available free of charge for academic research use. The code implements a branch-and-cut algorithm for the STSP developed by Applegate, Bixby, Chvátal and Cook [29] (see Chapter 2). It also provides a number of additional interesting features such as implementations of (i) heuristic algorithms (see Section 3 and Chapter 9), (ii) general algorithms for network optimization (e.g., algorithms for Minimum Cut computation, see Section 6), and (iii) a solver for the TSP variant called *multiple* TSP.

The current release 12/15/1999 is well documented and can be obtained in `tar.gz` format (“co991215.tgz”) from the web site:

<http://www.math.princeton.edu/tsp/concorde.html>.

- 2 **CDT**: This is a FORTRAN 77 code to compute an exact solution to the ATSP. It is published as code 750 in *ACM Transactions on Mathematical Software*. The code implements the branch-and-bound AP-based algorithm by Carpaneto, Dell’Amico and Toth [164] (see Chapter 4 for details and computational results). The release also includes an implementation of the well known patching heuristic by Karp (see again Chapter 4 for details).

The code can be obtained in `gz` format (“750.gz”) from the web site:

<http://www.acm.org/calgo/contents/>.

- 3 **TSP1 (TSP2)**: These are Turbo Pascal codes for solving the STSP as developed by Volgenant and van den Hout in the ORSEP context [427]. Code TSP2 is menu driven with graphical options and implements a modified version of TSP1. The codes implement the ‘1-tree’-based branch-and-bound algorithm by Jonker and Volgenant [470]. (The release includes an implementation of Christofides heuristic [189] improved with some restricted 3-opt moves.)

The code can be obtained in `zip` format (“volgenan.zip”) from the web site:

<http://www.mathematik.uni-kl.de/~wwwwi/WWWI/ORSEP/contents.html>.

- 4 `tsp_solve`: This is a C++ code for the exact solution of both ATSP and STSP. The release incorporates a number of algorithms: exact branch-and-bound algorithms based on the AP, 1-tree, and arborescence relaxations, and heuristic algorithms (see Section 3). Developed by Hurwitz and Craig, the package comes with good documentation and is available free of charge.

The release 1.3.6 can be obtained in `tar.gz` format (“tsp_solve-1.3.6.tar.gz”) from the web site:

<http://www.cs.sunysb.edu/~algorithm/implement/tsp/implement.shtml>

A new release 1.3.7 is also available. For details contact the authors using the e-mail address `churitz@cts.com`.

- 5 `SYMPHONY`: This is a general-purpose parallel branch-and-cut code for integer and mixed-integer programming (mainly over networks), written in ANSI C programming language by Ralphs. Developed for solving vehicle routing problems, the recent release 2.8 contains a (parallel) STSP solver which exploits some of the separation routines of *Concorde*.

The code can be obtained, free of charge, by filling an appropriate form at the web site:

<http://www.branchandcut.org/>.

- 6 `tsp*`: This is an AMPL code to compute an exact solution of the STSP, as developed by Lee. This code is part of a didactic web site containing computational tools for combinatorial optimization problems including an AMPL implementation of the Christofides heuristic [189], and some codes for related problems such as minimum cut computation. (The package includes codes for finding: the minimum weight Euclidean spanning tree T , the minimum weight perfect matching M of odd degree nodes of T , and an Eulerian tour in $T \cup M$.) Finally, the package also includes some utilities for viewing the solutions of Euclidean (2-D) problems with *Mathematica*.

The code can be obtained from the web site:

<http://www.ms.uky.edu/~jlee/jlsup/jlsup.html>.

- 7 `Combinatorica`: This is an add-on to the package *Mathematica* developed by Skiena [763]. It includes the capability of solving the STSP and the Hamiltonian cycle problem. For the STSP,

Combinatorica can be used to compute upper and lower bounds, and the optimal solution.

The package is included in the standard distribution of *Mathematica* (directory `Packages/DiscreteMath/Combinatorica`). It can also be obtained from the ftp site:

`ftp://ftp.cs.sunysb.edu/pub/Combinatorica/`.

- 8 **rank**: This is a software for ranking the solutions of the assignment problem by non-increasing values of the objective function. (Thus, this software could be used for solving the ATSP by testing feasibility of each assignment.) It is an executable file for DOS platform, developed by Metrick and Maybee in the ORSEP context [427] and implements the algorithm by Murty [610].

The software can be obtained in zip format (“murty.zip”) from the web site:

`http://www.mathematik.uni-kl.de/~wwwwi/WWWI/ORSEP/contents.html`.

- 9 **babtsp**: This is a Pascal code implementing the branch-and-bound algorithm by Little, Murty, Sweeney, and Karel [565], and developed by Syslo for the book by Syslo, Deo and Kowalik [779].

The code can be obtained in zip format (“syslo.zip”) from the web site:

`http://www.mathematik.uni-kl.de/~wwwwi/WWWI/ORSEP/contents.html`.

3. Approximation Algorithms for TSP

In this section we consider various software for the heuristic solution of STSP and ATSP.

- 1 **LKH**: This is an ANSI C code to compute an approximate solution of STSP. It implements a variation of the well known Lin-Kernighan heuristic [563] developed by Helsgaun [446] (see Chapter 9). The code is available free of cost for academic research.

The current release 1.0 accepts input data in any of the TSPLIB [709] formats, and the output format can be controlled by appropriate parameter settings. The code can be obtained either in tar.gz format (“LKH-1.0.tgz”) or in a stuffit archive format (“LKH-1.0.sit”) from the web site:

`http://www.dat.ruc.dk/~keld/`.

- 2 **SaCEC**: This is a software package for the approximate solution of the STSP. It implements the “Stem-and-Cycle Ejection Chain Algorithm” by Rego, Glover and Gamboa developed in the context

of the 8th DIMACS Implementation Challenge devoted to the TSP, the results of which are summarized in Chapter 9.

The software is available for Microsoft Windows, Linux and Unix platforms at the web site:

<http://faculty.bus.olemiss.edu/crego/>.

- 3 **Concorde**: As mentioned in the previous section, the **Concorde** package includes effective ANSI C implementations of the following approximation algorithms for the STSP:

- (a) Chained Lin-Kernighan heuristic [588],
- (b) k -opt heuristic with $k = 2, 2.5, 3$,
- (c) Greedy, Nearest Neighbor, Boruvka, and Farthest addition.

An ANSI C implementation of the 1-tree relaxation algorithm by Held and Karp [444, 445] is also included to compute a lower bound on the optimal objective function value (see Chapter 9 for experimental results).

- 4 **DynOpt**: This is an ANSI C implementation a dynamic programming based algorithm developed by Balas and Simonetti [80]. It computes approximate solutions of both STSP and ATSP. Some flexibility in the input format is guaranteed by allowing the user to implement a macro (“theName(a,b)” in the code) which determines how to calculate the distances among cities.

The code can be obtained in **ascii** format from the web site:

<http://www.andrew.cmu.edu/~neils/tsp/>.

- 5 **LK**: This is an ANSI C code for computing an approximate solution of the STSP. The code implements another effective variation [626] of the Lin-Kernighan heuristic [563], and it is available free of cost under the GNU Library General Public License. The code is developed using the “Cweb” tool set [509], and to read it “Cweb” and “LaTeX” [391] are required. In addition, the (non-standard) BSD resource usage functions (“getrusage”) are required to run the program. The code uses the TSPLIB [709] input formats allowing various output options.

The current release 0.5.0 can be obtained in **tar.gz** format (“lk-0.5.0.tar.gz”) from the web site:

<http://www.cs.utoronto.ca/~neto/research/lk>.

- 6 **TSP**: This is a FORTRAN code for computing an approximate solution of the STSP. The code implements the well known Christofides heuristic [189], and is published in the book by Lau [544].

- 7 **Routing**: This is a FORTRAN code for computing an approximate solution of the ATSP and published as code 456 on *Communications of the ACM*. The code implements the node-insertion plus 3-opt heuristic by Fencel [288].

The code, to the best of our knowledge, is not available on-line but can be obtained in pdf format (inside the article) at the web address:

<http://www.netlib.org/tomspdf/456.pdf>.

- 8 **twoopt, threeopt, fitsp**: These are Pascal codes for computing a heuristic solution of the symmetric TSP and published in the book by Syslo, Deo and Kowalik [779]. These codes implement the two-opt, three-opt and farthest insertion heuristics, respectively.

The codes can be obtained in zip format (“syslo.zip”) at the web site:

<http://www.mathematik.uni-kl.de/~wwwwi/WWWI/ORSEP/contents.html>.

- 9 **GATSS**: This is a GNU C++ code for computing a heuristic solution of the STSP and is linked to a user interface in HTML via CGI-script. The code implements a standard genetic algorithm, and is suitable for didactic purposes.

The code can be obtained in ascii format at the web site:

<http://www.acc.umu.se/~top/travel.information.html>.

- 10 **Tours**: This is a software package for computing approximate solutions of both STSP and ATSP. The software is able to compute several upper and lower bounds for the problems using various algorithms from the literature. In addition, a branch-and-bound code can be used to solve small instances to optimality. The software has graphical facilities, and has options of reading/saving both ascii and binary files. User’s manual and on-line help are also included.

The software is commercially available for Microsoft Windows platforms, and information on how to buy it can be found from the web site:

<http://www.logisticsCAD.com>.

- 11 **RAI**: This is an ANSI C code for the approximate solution of an ATSP and uses the “flex/lex” tool. The code implements the algorithm by Brest and Zerovnik [137] based on a sequence of randomized insertions.

The code can be obtained (directly as a source code) at the web site:

<http://marcel.uni-mb.si/~janez/rai>.

- 12 **ECTSP**: This is a software package containing two different heuristics for solving the STSP. The first heuristic is based on a genetic algorithm while the second is an evolutionary programming heuristic. The package contains two executable files for the Windows 9x platforms, can be used free of charge, and comes with graphical interfaces.

The software has been developed by Ozdemir and Embrechts, and can be obtained from the authors (using the two e-mail addresses {ozdemm, embrem}@rpi.edu).

- 13 **GlsTsp**: This is a C++ code for the heuristic solution of the STSP. The code implements the guided local search approach by Voudouris and Tsang [815]. The current release 2.0 includes an executable file for Microsoft Windows platforms, and the source code in C++ for Unix/Linux platforms.

The release can be obtained free of charge in zip format (from "glstsp.zip") from the web site:

http://www.labs.bt.com/people/voudouc/index_download.htm.

- 14 **TSPGA**: This is an ANSI C code for the approximate solution of the STSP using the "pgapack" package. The code implements the approach of Frick [323] which combines evolutionary computation and local search.

The code can be obtained from the author (using the e-mail address afr@aifd.uni-karlsruhe.de), and additional information can be found at the web site:

<http://www.rz.uni-karlsruhe.de/~ul63>.

- 15 **Operations_Research_2.0**: This is a *Mathematica*-based software for solving operations research problems. It contains heuristic algorithms for the TSP, based on simulated annealing and ant colony systems. It also contains a specialized branch-and-bound algorithm. *Mathematica* version 3 or up is required to use this software.

The software can be purchased from SoftAS GmbH and information on how to buy it can be found at the web site:

<http://www.softas.de/products.html>

4. Java Applets

Java applets are useful in demonstrating how an algorithm works, especially over the web by providing animations. They can also be used for solving small scale problems. The web sites given below contain some Java applets for well known TSP heuristics. In our view, most of these codes appear to be preliminary versions and much work needs to be done in order to bring them to a level where they can be used for any serious applications.

- 1 <http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/java/tsp.html>.
- 2 <http://home.planet.nl/~onno.waalewijn/tsp.html>.
- 3 <http://www.wiwi.uni-frankfurt.de/~stockhei/touropt.html>.
- 4 <http://itp.nat.uni-magdeburg.de/~mertens/TSP/index.html>.

5. Variations of the TSP

This section is devoted to the discussion of software for some important variations of the TSP.

- 1 **concorde**: As mentioned in Section 2, a linear programming based code for the TSP variant called *Multiple TSP* is provided within the **concorde** distribution.
- 2 **RA-TSP**: A variant of the ATSP called *Arc Replenishment Traveling Salesman Problem* has been studied by Mak and Boland [577], and their ANSI C codes for this problem and its variations are available for research purposes. In particular, the codes consist of a simulated annealing to obtain heuristic solutions, a Lagrangian relaxation for computing lower bounds, and a branch-and-bound algorithm. Both Lagrangian relaxation and branch-and-bound codes use CPLEX 6.0.

Information on problems and codes can be obtained from the web site:

<http://www.ms.unimelb.edu.au/~vmak>

- 3 **Salazar's_codes**: Many interesting variants of the TSP has been considered by Salazar and co-authors. These include the *Capacitated Vehicle Routing Problem*, the *Symmetric Generalized TSP*, the *Orienteering Problem*, the *Traveling Purchaser Problem*, and the *Pickup-and-Delivery TSP* (see details on some of these problems in Chapter 13). For all these variations Salazar developed ANSI C codes which can be shared for research purposes.

Information on problems and codes can be obtained from the web site:

<http://webpages.ull.es/users/jjsalaza/>.

- 4 **Ascheuer's_codes**: Interesting variations of the ATSP such as *ATSP with Precedence Constraints* and the *ATSP with Time Windows* have been studied by Ascheuer and co-authors. Ascheuer developed ANSI C codes for solving these problems.

Information on these codes can be obtained from the web site:

<http://www.zib.de/ascheuer>.

- 5 **HC**: This is a FORTRAN 77 code to compute one or more Hamiltonian circuits in a directed graph and published as code 595 on *ACM Transactions on Mathematical Software*. It implements the algorithm by Martello [585].

The code can be obtained in **gz** format ("595.gz") from the web site:

<http://www.acm.org/calgo/contents/>.

- 6 **hamcycle**: This is a computer code which implements several algorithms to compute a Hamiltonian cycle in a graph, also including random graph generators. The codes, developed by Vandegriend and Culberson, are written in ANSI C programming language. The release includes a user's manual, and it is available free of charge.

The code can be obtained in **tar.gz** format ("prog.tar.gz") at the web site:

<http://web.cs.ualberta.ca/~joe/Theses/Hcarchive/main.html>.

- 7 **Groups&Graphs**: This package contains computer codes to solve several graph theoretical problems including the Hamiltonian cycle problem. The codes, developed by Kocay, has two different releases: 3.0 for Macintosh platforms and 0.9 for Microsoft Windows platforms. Both are available free of charge.

These releases can be obtained from the web site:

<http://kohlrabi.cs.umanitoba.ca/G&G/G&G.html>.

- 8 **Ariadne100**: This is a software package for computing Hamiltonian cycles in a graph. Various experimental modes are also available with random graph generators. The software package is developed for Microsoft Windows platforms and is available free of charge.

The current release 8.23.2000 can be obtained in `exe` format (“SetupAriadne.exe”) at the web site:

<http://www.geocities.com/nasukun/Ariadne/Ariadne.html>.

- 9 `hamcrc`: This is a FORTRAN code for finding the k^{th} vertex in a Hamiltonian cycle. By repeatedly calling this subroutine, a Hamiltonian cycle in a graph can be identified. The code is included in the book by Nijenhuis and Wilf [628].

The code “hamcrc.f” is available as part of the distribution file “other.tar.gz” (which contains all the codes of the book) at the web site:

<http://www.cs.sunysb.edu/~algorithm/implement/wilf/distrib/>.

- 10 `LEP-CR`: This is a LEDA-Extension Package to solve the *Curve Reconstruction Problem*. This problem is not exactly a variation of the TSP, but is closely related to the TSP. A specific TSP-based code to solve the Curve Reconstruction has been developed by Althaus.

The package in `tgz` format (“LEP_curve_reconstruction_1.0.tgz”) can be obtained at the web site:

<http://www.mpi-sb.mpg.de/~althaus/LEP:Curve-Reconstruction/>.

6. Other Related Problems and General-Purpose Codes

- 1 `LP`: Efficiently solving linear programming relaxations is crucial for state of the art exact TSP solvers. Since a review of (commercial) software for LP is out of the scope of this chapter, we refer to the Software Survey by Fourer published in *OR/MS Today* in 1999 [318].

This survey is available on line at the web site:

<http://lionhrtpub.com/orms/orms-8-99/survey.html>.

Almost all the presented packages are still available, often in more recent releases, thus the survey is still useful for giving the related pointers to the interested reader. (We are also confident that the survey will be updated soon.)

- 2 `AP`: As mentioned in the introduction, algorithms for the ATSP often require to solve the Assignment Problem as a relaxation. Several algorithms and codes for AP have been developed in the literature and are available on the web. We refer to the recent survey by Dell’Amico and Toth [249] for a thorough treatment of the subject.

- 3 **mincut**: Many of the LP-based methods for TSP need to compute a *Minimum Cut* in a graph in order to separate Subtour Elimination Constraints (see Chapter 2 for details). In recent years, several studies have been devoted to efficient algorithms and codes to solve this problem. We refer to the paper by Jünger, Rinaldi and Thienel [476] and to their library of codes which can be obtained at the web site:

<http://www.informatik.uni-koeln.de/lj-juenger/projects/mincut.html>.

Let us now consider some general-purpose software and libraries for developing TSP applications.

- 1 **COIN-OR**: The name COIN-OR stands for “Common Optimization Interface for Operations Research”. According the COIN-OR web page: “Our goal is to create for mathematical software what the open literature is for mathematical theory”.

Detailed information can be found at the web site:

<http://oss.software.ibm.com/developerworks/opensource/coin/>.

- 2 **BOB (BOB++)**: This is a general-purpose software library to implement enumerative algorithms which exploit parallelism. TSP is used, among other combinatorial problems, to illustrate the use of the library.

The current release 1.0 is available free of cost at the web site:

http://www.prism.uvsq.fr/optimize/index_us.html.

- 3 **ABACUS**: This is a callable C++ library designed to provide a general framework for the implementation of enumerative algorithms. It supports cutting plane generation and/or column generation, and the linear programming solvers CPLEX and XPRESS.

The package can be purchased from OREAS, and information can be found at the web site:

<http://www.oreas.de>.

- 4 **MINTO**: This is a general-purpose software to solve mixed integer programs by using linear programming relaxations and branch-and-cut. The system allows the user to consider specific problems e.g. by adding problem-specific cutting planes and defining appropriate branching strategies. The current version supports CPLEX and OSL.

Detailed information can be found at the web site:

<http://akula.isye.gatech.edu/~mwps/projects/minto.html>.

5 CP: In the last decade, *Constraint Programming* (CP) [584] has shown its effectiveness in modeling and solving real-world combinatorial optimization problems. CP is a programming paradigm exploiting Constraint Satisfaction techniques, and many CP tools have been developed to tackle discrete (optimization) problems. Although CP is currently far from being competitive for solving “pure” problems like TSP, many TSP-like applications involving side constraints have been successfully solved through CP algorithms (such as the TSP with Time Windows, see, e.g., Focacci, Lodi, Milano [315]).

An exhaustive list of CP tools would be too long and is obviously outside the scope of this chapter. However, useful pointers along with the C++ source code of the TSP constraint (in zip format, “cost-based.zip”) used in [315] are available at the web site:

http://www.or.deis.unibo.it/research_pages/ORcodes/CP.html.

6 parSA-Lib: This is a general-purpose software library providing a general framework for implementing simulated annealing algorithms in parallel. The library is written in C++ programming language.

Available by filling up a request form at the web site:

<http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PARSA/>

7 PPBB-Lib: This is a general-purpose software library that can be used to parallelize sequential branch-and-bound algorithms. The library is written in ANSI C programming language.

Detailed information can be found at the web site:

<http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PPBB/>

Acknowledgments

The authors wish to thank Matteo Fischetti, David S. Johnson, and Paolo Toth for reading preliminary versions of the chapter and pointing out missing entries. Thanks are also due to the authors of the software cited in this chapter for sending the required material. The work of Abraham Punnen was partially supported by the NSERC grant OPG0170381.

Appendix: A. Sets, Graphs and Permutations

Gregory Gutin

*Department of Computer Science, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK*

G.Gutin@rhul.ac.uk

In this appendix we will provide certain terminology, notation and results on sets, graphs and permutations frequently used in various chapters of this book. In this book we consider both directed and undirected graphs (often called digraphs and graphs, respectively) and mostly follow standard terminology, see, e.g., [84, 126, 130, 254].

1. Sets

For the sets of real numbers, rational numbers and integers we will use \mathbb{R} , \mathbb{Q} and \mathbb{Z} , respectively. Also, let $\mathbb{Z}_+ = \{z \in \mathbb{Z} : z > 0\}$ and $\mathbb{Z}_0 = \{z \in \mathbb{Z} : z \geq 0\}$. The sets \mathbb{R}_+ , \mathbb{R}_0 , \mathbb{Q}_+ and \mathbb{Q}_0 can be defined similarly.

We assume that the reader is familiar with the following basic operations for a pair A, B of sets: the *intersection* $A \cap B$, the *union* $A \cup B$, and the *difference* $A \setminus B$ (also denoted by $A - B$). Sets A and B are *disjoint* if $A \cap B = \emptyset$. The *Cartesian product* of sets X_1, X_2, \dots, X_p is $X_1 \times X_2 \times \dots \times X_p = \{(x_1, x_2, \dots, x_p) : x_i \in X_i, 1 \leq i \leq p\}$.

For sets A, B , $A \subseteq B$ means that A is a subset of B ; $A \subset B$ stands for $A \subseteq B$ and $A \neq B$. A collection S_1, S_2, \dots, S_t of (not necessarily non-empty) subsets of a set S is a *partition* of S if $S_i \cap S_j = \emptyset$ for all $1 \leq i \neq j \leq t$ and $\cup_{i=1}^t S_i = S$. A family $\mathcal{F} = \{X_1, X_2, \dots, X_n\}$ of sets is *covered* by a set S if $S \cap X_i \neq \emptyset$ for every $i = 1, 2, \dots, n$. We say that S is a *cover* of \mathcal{F} . For a finite set X , the number of elements in X (i.e. its *cardinality*) is denoted by $|X|$. We also say that X is an $|X|$ -*element set*.

2. Graphs

A *directed graph* (or just *digraph*) D consists of a non-empty finite set $V(D)$ of elements called *vertices* or *nodes* and a finite set $A(D)$ of ordered pairs of distinct vertices called *arcs*. We call $V(D)$ the *vertex set* and $A(D)$ the *arc set* of D . We also write $D = (V, A)$ which means that V and A are the vertex set and arc set of D , respectively. The first (second) vertex of an arc is its *tail* (*head*). An arc a with tail x and head y is written as xy or (x, y) ; we say that x is an *in-neighbor* of y and y is an *out-neighbor* of x , and x and y are *adjacent*.

An *undirected graph* (or just *graph*) G consists of a non-empty finite set $V(G)$ of elements called *vertices* or *nodes* and a finite set $E(G)$ of unordered pairs of distinct vertices called *edges*. We call $V(G)$ the *vertex set* and $E(G)$ the *edge set* of G . We also write $G = (V, E)$ if $V = V(G)$ and $E = E(G)$. The two vertices of an edge are its *end-vertices*. An edge e with end-vertices x and y is written as xy or (x, y) ; we

say that x and y are *adjacent*, or *neighbors*. The set of neighbors of x is denoted by $N_G(x)$.

A directed or undirected graph H is *complete* if every pair of distinct vertices of H are adjacent. A complete undirected (directed) graph on n vertices is denoted by K_n (\vec{K}_n). A set X of vertices in directed or undirected graph is *independent* (or *stable*) if no pair of vertices in X are adjacent. An undirected graph G is *complete multipartite* if $V(G)$ can be partitioned into a number of subsets, called *partite sets*, such that every partite set is independent and every pair of vertices from different partite sets are adjacent. A complete multipartite graph is denoted by $K(n_1, n_2, \dots, n_p)$, where n_i is the number of vertices in the i th partite set; $K(n_1, n_2, \dots, n_p)$ is also called *complete p -partite*, and *complete bipartite*, if $p = 2$.

A digraph H is a *subgraph* of a digraph D if $V(H) \subseteq V(D)$, $A(H) \subseteq A(D)$ and every arc in $A(H)$ has both end-vertices in $V(H)$. If $V(H) = V(D)$, we say that H is a *spanning subgraph* (or a *factor*) of D . If every arc of $A(D)$ with both end-vertices in $V(H)$ is in $A(H)$, we say that H is *induced* by $X = V(H)$ (we write $H = D[X]$) and call H an induced subgraph of D . Similarly, one can define *subgraphs*, *spanning* and *induced subgraphs* of undirected graphs.

The *degree* $d(x)$ of a vertex x in a graph G is the number of neighbors of x . The *out-degree* $d^+(x)$ (*in-degree* $d^-(x)$) of a vertex x of a digraph D is the number of out-neighbors (in-neighbors) of x . A directed (undirected) graph H is *k -regular* if the out-degree and in-degree (degree) of every vertex of H is equal to k . A *k -regular spanning subgraph* of a graph (digraph) H is called a *k -factor*; a *k -matching* is a subgraph of a k -factor; a *matching* is a 1-matching; a *perfect matching* is a 1-factor (the last two terms are mostly used for undirected graphs). For a digraph D , a 1-matching with $|V(D)| - 1$ arcs is an *almost 1-factor*. For any digraph $G = (V, A)$ and any $S \subseteq V$, $\delta^+(S) = \{(i, j) : i \in S, j \in V - S\}$, $\delta^-(S) = \delta^+(V - S)$ and $\delta(S) = \delta^+(S) \cup \delta^-(S)$.

A *chain* in a digraph D is an alternating sequence

$$W = x_1 a_1 x_2 a_2 x_3 \dots x_{k-1} a_{k-1} x_k$$

of vertices x_i and arcs a_j from D such that, for every $i = 1, 2, \dots, k - 1$, either $a_i = (x_i, x_{i+1})$ or $a_i = (x_{i+1}, x_i)$. If $a_i = (x_i, x_{i+1})$ for every $i = 1, 2, \dots, k - 1$, then W is a *walk*. A walk W is *closed* if $x_1 = x_k$, and *open*, otherwise. The set of vertices $\{x_1, x_2, \dots, x_k\}$ is denoted by $V(W)$; the set of arcs $\{a_1, a_2, \dots, a_{k-1}\}$ is denoted by $A(W)$. We say that W is a walk from x_1 to x_k or an (x_1, x_k) -*walk* and is of *length* $k - 1$. A *trail* is a walk with distinct arcs; a *path* is an open walk with distinct vertices; a *cycle* is a closed walk in which all vertices, but first and last, are distinct. We will often identify a walk W with the digraph $(V(W), A(W))$. In particular, we may say that a 1-factor is a collection of cycles. *Walks*, *trails*, *paths* and *cycles* in undirected graphs can be defined similarly. Since paths and cycles are well-defined by the corresponding sequences of their vertices, we denote them using these vertex sequences, e.g., $x_1 x_2 \dots x_k$ and $x_1 x_2 \dots x_k x_1$ are a path and a cycle (provided all vertices are distinct). We sometimes use the "brackets" notation: (x_1, x_2, \dots, x_k) and $(x_1, x_2, \dots, x_k, x_1)$.

A path or cycle W in directed or undirected graph G is a *Hamilton* (or *Hamiltonian*) path or cycle if $V(W) = V(G)$; G is *Hamiltonian* (*Hamiltonian connected*) if it contains a Hamilton cycle (a Hamilton path between every pair of distinct vertices). Clearly, every Hamiltonian connected graph is Hamiltonian. In the context of the TSP, a *tour* is a Hamilton cycle, a *subtour* is an arbitrary cycle and a *partial tour* is

a subgraph of a tour. A trail W is *Euler* (or *Eulerian*) if $A(W) = A(G)$; an Eulerian trail W is an *Eulerian tour* if W is closed; G is *Eulerian* if G has an Euler tour. An undirected graph G is *connected* if there exists a path between any pair of vertices in G . The *underlying graph* of a digraph D is the graph G such that $V(G) = V(D)$ and a pair of vertices are adjacent in G if and only if they adjacent in D . A digraph D is *connected* if its underlying graph is connected. A digraph D is *strongly connected* (or *strong*) if D contains a path from x to y for every ordered pair x, y of vertices. The following result is well known:

Theorem A-1 *The following three statements are equivalent for a directed (undirected) graph $G = (V, A)$:*

1. G is Eulerian;
2. G is connected and $d^+(x) = d^-(x)$ ($d(x)$ is even) for every vertex x in G ;
3. G is connected and A can be decomposed into arc-disjoint (edge-disjoint) cycles.

A digraph D is *acyclic* if it has no cycle. An undirected graph with no cycle is a *forest*; a connected forest is a *tree*. A *wheel* W_n is a graph consisting of a vertex u , called the *center*, adjacent to each of the remaining vertices v_1, v_2, \dots, v_{n-1} , and the cycle $v_1 v_2 \dots v_{n-1} v_1$; the edges uv_i are predictably called *spikes*. A *plane graph* is a graph embedded in the plane such that the interiors of the edges (depicted as arcs between their vertices) do not intersect. Consider a plane tree T with no vertex of degree 2. Join the degree 1 vertices of T by a cycle so that the resulting graph remains plane. This graph is called a *Halin graph*. Clearly, every wheel is a Halin graph. Let x be a vertex in T such that x is adjacent to several vertices of degree one and exactly one vertex of degree more than one in T . The subgraph induced by x and all its adjacent vertices of degree one is called the *fan* at x .

For a set X of vertices of a graph $G = (V, E)$ the operation of *contraction* consists of replacing X in G by a new vertex r such that r is adjacent to a vertex $v \in V - X$ if and only if there is an $x \in X$ such that $xv \in E$; the edges in $G[V - X]$ remain the same. The following properties of Halin graphs are used in this book.

Theorem A-2 1. *Every Halin graph is Hamiltonian connected.*

2. *If we contract a fan of a Halin graph (with at least two fans), then the resulting graph will also be Halin.*

A directed (undirected) graph G is *bipartite* if its vertices can be partitioned into two sets U, W such that every arc (edge) of G has one end vertex in U and the other in W . We denote a bipartite directed (undirected) graph by $G = (U \cup W, E)$, where U, W is the bipartition above. It is well known that an undirected (strongly connected directed) graph G is bipartite if and only if every cycle of G is of even length [84].

A *weighted digraph* is a digraph $D = (V, A)$ along with a mapping $c : A \rightarrow \mathbb{R}$, where \mathbb{R} is the set of real numbers. Thus, a weighted digraph is a triple $D = (V, A, c)$. If a is an arc of a weighted digraph $D = (V, A, c)$, then $c(a)$ is called the *weight* or *cost* of a . The weight or cost $c(A')$ of a subset $A' \subseteq A$ is the sum of the costs of arcs of A' . The weight of a subgraph $H = (V', A')$ of D is $c(A')$. If the vertices of $D = (V, A, c)$ are labeled, say x_1, x_2, \dots, x_n , it is convenient to determine the cost of the arcs by the *cost matrix* with entries $c_{ij} = c(x_i, x_j)$ for $i \neq j$ and $c_{ii} = 0$. The definitions and notation of this paragraph can be readily extended to undirected graphs. The following result is well-known (see, e.g., [570]).

Theorem A-3 *Maximum (minimum) weight 1-factor of a weighted directed and undirected graph on n vertices can be computed in time $O(n^3)$.*

For 2-factors, a similar result holds.

Theorem A-4 [435] *Maximum (minimum) weight 2-factor of a weighted undirected graph on n vertices can be computed in time $O(n^3)$.*

When a directed or undirected graph H admits parallel arcs or edges (i.e., arcs or edges with the same end-vertices), we speak of a *directed* or *undirected multigraph*. If also loops (i.e., arcs or edges of the form (x, x)) are allowed, we speak of *directed* or *undirected pseudographs*. The majority of definitions for directed and undirected graphs can be easily and naturally extended to directed and undirected pseudographs.

3. Permutations

Let π be an arbitrary permutation on a set $N = \{1, 2, \dots, n\}$. We interpret π as an assignment of a unique successor $\pi(i)$ to each element i of N such that each element of N has a unique predecessor. We associate with π a digraph, $D_\pi = (N, A_\pi)$, where $A_\pi = \{(i, \pi(i)) : i \in N\}$. Let $D_1 = (N_1, A_1), D_2 = (N_2, A_2), \dots, D_r = (N_r, A_r)$ be the connected components of D_π . Then, for each $1 \leq i \leq r$, D_i defines a subtour \mathcal{C}_i on the node set N_i . We call $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_r$ the *subtours* of π . If $r = 1$, then π defines a *tour* on N and we call such a permutation π a *tour*. Subtours of π on more than one node are called *non-trivial subtours* of π . A permutation with a single non-trivial subtour is called a *circuit*. A circuit with its only non-trivial subtour of the form (i, j, i) is called a *transposition* and is denoted by α_{ij} . A transposition of the form $\alpha_{i, i+1} = \alpha_{i+1, i}$ is called an *adjacent transposition* and is denoted by β_i . We denote by ξ the identity permutation, (that is, $\xi(i) = i$ for all $i \in N$).

For any two permutations π and ψ on the set N we define $\pi \circ \psi$, *product* of π with ψ , as $\pi \circ \psi(i) = \pi(\psi(i))$ for all $i \in N$. For any permutation π there exists a unique permutation, which we denote by π^{-1} , such that $\pi \circ \pi^{-1} = \pi^{-1} \circ \pi = \xi$. For a permutation π , $\pi^{-1} = \pi$ if and only if all the non-trivial subtours of π are of size 2. In particular, for any transposition α_{ij} , $\alpha_{ij}^{-1} = \alpha_{ij}$.

Appendix: B. Computational Complexity

Abraham P. Punnen

*Department of Mathematical Sciences
University of New Brunswick-Saint John
New Brunswick, Canada
punnen@unbsj.ca*

1. Introduction

In this appendix we give an overview of complexity results for the traveling salesman problem. For details of fundamentals of complexity theory we refer to the books by Garey and Johnson [347], Papadimitriou [652], and Papadimitriou and Steiglitz [655]. The book by Ausiello et al [49] gives a thorough discussion of complexity theory in the context of approximation algorithms for various combinatorial optimization problems. Several complexity results of fundamental importance in the case of the traveling salesman problem are discussed in [466]. The survey paper by Melamed et al [590] also provides a summary of various results on complexity of TSP.

An optimization problem \mathcal{P} can be represented by a 4-tuple $(I_{\mathcal{P}}, \mathbb{F}_{\mathcal{P}}, f_{\mathcal{P}}, \text{goal}_{\mathcal{P}})$ [49], where

1. $I_{\mathcal{P}}$ is the collection of all instances of \mathcal{P}
2. $\mathbb{F}_{\mathcal{P}}$ is a set valued function with domain $I_{\mathcal{P}}$. For $X \in I_{\mathcal{P}}$, $\mathbb{F}_{\mathcal{P}}(X)$ is the family of feasible solutions of the instance X .
3. $f_{\mathcal{P}}$ is the measure function which is used to compare two solutions. Let $\mathbb{P} = \{(X, S) : X \in I_{\mathcal{P}} \text{ and } S \in \mathbb{F}_{\mathcal{P}}(X)\}$. Then $f_{\mathcal{P}} : \mathbb{P} \rightarrow \mathbb{Z}$.
4. $\text{goal}_{\mathcal{P}}$ indicates whether the optimization problem is a minimization problem or a maximization problem.

We are primarily concerned with minimization problems and hence represent an optimization problem in minimization form by the triplet $(I_{\mathcal{P}}, \mathbb{F}_{\mathcal{P}}, f_{\mathcal{P}})$. Also for simplicity we drop the suffix \mathcal{P} in the triplet whenever the context is obvious. Given an instance $X \in I$, $S^* \in \mathbb{F}(X)$ is an *optimal solution* to X if $f(X, S^*) \leq f(X, S)$ for all $S \in \mathbb{F}(X)$.

Let $G = (V, E)$ be a complete graph with a prescribed cost c_e for each edge $e \in E$. Then the symmetric traveling salesman problem (TSP) is to find a least cost hamiltonian cycle in G . This problem is characterized by the triplet (I, \mathbb{F}, f) where

1. $I = \{G = (V, E) : G \text{ is a complete graph with prescribed cost } c_e \text{ for each } e \in E\}$;
2. $\mathbb{F}(G) =$ Set of all hamiltonian cycles in G ;
3. $f(G, H) = \sum_{e \in H} c_e$ for any $H \in \mathbb{F}(G)$.

A *decision problem* is an ‘yes’ or ‘no’ question. Let \mathcal{P} be a decision problem and $I_{\mathcal{P}}$ be the collection of all instances of \mathcal{P} . Then $I_{\mathcal{P}}$ can be partitioned into two classes $Y_{\mathcal{P}}$ and $N_{\mathcal{P}}$, where $Y_{\mathcal{P}}$ contains all instances with answer *yes* and $N_{\mathcal{P}}$ contains all instances with answer *no*. Thus given an instance $X \in I_{\mathcal{P}}$, a decision problem is to determine if $X \in Y_{\mathcal{P}}$.

The decision version of an optimization (minimization) problem can be described as follows: Given an instance $X \in I_{\mathcal{P}}$ and an integer K , “does there exist an $S \in \mathbb{F}_{\mathcal{P}}(X)$ such that $f_{\mathcal{P}}(X, S) \leq K$?”

An algorithm α for a computational problem \mathcal{P} (optimization or decision version) is said to be a *polynomial time (space) algorithm* if any $X \in I_{\mathcal{P}}$ can be solved in time (memory space) $O(n^k)$ for some constant k , where n is the input size [652] of X . The collection of decision problems that can be solved by a polynomial time algorithm constitutes the class P . A superset of P , called NP , is defined as the collection of decision problems \mathcal{P} where every $X \in Y_{\mathcal{P}}$ has a concise certificate the validity of which can be verified in polynomial time. For example, consider the decision version of the symmetric traveling salesman problem:

“Given a complete graph $G = (V, E)$ with cost c_e for each $e \in E$ and an integer K , does there exist a hamiltonian cycle H in G such that $\sum_{e \in H} c_e \leq K$?”

For an ‘yes’ instance of this problem a concise certificate would be a collection of edges that form a hamiltonian cycle in G with cost less than or equal to K . Clearly, whether this collection of edges forms a hamiltonian cycle in G and its cost is less than or equal to K can be verified in $O(n)$ time, where $n = |V|$. Thus the decision version of symmetric TSP is in NP .

An important decision problem in the study of computational complexity theory is the satisfiability problem (SAT). It can be stated as follows. “Given m clauses B_1, B_2, \dots, B_m involving boolean variables x_1, x_2, \dots, x_n does there exist a truth assignment such that the product of B_1, B_2, \dots, B_m is satisfiable? [347]”

SAT is clearly in NP since for an ‘yes’ instance of SAT, a concise certificate would be the values of the boolean variable and a direct substitution of these values in the boolean formula verifies the validity of the certificate.

Let us consider another decision problem. “Given a graph $G = (V, E)$ and an integer K , do there exist $\lceil \frac{|V|}{K} \rceil!$ hamiltonian cycles in G ?”

This problem is not known to be in NP since we do not know a concise certificate for an ‘yes’ instance of this problem, the validity of which can be verified in polynomial time. One possible certificate is a list of $\lceil \frac{|V|}{K} \rceil!$ hamiltonian cycles. However verifying each member of this list is indeed a hamiltonian cycle and counting there are $\lceil \frac{|V|}{K} \rceil!$ of them takes time exponential in $|V|$ and hence the certificate is not concise.

An algorithm for a computational problem \mathcal{P} (optimization or decision version) is said to be a *pseudopolynomial time* algorithm if for every $X \in I_{\mathcal{P}}$, its running time is bounded above by a polynomial function in $size(X)$, the input size of X , and $number(X)$, the largest number in the instance X [655]. For SAT, $number(X)$ is less than or equal to $size(X)$ whereas for the TSP, $number(X)$ is the largest cost of edges in G which may or may not be greater than $size(X)$.

A decision problem \mathcal{P}_1 is said to be *reducible* to another decision problem \mathcal{P}_2 if there exists an algorithm \mathcal{A} which with input any instance $X \in I_{\mathcal{P}_1}$, produces $Z \in I_{\mathcal{P}_2}$ as output such that $X \in Y_{\mathcal{P}_1}$ if and only if $Z \in Y_{\mathcal{P}_2}$. We call the algorithm \mathcal{A} a *reduction* from \mathcal{P}_1 to \mathcal{P}_2 . If \mathcal{A} is a polynomial time algorithm with $size(Z)$ bounded above by polynomial function of $size(X)$ then we say that \mathcal{P}_1 is *polynomially reducible* to \mathcal{P}_2 and denote this by $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2$. If \mathcal{P}_1 is polynomially reducible \mathcal{P}_2 and $number(Z)$ is bounded above by a polynomial function of $size(X)$, then we say that \mathcal{P}_1 is *polynomially reducible to \mathcal{P}_2 in the strong sense*, which is denoted by $\mathcal{P}_1 \rightarrow \mathcal{P}_2$.

A decision problem \mathcal{P} is said to be *NP-complete* if it is in NP and for any $\mathcal{P}' \in NP$, $\mathcal{P}' \rightsquigarrow \mathcal{P}$.

Theorem B-1 (Cook-Levin theorem) [206, 560] *SAT is NP-complete.*

From Cook-Levin theorem and transitivity of polynomial time reduction, we have the following recursive definition of the class of NP-complete problems.

- 1) SAT is NP-complete.
- 2) If \mathcal{P}_1 is NP-complete, and $\mathcal{P}_2 \in \text{NP}$ is such that $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2$ then \mathcal{P}_2 is NP-complete.

A computational problem (optimization or decision) is said to be NP-hard if it is as 'hard' as an NP-complete problem but not necessarily in NP. A recursive definition of the class of NP-hard problems is given below.

- 1) SAT is NP-hard.
- 2) If \mathcal{P}_1 is NP-hard and $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2$ then \mathcal{P}_2 is NP-hard.

The class of strongly NP-complete (strongly NP-hard) problems can be defined recursively in a similar manner by replacing 'NP-complete' by 'strongly NP-complete', ('NP-hard' by 'strongly NP-hard'), and 'polynomial reduction' by 'polynomial reduction in the strong sense' in the recursive definition of the class of NP-complete (NP-hard) problems. We are now ready to describe our first set of complexity results.

2. Basic Complexity Results

Problem: The hamiltonian cycle problem (HC)

Instance: Given a graph G , is it hamiltonian?

Results: HC is strongly NP-complete even if G is

- 1) a bipartite graph where each node is of degree 2 or 3 [466];
- 2) a cubic bipartite planar graph [13];
- 3) a grid graph [466];
- 4) a 3-connected planar cubic graph [13];
- 7) a graph G such that G^2 is known to be hamiltonian [797];
- 8) a graph with a given hamiltonian path [590];
- 9) a planar triangulation with maximum vertex degree at least 6 [196];
- 10) a 3-regular graph [590];
- 11) graphs that are squares of some other graphs [590].

Remarks: As a direct consequence, TSP when restricted to any of these special graphs is strongly NP-hard. There are specially structured graphs on which TSP can be solved in polynomial time. These include bandwidth limited graphs, Halin graphs and other graphs with 3-edge cutsets (see Chapter 11).

Problem: Second hamiltonian cycle (SHC)

Instance: Given a graph G and a hamiltonian cycle in G , does there exist another hamiltonian cycle in G ?

Results: SHC is strongly NP-complete [466]

Problem: Traveling salesman problem-Decision version (TSPD).

Instance: Given a complete graph G , a cost matrix C of rational numbers and a rational number K , does there exist a tour H in G such that $\sum_{ij \in H} c_{ij} \leq K$?

Results: TSPD is NP-complete even if

- 1) the elements of C have exactly two distinct values;
- 2) the elements of C satisfy the triangle inequality;
- 3) the $(i, j)^{th}$ element of C is given by $c_{ij} = a_i b_j$, where a_1, a_2, \dots, a_n and b_1, b_2, \dots, b_n are rational numbers [741];
- 4) the vertices are represented by bit strings and c_{ij} is the Hamming distance between the vertices i and j [282].

Remarks: There are several classes of cost matrices C such that the corresponding traveling salesman problem can be solved in polynomial time (see Chapter 11).

Problem: Euclidean Traveling salesman problem (ETSP)

Instance: Given n points in plane, does there exist a tour through these points such that the total distance traveled is at most K , where K is a rational number and the distance between two points is calculated using the Euclidean norm.

Results: 1) ETSP is strongly NP-complete [466];
 2) If the distances are calculated as Hamming distance, the resulting TSP is strongly NP-complete [795].

Remarks: The traveling salesman problem corresponding to points in \mathbb{R}^d with distances measured as polyhedral norm (Geometric TSP) is also NP-hard (see chapters 11, 5, 12).

Problem: Multiplicative TSP (MTSP)

Instance: Given a graph G with two edge costs, c_e and w_e , does there exist a tour H in G such that $(\sum_{e \in H} c_e)(\sum_{e \in H} w_e) \leq K$ for a given constant K ?

Result: MTSP is NP-complete on a Halin graph [679].

Problem: Generalized 1-regular factor problem (GORFP).

Instance: Let $D = (V, A)$ be a digraph and let V_1, V_2, \dots, V_k be a partition of V . Check whether D has a 1-regular subdigraph containing exactly one vertex from each V_i .

Result: GORFP is NP-hard [422].

Problem: TSP suboptimality (TSPS)

Instance: Given a complete graph G with edge cost c_e and a tour H in G , does there exist a tour H^* in G such that $\sum_{e \in H} c_e < \sum_{e \in H^*} c_e$?

Result: TSPS is strongly NP-complete [466].

Remarks: TSPS remains strongly NP-complete even if the edge costs satisfy triangle inequality.

Problem: TSP nonadjacency (TSPN)

Instance: Given two tours H and H^* on a complete graph G , are H and H^* non-adjacent vertices of the TSP polytope?

Result: TSPN is NP-complete [466].

Problem: TSP k -median

Instance: Given a complete graph G , cost matrix C , and constant integers $k > p \geq 1$, compute the value $c(H) = \sum_{e \in H} c_e$ of a tour H such that there are exactly $\lfloor (n-1)p/k \rfloor$ tours with value greater than or equal to $c(H)$.

Result: TSP k -median is strongly NP-hard [684].

Remarks: When $k = 2$ the above problem reduces to computing the median of tour values. Note that the average of tour values can be obtained in polynomial time by a direct formula (see Chapter 6).

Problem: TSP relaxation (TSPR)

Instance: Given a complete graph G with edge cost c_e , does the subtour relaxation of the TSP defined on G have an integer optimal solution?

Result: TSPR is NP-complete [197].

Remarks: Given that the subtour relaxation has an integer optimal solution, identifying such a solution is an NP-hard problem. For related results on 2-matching and comb constraints or 2-matching and clique tree inequalities we refer to [698].

Problem: k th best TSP (TSPK)

Instance: Given a complete edge weighted graph G and integer k , find the k th best tour in G .

Result: TSPK is NP-hard [356].

3. Complexity and Approximation

For many heuristic algorithms for TSP, the quality of the solution produced may depend on some 'choices' made in the algorithm. For the Gilmore-Gomory patching algorithm (see Chapter 11), the order in which subtours are selected for patching affects the quality of the solution produced. A *best GG-patching tour* is the best tour obtained from all possible GG-patchings of given subtours. Similarly in the Christofides algorithm, the tour produced depends on the selection of a starting node for the Eulerian traversals and the order in which the edges are traversed. A *best Christofides tour* is the tour obtained by considering all possible Eulerian traversals and short cuttings in the Christofides algorithm. In the same way one can define a *best double-tree tour* for the double tree algorithm.

Problem: TSP best GG-patching (TSPGGP)

Instance: Given a complete directed graph G with edge weights, a node numbering and a collection of node disjoint spanning subtours, find the best GG-patching tour in G .

Result: TSPGGP is NP-hard (see Chapter 11 and [741]).

Problem: TSP best heuristic solution (TSP-BHS)

Instance: Given a complete graph G with edge weights, find the best Christofides (best double tree) tour in G .

Result: TSP-BHS is strongly NP-hard [657, 684].

Problem: TSP polynomial 2-Opt (TSP-P2-Opt)

Instance: Given a complete graph G with edge weights, a tour H in G , and a positive integer d , starting from the initial solution H is it possible to reach a local optimum with respect to 2-Opt neighborhood in d iterations of the 2-Opt heuristic?

Result: TSP-P2-Opt is strongly NP-complete [292].

Remarks: Using 2-Opt, a solution with objective function value at least the average value of all tours in G can be reached in $O(n^3 \log n)$ iterations [684].

Problem: All but constant domination (ABCD)

Instance: Given a complete directed graph G on n nodes with arc weights, does there exist a polynomial time heuristic algorithm for the TSP on G with domination number at least $(n - 1)! - k$ for any constant k ?

Result: ABCD is strongly NP-hard [684].

Problem: All but factorial domination (ABFD)

Instance: Given a complete directed graph G on n nodes with arc weights, does there exist a polynomial time heuristic algorithm for the TSP on G with domination number at least $(n - 1)! - (\frac{k}{k+1}(n + r)! - 1$ for any constants k and r such that $k + 1 \equiv 0 \pmod{n} + r$?

Result: ABFD is strongly NP-hard [684].

For any heuristic algorithm α for the TSP with edge weights c_e , let $HUR(\alpha, C)$ be the objective function value of the solution produced and $OPT(C)$ be the objective function value of an optimal solution. One of the most commonly used measures of the worst-case performance of an algorithm is the performance ratio. The *performance ratio* of the algorithm α when the cost matrix C is restricted to the domain \mathbb{D} is given by

$$P_\alpha(\mathbb{D}) = \sup_{C \in \mathbb{D}} \left\{ \frac{HUR(\alpha, C)}{OPT(C)} : OPT(C) \neq 0 \right\}.$$

Clearly $P_\alpha(\mathbb{D}) \geq 1$. The closer the performance ratio is to one, the better is the worst case performance of the algorithm α . Identifying the exact value of $P_\alpha(\mathbb{D})$ is usually difficult and hence upper bounds on the performance ratio are used to measure the quality of an approximation algorithm. An approximation algorithm α is called a δ -approximation algorithm with respect to the domain \mathbb{D} if $P_\alpha(\mathbb{D}) \leq \delta$.

Problem: Traveling salesman problem (TSP)

Instance: Given a complete graph G (directed or undirected) and a cost matrix C , find a least cost hamiltonian cycle in G .

Results: Unless $P=NP$, there is no polynomial time δ -approximation algorithm for the TSP for

1) any $\delta > 1$ [734],

- 2) $\delta = 5381/5380 - \epsilon$ for any $\epsilon > 0$ when C is symmetric and $c_{ij} = 1$ or 2 [273],
- 3) $\delta = 3813/3812 - \epsilon$ for any $\epsilon > 0$ when $c_{ij} = 1, 2,$ or 3 [117],
- 4) $\delta = 2805/2804 - \epsilon$ for any $\epsilon > 0$ when C is asymmetric and $c_{ij} = 1$ or 2 [273].

Remarks: When the edge weights satisfy triangle inequality, polynomial time approximation algorithms with $\delta = 3/2$ are known [189]. Improving this bound is an open problem. Approximation results with relaxations of the triangle inequality are given in [22] and with restrictions on triangle inequality are given in [117, 118, 525, 659, 795, 809]

There are several other interesting complexity classes that are relevant in the study of the traveling salesman problem. We do not discuss them in detail here. Interested reader may consult the references that follow. Syntactic classes such as MAX SNP [659] and computational classes such as APX [49] are studied in the context of approximation algorithms. For a comparative study of these classes we refer to [504]. Another interesting complexity class in the study of approximation algorithms (local search) is the class of PLS-complete problems [832]. The classes PO and NPO studied in the context of optimization problems are discussed in detail in the book [49]. Finally, for parametrized complexity and related complexity classes we refer to [261].

References

- [1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N Papakostantinou. The complexity of the traveling repairman problem. *RAIRO Informatique Theorique et Applications*, 20:79–87, 1986.
- [2] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.
- [3] A. Agnetis, D. Pacciarelli, and F. Rossi. Batch scheduling in two-machine flow shop with limited buffer. *Discrete Appl. Math.*, 72:243–260, 1997.
- [4] R.H. Ahmadi and J.W. Mamer. Routing heuristics for automated pick and place machines. *Eur. J. Oper. Res.*, 117:533–552, 1999.
- [5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.
- [6] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. Network flows. In G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd, editors, *Optimization, vol. I*, pages 211–370. North-Holland, 1989.
- [7] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [8] V.S. Aizenshtat and D.N. Kravchuk. Minimum of a linear form on the set of all complete cycles of the symmetric group S_n . *Cybernetics*, 4:52–53, 1968.
- [9] V.S. Aizenshtat and E.P. Maksimovich. Certain classes of traveling-salesman problems. *Cybernetics*, 14:565–569, 1979.
- [10] M. Ajtai, J. Komlós, and E. Szemerédi. The first occurrence of Hamilton cycles in random graphs. *Ann. Discrete Math.*, 27:173–178, 1985.
- [11] M. Ajtai, J. Komlós, and G. Tusnády. On optimal matchings. *Combinatorica*, 4:259–264, 1984.
- [12] M. Akcoglu and U. Krengel. Ergodic theorems for superadditive processes. *J. Reine Ang. Math.*, 323:53–67, 1981.
- [13] T. Akiyama, T. Nashizeki, and N. Saito. NP-completeness of the Hamiltonian cycle problem for the bipartite graph. *J. Inf. Process.*, 3:73–76, 1980.
- [14] D. Aldous and J.M. Steele. Asymptotics for Euclidean minimal spanning trees on random points. *Probab. Theory Relat. Fields*, 92:247–258, 1992.
- [15] K. Alexander. Rates of convergence of means for distance – minimizing subadditive Euclidean functionals. *Ann. Appl. Prob.*, 4:902–922, 1994.

- [16] K. Alexander. The RSW theorem for continuum percolation and the CLT for Euclidean minimal spanning trees. *Ann. Appl. Prob.*, 6:466–494, 1996.
- [17] S.M. Allen, D.H. Smith, and S. Hurly. Lower bounding techniques for frequency assignment. *Discrete Math.*, 197/198:41–52, 1999.
- [18] N. Alon and J.H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992. With an appendix by Paul Erdős.
- [19] B. Alspach, J.-C. Bermond, and D. Sotteau. Decomposition into cycles. I. Hamilton decompositions. In *Cycles and Rays (Montreal, PQ, 1987)*, pages 9–18. Kluwer, Dordrecht, 1990.
- [20] E. Althaus and K. Mehlhorn. Polynomial time TSP-based curve reconstruction. In *Proc. 11th ACM-SIAM Symp. Discrete Algorithms*, pages 686–695. SIAM, Philadelphia, 2000.
- [21] I. Althöfer, G. Das, D. Dobkin, and D. Joseph. On sparse spanners of weighted graphs. *Discrete Computat. Geometry*, 9:81–100, 1993.
- [22] T. Andreae and H.S. Bandelt. Performance guarantees for approximation algorithms depending on parametrized triangle inequality. *SIAM J. Discrete Math.*, 8:1–16, 1995.
- [23] Y.P. Aneja and H. Kamoun. Scheduling of parts and robot activities in a two machine robotic cell. *Comput. Oper. Res.*, 26:297–312, 1999.
- [24] D. Angluin and L.G. Valiant. Fast probabilistic algorithms for Hamilton circuits and matchings. *J. Comput. Syst. Sci.*, 18:155–193, 1979.
- [25] S. Anily, J. Bramel, and A. Hertz. A 5/3 approximation algorithm for the clustered traveling salesman tour and path problem. *Oper. Res. Lett.*, 24:29–35, 1999.
- [26] D. Applegate. Code provided to the authors in 1993.
- [27] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP, 1998. Draft available from <http://www.math.princeton.edu/tsp/>.
- [28] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP (preliminary report). Technical report, Computer Sciences, Rutgers University, 1994.
- [29] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. On the solution of traveling salesman problems. *Doc. Math., J. DMV*, Extra Vol. ICM Berlin 1998, III:645–656, 1998. The 12/15/1999 release of the Concorde code is currently available from <http://www.math.princeton.edu/tsp/concorde.html>.
- [30] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Project and lift (preliminary report). Technical report, Computer Sciences, Rutgers University, 1998.
- [31] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. TSP cuts outside the template paradigm. Technical report, Computer Sciences, Rutgers University, <http://www.cs.rutgers.edu/~chvatal/dagstuhl.ps>, 2000.
- [32] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS J. Comput.*, to appear.
- [33] E.M. Arkin, Y. Chiang, J.S.B. Mitchell, S.S. Skiena, and T. Yang. On the maximum scatter traveling salesman problem. *SIAM J. Comput.*, 29:515–544, 1999.

- [34] R.K. Arora and S.P. Rana. Scheduling in a semi-ordered flow-shop without intermediate queues. *AIIE Trans.*, 12(3):263–272, 1980.
- [35] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, 1998. Preliminary versions in Proc. 37th IEEE Symp. Found. Computer Sci., 1996, and Proc. 38th IEEE Symp. Found. Computer Sci., 1997.
- [36] S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. In *Proc. 31th Ann. ACM Symp. Theory Comput.*, 1999.
- [37] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *J. Assoc. Comput. Mach.*, 45:501–555, 1998.
- [38] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for the Euclidean k -medians and related problems. In *Proc. 30th ACM Symp. Theory Comput.*, pages 106–113, 1998.
- [39] T.S. Arthanari. On the Traveling Salesman Problem. Commun. XI Symp. Math. Program., Bonn, 1982.
- [40] T.S. Arthanari and M. Usha. On the equivalence of multistage-insertion and subtour elimination formulations. Technical report, Indian Stat. Institute, 1997.
- [41] T.S. Arthanari and M. Usha. An alternate formulation of the Symmetric Traveling Salesman Problem and its properties. *Discrete Appl. Math.*, 98:173–190, 2000.
- [42] T.S. Arthanari and M. Usha. On the equivalence of multistage-insertion and cycle-shrink formulations for the symmetric traveling salesman problem. *Oper. Res. Lett.*, 29:129–139, 2001.
- [43] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, Technische Universität Berlin, 1995.
- [44] N. Ascheuer, M. Fischetti, and M. Grötschel. A polyhedral study of the asymmetric travelling salesman problem with time windows. *Networks*, 36:69–79, 200.
- [45] N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Math. Program., Ser. A*, to appear.
- [46] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Comput. Optim. Appl.*, 17:61–84, 2000.
- [47] A.S. Asratian, T.M.J. Denley, and R. Häggkvist. *Bipartite Graphs and Their Applications*. Univ. Press, Cambridge, 1998.
- [48] T. Atan and N. Secomandi. A rollout-based application of Scatter Search/Path Relinking template to the multi-vehicle routing problem with stochastic demands and restocking. Technical report, PROS Revenue Management, Inc. Houston, TX, 1999.
- [49] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, Berlin, 1999.
- [50] Y. Azar. Lower bounds for insertion methods for TSP. *Comb. Probab. Comput.*, 3:285–292, 1994.

- [51] M.F. Baki. *Solvable Cases of Traveling Salesman Problem*. MBA thesis, Faculty of Administration, University of New Brunswick, Canada, 1995.
- [52] M.F. Baki. A note on a paper by Burkard, Demidenko and Rudolf. Working paper, Department of Management Sciences, University of Waterloo, 1997.
- [53] M.F. Baki. A note on Supnick and Kalmanson conditions. Working paper, Department of Management Sciences, University of Waterloo, 1997.
- [54] M.F. Baki. A note on van der Veen matrices. Working paper, Department of Management Sciences, University of Waterloo, 1997.
- [55] M.F. Baki and S.N. Kabadi. Generalization of some results on Euclidean Traveling Salesman Problem. Working Paper 95-007, Department of Management Sciences, University of Waterloo, 1995.
- [56] M.F. Baki and S.N. Kabadi. Some sufficient conditions for pyramidal optimal traveling salesman tour. Working Paper 95-021, Department of Management Sciences, University of Waterloo, 1995.
- [57] M.F. Baki and S.N. Kabadi. A note on recognition of Demidenko matrices. Working paper, Department of Management Sciences, University of Waterloo, 1997.
- [58] M.F. Baki and S.N. Kabadi. Some new pyramidally solvable cases of Traveling Salesman Problem. Working paper, Department of Management Sciences, University of Waterloo, 1997.
- [59] M.F. Baki and S.N. Kabadi. A generalization of the convex-hull-and-line traveling salesman problem. *J. Appl. Math. Decis. Sci.*, 2:177–192, 1998.
- [60] M.F. Baki and S.N. Kabadi. Pyramidal traveling salesman problem. *Comput. Oper. Res.*, 26:353–369, 1999.
- [61] E. Balas. The prize collecting traveling salesman problem. ORSA/Tims Meeting in Los Angeles (Spring 1986).
- [62] E. Balas. The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. *SIAM J. Discrete Math.*, 2:425–451, 1989.
- [63] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
- [64] E. Balas. Finding out whether a valid inequality is facet defining. In R. Kannan and W.R. Pulleyblank, editors, *Integer Programming and Combinatorial Optimization*, pages 45–61. Univ. Waterloo Press, 1990.
- [65] E. Balas. The prize collecting traveling salesman problem: II. Polyhedral results. *Networks*, 25:199–216, 1995.
- [66] E. Balas. New classes of efficiently solvable generalized traveling salesman problems. *Ann. Oper. Res.*, 86:529–558, 1999.
- [67] E. Balas. Personal communication, 2000.
- [68] E. Balas, S. Ceria, and G. Cornuéjols. A Lift-and-Project cutting plane algorithm for mixed 0-1 programs. *Math. Program. Ser. A*, 58:295–324, 1993.
- [69] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by Lift-and-Project in a Branch-and-Cut framework. *Manage. Sci.*, 42:1229–1246, 1996.

- [70] E. Balas and N. Christofides. A restricted Lagrangean approach to the Traveling Salesman Problem. *Math. Program., Ser. A*, 21:19–46, 1981.
- [71] E. Balas, F. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Math. Program. Ser. A*, 68:241–265, 1995.
- [72] E. Balas and M. Fischetti. The fixed-outdegree 1-arborescence polytope. *Math. Oper. Res.*, 17:1001–1018, 1992.
- [73] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Math. Program., Ser. A*, 58:325–352, 1993.
- [74] E. Balas and M. Fischetti. On the monotoneization of polyhedra. *Math. Program. Ser. A*, 78:59–84, 1997.
- [75] E. Balas and M. Fischetti. Lifted cycle inequalities for the asymmetric traveling salesman problem. *Math. Oper. Res.*, 24:273–292, 1999.
- [76] E. Balas and C.H. Martin. Combinatorial optimization in steel rolling (extended abstract). Workshop on Combinatorial Optimization in Science and Technology (COST), RUTCOR, Rutgers University (April 1991).
- [77] E. Balas and G. Martin. ROLL-A-ROUND: Software Package for Scheduling the Rounds of a Rolling Mill. Copyright Balas and Martin Associates, 104 Maple Heights Road, Pittsburgh, PA, 1986.
- [78] E. Balas and M. Oosten. On the dimension of projected polyhedra. *Discrete Appl. Math.*, 87:1–9, 1998.
- [79] E. Balas and M. Oosten. On the cycle polytope of a directed graph. *Networks*, 36:34–46, 2000.
- [80] E. Balas and N. Simonetti. Linear time dynamic programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS J. Comput.*, 13:56–75, 2001. The code is currently available from <http://www.contrib.andrew.cmu.edu/~neils/tsp/index.html>.
- [81] E. Balas and P. Toth. Branch and Bound Methods. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 361–401. Wiley, Chichester, 1985.
- [82] J.L. Balcazar, J. Diaz, and J. Gabarro. *Structural Complexity*, volume I of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 2 edition, 1995.
- [83] M. Ball and M. Magazine. Sequencing of insertion in printed circuit board assembly. *Oper. Res.*, 36:192–210, 1989.
- [84] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, London, 2000.
- [85] G. Barahona and M. Grötschel. The traveling salesman problem for graphs not contractible to $K_5 \setminus \{e\}$. Preprint 77, Mathematisches Institut-Universität Augsburg, Augsburg, Germany, 1985.
- [86] A. Barvinok. Two algorithmic results for the traveling salesman problem. *Math. Oper. Res.*, 21:65–84, 1996.

- [87] A. Barvinok, S.P. Fekete, D.S. Johnson, A. Tamir, G.J. Woeginger, and R. Woodroffe. The maximum traveling salesman problem. Technical Report ZPR98-333, University of Köln, 1998.
- [88] A. Barvinok, D.S. Johnson, G.J. Woeginger, and R. Woodroffe. The maximum traveling salesman problem under polyhedral norms. Technical Report Woe-14, Dept. Math., TU Graz, Austria, 1997.
- [89] A. Barvinok, D.S. Johnson, G.J. Woeginger, and R. Woodroffe. Finding maximum length tours under polyhedral norms. Working paper, Dept. of Mathematics, University of Michigan, 1998.
- [90] A. Barvinok, D.S. Johnson, G.J. Woeginger, and R. Woodroffe. The maximum traveling salesman problem under polyhedral norms. In *Proceedings of IPCO VI, Lect. Notes Comput. Sci.*, volume 1412, pages 195–201. Springer Verlag, Berlin, 1998.
- [91] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA J. Comput.*, 6(2):126–140, 1992.
- [92] P. Bauer. The cycle polytope: facets. *Math. Oper. Res.*, 22:110–145, 1997.
- [93] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems, 1986. Unpublished Manuscript.
- [94] J. Beardwood, J.H. Halton, and J.M. Hammersley. The shortest path through many points. *Proc. Camb. Philos. Soc.*, 55:299–327, 1959.
- [95] R. E. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach.*, 9:61–63, 1962.
- [96] M. Bellmore and S. Hong. Transformation of multisalesman problem to the standard traveling salesman problem. *J. Assoc. Comput. Mach.*, 21:500–504, 1974.
- [97] M. Bellmore and J.C. Malone. Pathology of traveling salesman subtour elimination algorithms. *Oper. Res.*, 19:278–307, 1971.
- [98] A. Belloni and A. Lucena. A relax and cut algorithm for the traveling salesman problem. Technical report, Laboratorio de Metodos Quantitativos, Departamento de Administracao, Universidade Federal do Rio de Janeiro, 2000.
- [99] E.A. Bender and E.R. Canfield. The asymptotic number of labelled graphs with given degree sequences. *J. Combin. Theory, Ser. A*, 24:296–307, 1978.
- [100] M.A. Bender and C. Chekuri. Performance guarantees for the TSP with a parameterized triangle inequality. *Inf. Process. Lett.*, 73:17–21, 2000.
- [101] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.
- [102] J. L. Bentley. K - d trees for semidynamic point sets. In *Proc. 6th ACM Symp. on Computational Geometry*, pages 187–197, 1990.
- [103] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA J. Comput.*, 4:387–411, 1992.
- [104] C. Berge. *The Theory of Graphs*. Methuen, London, 1958.
- [105] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. Unpublished manuscript, January 1995.

- [106] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtree and quality triangulations. In *Proc. 3rd WADS, Lect. Notes Comput. Sci. Vol. 709*, pages 188–199, 1993.
- [107] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inform. Process. Lett.*, 32:171–176, 1989.
- [108] D. Bertsimas and M. Grigni. On the space-filling curve heuristic for the Euclidean traveling salesman problem. *Oper. Res. Lett.*, 8:241–244, 1989.
- [109] D. Bienstock, M. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Math. Program., Ser. A*, 59:413–420, 1993.
- [110] A.E. Bixby, C. Coullard, and D. Simchi-Levi. A branch-and-cut algorithm for the capacitated prize-collecting traveling salesman problem. Working paper, Northwestern University, 1996.
- [111] R.G. Bland and D.F. Shallcross. Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation. *Oper. Res. Lett.*, 8:125–128, 1989.
- [112] D. Blokh and G. Gutin. Maximizing traveling salesman problem for special matrices. *Discrete Appl. Math.*, 56:83–86, 1995.
- [113] A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation for the k -MST problem in the plane. In *Proc. 27th ACM Ann. Symp. Theory Comput. (STOC'95)*, pages 294–302. L.A., California, 1995.
- [114] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J. Assoc. Comput. Mach.*, 41(4):630–647, 1994.
- [115] F. Bock. An algorithm for solving “traveling-salesman” and related network optimization problems. Unpublished manuscript associated with talk presented at the 14th ORSA National Meeting, 1958.
- [116] F. Bock. Mathematical programming solution of traveling salesman examples. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*. McGraw-Hill, New York, 1963.
- [117] H-J. Bockenhauer, J. Hromkovic, R. Klasing, S. Seibert, and W. Unger. An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In *Proc. 17th Symp. Theor. Aspects Comput. Sci., Lect. Notes Comput. Sci., Vol. 1770*, pages 382–394. Springer-Verlag, Berlin, 2000.
- [118] H-J. Bockenhauer, J. Hromkovic, R. Klasing, S. Seibert, and W. Unger. Towards the notion of stability of approximation of hard optimization tasks and the traveling salesman problem. In *Lect. Notes Comput. Sci., Vol. 1767*, pages 72–86. Springer-Verlag, Berlin, 2000.
- [119] F.F. Boctor, G. Laporte, and J. Renaud. Heuristics for the traveling purchaser problem. Working paper, Centre for research on transportation, Montreal, 2000.
- [120] L. Bodin, B. Golden, A. Assad, and M. Ball. The state of the art in the routing and scheduling of vehicles and crews. *Comput. Oper. Res.*, 10:63–211, 1983.
- [121] B. Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *Eur. J. Comb.*, 1:311–316, 1980.

- [122] B. Bollobás. Almost all regular graphs are Hamiltonian. *Eur. J. Comb.*, 4:97–106, 1983.
- [123] B. Bollobás. The evolution of sparse graphs. In B. Bollobás, editor, *Graph Theory and Combinatorics, Proc. Cambridge Combinatorics, Conference in Honour of Paul Erdős*, pages 35–57. Academic Press, 1984.
- [124] B. Bollobás. *Random Graphs*. Academic Press, 1985.
- [125] B. Bollobás. Complete matchings in random subgraphs of the cube. *Random Struct. Algorithms*, 1:95–104, 1990.
- [126] B. Bollobás. *Modern Graph Theory*. Springer, New York, 1998.
- [127] B. Bollobás, C. Cooper, T.I. Fenner, and A.M. Frieze. On Hamilton cycles in sparse random graphs with minimum degree at least k . In A. Baker, B. Bollobás, and A. Hajnal, editors, *A tribute to Paul Erdős*, pages 59–96. Academic Press, 1991.
- [128] B. Bollobás, T.I. Fenner, and A.M. Frieze. An algorithm for finding Hamilton paths and cycles in random graphs. *Combinatorica*, 7:327–341, 1987.
- [129] B. Bollobás and A.M. Frieze. On matchings and hamiltonian cycles in random graphs. *Ann. Discrete Math.*, 28:23–46, 1985.
- [130] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. American Elsevier, New York, 1979.
- [131] K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using pq -tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.
- [132] O. Borůvka. O Jistém Problému Minimálním (About a certain minimal problem). *Práce Moravské Přírodovědecké Společnosti v Brně (Acta Societ. Science. Natur. Moraviae)*, 3:37–58, 1926.
- [133] M. Bourgeois, G. Laporte, and F. Semet. Heuristics for the black and white traveling salesman problem. *Comput. Oper. Res.*, to appear, 2001.
- [134] S. Boyd, S. Cockburn, and D. Vella. On the domino-parity inequalities for the traveling salesman problem. Technical report, University of Ottawa, 2000.
- [135] S.C. Boyd and W.H. Cunningham. Small traveling salesman polytopes. *Math. Oper. Res.*, 16, 1991.
- [136] S.C. Boyd, W.H. Cunningham, M. Queyranne, and Y. Wang. Ladders for the traveling salesman. *SIAM J. Optimization*, 5:408–420, 1993.
- [137] J. Brest and J. Zerovnik. An approximation algorithm for the asymmetric traveling salesman problem. *Ricerca Operativa*, 28:59–67, 1998.
- [138] A. Broder, A.M. Frieze, and E. Shamir. Finding hidden Hamilton cycles. *Random Struct. Algorithms*, 5:395–410, 1994.
- [139] V.Y. Burdyuk and V.N. Trofimov. Generalization of the results of Gilmore and Gomory on the solution of the traveling salesman problem. *Engg. Cybernetics*, 11:12–18, 1976.
- [140] L. Buriol, P.M. França, and P. Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. Submitted for publication, 2001.

- [141] R.E. Burkard and V.G. Deineko. Polynomially solvable cases of the traveling salesman problem and a new exponential neighborhood. *Computing*, 54:191–211, 1995.
- [142] R.E. Burkard, V.G. Deineko, R. van Dal, J.A.A. van der Veen, and G.J. Woeginger. Well-solvable special cases of the traveling salesman problem: a survey. *SIAM Review*, 40(3):496–546, 1998.
- [143] R.E. Burkard, V.G. Deineko, and G.J. Woeginger. The travelling salesman problem on permuted Monge matrices. *J. Combin. Optim.*, 2:333–350, 1999.
- [144] R.E. Burkard, V.M. Demidenko, and R. Rudolf. A general approach for identifying special cases of the travelling salesman problem with a fixed optimal tour. Bericht 13, Karl-Franzens-Universität Graz & Technische Universität Graz, Austria, 1997.
- [145] R.E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Appl. Math.*, 70:95–161, 1996.
- [146] R.E. Burkard, J. Krarup, and P.M. Pruzan. Efficiency and optimality in minsum and minmax 0-1 programming problems. *J. Oper. Res. Soc.*, 33:137–151, 1982.
- [147] R.E. Burkard and W. Sandholzer. Effectively solvable special cases of bottleneck traveling salesman problems. *Discrete Appl. Math.*, 32:61–67, 1991.
- [148] R.E. Burkard and J.A.A. van der Veen. Universal conditions for algebraic travelling salesman problems to be efficiently solvable. *Optimization*, 22:787–814, 1991.
- [149] E.K. Burke, P.I. Cowling, and R. Keuthen. Embedded local search and variable neighborhood search heuristics applied to the travelling salesman problem. Unpublished manuscript, 2000.
- [150] E.K. Burke, P.I. Cowling, and R. Keuthen. Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. In E.J.W. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing, Proc. EvoWorkshops 2001*, Lect. Notes Comput. Sci., Vol. 2037, pages 203–212, Berlin, 2001. Springer-Verlag.
- [151] V.N. Burkov and M.I. Rubinshtein. A sufficient condition for existence of a Hamiltonian circuit and a new solvable case of traveling salesman problem. *Large Scale Systems*, 4:137–148, 1983.
- [152] R.M. Burstall. A heuristic method for a job sequencing problem. *Oper. Res. Quart.*, 17:291–304, 1966.
- [153] P.M. Camerini. The minmax spanning tree problem and some extensions. *Inf. Process. Lett.*, 7:10–14, 1978.
- [154] P.M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309–312, 1979.
- [155] P.M. Camerini, L. Fratta, and F. Maffioli. The k best spanning arborescences of a network. *Networks*, 10:91–110, 1980.
- [156] P. Camion. Chemins et circuits hamiltoniens des graphes complets. *C.R. Acad. Sci. Paris*, 249:2151–2152, 1959.

- [157] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a Scatter Search for the Linear Ordering Problem. Technical Report HCES-06-99, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, MS, 1999.
- [158] B. Cao and F. Glover. Tabu Search and Ejection Chains - application to a node weighted version of the cardinality-constrained TSP. *Manage. Sci.*, 43(7):908–921, 1997.
- [159] A. Caprara and M. Fischetti. 0-1/2 Chvátal-Gomory cuts. *Math. Program., Ser. A*, 74:221–236, 1996.
- [160] A. Caprara and M. Fischetti. Branch-and-cut algorithms: an annotated bibliography. In *Annotated Bibliographies in Combinatorial Optimization*, pages 45–63. John Wiley and Sons, Chichester, 1997.
- [161] A. Caprara, M. Fischetti, and A. Letchford. On the separation of maximally violated mod- k cuts. *Math. Program. Ser. A*, 87:37–56, 2000.
- [162] J. Carlier and P. Villon. A new heuristic for the travelling salesman problem. *RAIRO, Recherche Operationelle*, 24:245–253, 1990.
- [163] G. Carpaneto, M. Dell'Amico, and P. Toth. Algorithm CDT: a subroutine for the exact solution of large-scale asymmetric traveling salesman problems. *ACM Trans. Math. Softw.*, 21:410–415, 1995.
- [164] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large-scale asymmetric traveling salesman problems. *ACM Trans. Math. Softw.*, 21:394–409, 1995.
- [165] G. Carpaneto, S. Martello, and P. Toth. An algorithm for the bottleneck traveling salesman problem. *Oper. Res.*, 32:380–389, 1984.
- [166] G. Carpaneto and P. Toth. Some new branching and bounding criteria for the asymmetric traveling salesman problem. *Manage. Sci.*, 26:736–743, 1980.
- [167] G. Carpaneto and P. Toth. An algorithm for the solution of the bottleneck assignment problem. *Computing*, 27:179–187, 1981.
- [168] G. Carpaneto and P. Toth. Primal-dual algorithms for the assignment problem. *Discrete Appl. Math.*, 18:137–153, 1987.
- [169] R.D. Carr. *Polynomial separation procedures and facet determination for inequalities of the traveling salesman polytope*. PhD thesis, Department of Mathematics, Carnegie Mellon University, 1995.
- [170] R.D. Carr. Separating clique tree and bipartition inequalities in polynomial time. In E. Balas and J. Clausen, editors, *Proc. 4th IPCO Confer.*, Lect. Notes Comput. Sci., Vol. 920, pages 40–49. Springer, 1995.
- [171] R.D. Carr. Separating clique trees and bipartition inequalities having a fixed number of handles and teeth in polynomial time. *Math. Oper. Res.*, 22:257–265, 1997.
- [172] L. Cavique, C. Rego, and I. Themido. Subgraph Ejection Chains and Tabu Search for the Crew Scheduling Problem. *J. Oper. Res. Soc.*, 50:608–616, 1997.
- [173] L. Cavique, C. Rego, and I. Themido. A Scatter Search Algorithm for the Maximum Clique Problem. In *Essays and Surveys in Metaheuristics*. Kluwer, Boston, to appear.

- [174] E. Cela, R. Ruediger, and G.J. Woeginger. On the Barvinok rank of matrices. In The 2nd Aussois Workshop on Combinatorial Optimization, 1-7 February 1998, Abstracts, <http://dmawww.epfl.ch/rosa.mosaic/aussois/abstracts1998.html>.
- [175] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM J. Comput.*, 28:2133–2149, 1999.
- [176] R. Chandrasekaran. Recognition of Gilmore-Gomory traveling salesman problem. *Discrete Appl. Math.*, 14:231–238, 1986.
- [177] I. Chao, B.L. Golden, and E.A. Wasi. A new heuristic for the period traveling salesman problem. *Comput. Oper. Res.*, 22:553–565, 1995.
- [178] I.-M. Chao, B.L. Golden, and E.A. Wasi. A fast and effective heuristic for the orienteering problem. *Eur. J. Oper. Res.*, 88:475–489, 1996.
- [179] I. Charon and O. Hudry. The Noising Method: A new combinatorial optimization method. *Oper. Res. Lett.*, 14:133–137, 1993.
- [180] I. Charon and O. Hudry. Application of the Noising Method to the Traveling Salesman Problem. *Eur. J. Oper. Res.*, 125:266–277, 2000.
- [181] C. Chekuri, A. Goldberg, D. Karger, M. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. 8th ACM-SIAM Symp. Discrete Algorithms (SODA '97)*, pages 324–333. ACM/SIAM, 1997.
- [182] R. Cheng, M. Gen, and M. Sasaki. Film-copy deliverer problem using genetic algorithms. *Comput. Engin.*, 29:549–553, 1995.
- [183] S. Chopra and G. Rinaldi. The graphical asymmetric traveling salesman polyhedron: Symmetric inequalities. *SIAM J. Discrete Math.*, 9:602–624, 1996.
- [184] T. Christof. *Low-Dimensional 0/1-Polytopes and Branch-and-Cut in Combinatorial Optimization*. PhD thesis, Universität Heidelberg, 1997.
- [185] T. Christof and G. Reinelt. Combinatorial optimization and small polytopes. *TOP (Spanish Statistical and Operations Research Society)*, 4:1–64, 1996.
- [186] T. Christof and G. Reinelt. Decomposition and parallelisation techniques for enumerating the facets of 0/1-polytopes. Technical report, Universität zu Heidelberg, 1998.
- [187] T. Christof and G. Reinelt. Algorithmic aspects of using small instance relaxations in parallel branch-and-cut. *Algorithmica*, 2001. To appear.
- [188] N. Christofides. Bounds for the traveling salesman problem. *Oper. Res.*, 20:1044–1056, 1972.
- [189] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report CS-93-13, Carnegie Mellon University, 1976.
- [190] N. Christofides and S. Eilon. Algorithms for large-scale travelling salesman problems. *Oper. Res. Quart.*, 23:511–518, 1972.
- [191] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.
- [192] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. *Math. Program.*, 20:255–282, 1980.

- [193] G. Christopher, M. Farach, and M. Trick. The structure of circular decomposable matrices. In *Proc. ESA IV, Lect. Notes Comput. Sci. Vol. 1138*, pages 406–418. Springer-Verlag, New York, 1996.
- [194] V. Chvátal. On Hamilton's ideals. *J. Comb. Theory, Ser. B*, 12:163–168, 1972.
- [195] V. Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Math. Program.*, 5:29–40, 1973.
- [196] V. Chvátal. Hamiltonian cycles. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys, editors, *The traveling salesman problem*, pages 37–85. Wiley, Chichester, 1985.
- [197] V. Chvátal. A note on the traveling salesman problem. *Oper. Res. Lett.*, 8:77–78, 1989.
- [198] V. Chvátal, W. Cook, and M. Hartmann. On cutting-plane proofs in combinatorial optimization. *Linear Algebra Appl.*, 114/115:455–499, 1989.
- [199] V. Chvátal and P. Erdős. A note on Hamiltonian circuits. *Discrete Math.*, 2:111–113, 1972.
- [200] J. Cirasella, D.S. Johnson, L.A. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In A.L. Buchsbaum and J. Snoeyink, editors, *Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001*, Lect. Notes Comput. Sci., Vol. 2153, pages 32–59. Springer-Verlag, Berlin, 2001.
- [201] G. Clarke and J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.*, 12:568–581, 1964.
- [202] A. Claus. A new formulation for the traveling salesman problem. *SIAM J. Alg. Discrete Meth.*, 5:21–25, 1984.
- [203] J.-M. Clochard and D. Naddef. Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. In L. Wolsey and G. Rinaldi, editors, *Proc. 3rd IPCO Conference*, pages 291–311, 1993.
- [204] S. Cockburn. On the Domino Parity constraints for the traveling salesman problem. Technical report, University of Ottawa, 2000.
- [205] B. Codenotti, G. Manzini, L. Margara, and G. Resta. Perturbation: An efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS J. Comput.*, 8:125–133, 1996.
- [206] S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. Theory Comput.* ACM, 1971.
- [207] W. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS J. Comput.*, 11:38–148, 1999. Software available at <http://www.or.uni-bonn.de/home/rohe/matching.html>.
- [208] W. Cook and P.D. Seymour. A branch-decomposition heuristic for the TSP. In preparation.
- [209] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, New York, 1998.
- [210] C. Cooper. 1-pancyclic Hamilton cycles in random graphs. *Random Struct. Algorithms*, 4:469–472, 1993.

- [211] C. Cooper and A.M. Frieze. On the number of Hamilton cycles in a random graph. *J. Graph Theory*, 13:719–735, 1989.
- [212] C. Cooper and A.M. Frieze. Pancyclic random graphs. In M. Karonski, J. Jaworski, and A. Rucinski, editors, *Proc. Random Graphs '87*, pages 29–39. Wiley, Chichester, 1990.
- [213] C. Cooper and A.M. Frieze. Multicoloured Hamilton cycles in random graphs: an anti-Ramsey threshold. *Electron. J. Comb.*, 2:R19, 1995.
- [214] C. Cooper and A.M. Frieze. Hamilton cycles in random graphs and directed graphs. *Random Struct. Algorithms*, 16:369–401, 2000.
- [215] C. Cooper, A.M. Frieze, and M.J. Molloy. Hamilton cycles in random regular digraphs. *Comb. Probab. Comput.*, 3:39–50, 1994.
- [216] C. Cooper, A.M. Frieze, and B.A. Reed. Random regular graphs of non-constant degree. *Comb. Probab. Comput.*, to appear.
- [217] D. Coppersmith, P. Raghavan, and M. Tompa. Parallel graph algorithms that are efficient on average. *Inf. Comput.*, 81:318–333, 1989.
- [218] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, MA, 1990.
- [219] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related polyhedra. *Math. Program.*, 33:1–27, 1985.
- [220] G. Cornuéjols, D. Naddef, and W.R. Pulleyblank. Halin graphs and the traveling salesman problem. *Math. Program.*, 26:287–294, 1983.
- [221] G. Cornuéjols, D. Naddef, and W.R. Pulleyblank. The traveling salesman problem in graphs with 3-edge cutsets. *J. Assoc. Comput. Mach.*, 32:383–410, 1985.
- [222] G. Cornuéjols and G.L. Nemhauser. Tight bounds for Christofides' traveling salesman heuristic. *Math. Program. Ser. A*, 14:116–121, 1978.
- [223] G. Cornuéjols and W. Pulleyblank. A matching problem with side conditions. *Discrete Math.*, 29:135–159, 1980.
- [224] G. Cornuéjols and W. Pulleyblank. Perfect triangle-free 2-matchings. *Math. Program. Studies*, 13:1–7, 1980.
- [225] G. Cornuéjols and W. Pulleyblank. The travelling salesman problem and 0,2 matchings. *Ann. Discrete Math.*, 16:147–175, 1982.
- [226] S.S. Cosmadakis and C.H. Papadimitriou. The traveling salesman problem with many visits to few cities. *SIAM J. Comput.*, 13:99–108, 1984.
- [227] C.R. Coullard and W.R. Pulleyblank. On cycle cones and polyhedra. *Linear Algebra Appl.*, 114/115:613–640, 1989.
- [228] G.A. Croes. A method for solving traveling salesman problems. *Oper. Res.*, 6:791–812, 1958.
- [229] H. Crowder and M.W. Padberg. Solving large scale symmetric traveling salesman problems to optimality. *Manage. Sci.*, 26:495–509, 1980.
- [230] H.P. Crowder. Computational improvements of subgradient optimization. IBM Research Report RC 4907 (21841), 1974.

- [231] V-D. Cung, T. Mautor, P. Michelon, and A. Tavares. Scatter Search for the Quadratic Assignment Problem. In *Proc. 1996 IEEE International Confer. Evolutionary Comput.*, pages 165–169, 1996.
- [232] W.H. Cunningham and Y. Wang. Restricted 2-factor polytopes. *Math. Program. Ser. A*, 87:87–111, 2000.
- [233] J.R. Current and D.A. Schilling. The covering salesman problem. *Transp. Sci.*, 23:208–213, 1989.
- [234] J.R. Current and D.A. Schilling. The median tour and maximal covering problems. *Eur. J. Oper. Res.*, 73:114–126, 1994.
- [235] M. Cutler. Efficient special case algorithms for the n -line planar traveling salesman problem. *Networks*, 10:183–195, 1980.
- [236] D. Cvetkrović, V. Dimitrijević, and M. Milosavljević. A survey of some non-standard traveling salesman problems. *Yugoslav J. Oper. Res.*, 2:163–185, 1992.
- [237] A. Czumaj and A. Lingas. A polynomial time approximation scheme for Euclidean minimum cost k -connectivity. In *Proc. 25th Internat. Colloq. Automata Languages Programming*, pages 682–694, 1998.
- [238] M. Dam and M. Zachariassen. Tabu search on the geometric traveling salesman problem. In I.H. Osman and J.P. Kelly, editors, *Meta-heuristics: Theory and Applications*, pages 571–587. Kluwer Academic Publishers, Boston, 1996.
- [239] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large scale traveling salesman problem. *Oper. Res.*, 2:393–410, 1954.
- [240] J.B.J.M. De Kort. Bounds for the symmetric 2-peripatetic salesman problem. *Optimization*, 23:357–367, 1992.
- [241] A. de Vitis. The cactus representation of all minimum cuts in a weighted graph. Technical report, Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00185 Roma, Italy, 1997.
- [242] V.G. Deineko and V.L. Filonenko. On the reconstruction of specially structured matrices. In *Aktualnyje Problemy EVM i programirovaniye*, Dnepropetrovsk, DGU, 1979. (in Russian).
- [243] V.G. Deineko, R. Rudolf, and G.J. Woeginger. On the recognition of permuted Supnick and incomplete Monge matrices. *Acta Inform.*, 33:559–569, 1996.
- [244] V.G. Deineko, R. Rudolf, and G.J. Woeginger. Sometimes traveling is easy: the master tour problem. *SIAM J. Discrete Math.*, 11:81–93, 1998.
- [245] V.G. Deineko and V.N. Trofimov. On possible values of the function in the special traveling salesman problem. *Kibernetika*, 6:135–136, 1981. (in Russian).
- [246] V.G. Deineko, R. van Dal, and G. Rote. The convex-hull-and-line traveling salesman problem: a solvable case. *Inform. Process. Lett.*, 51:141–148, 1994.
- [247] V.G. Deineko and G.J. Woeginger. The convex-hull-and- k -line travelling salesman problem. *Inform. Process. Lett.*, 59:295–301, 1996.
- [248] V.G. Deineko and G.J. Woeginger. A study of exponential neighbourhoods for the traveling salesman problem and the quadratic assignment problem. *Math. Program., Ser. A*, 87:519–542, 2000.
- [249] M. Dell'Amico and P. Toth. Algorithms and codes for dense assignment problems: the state of the art. *Discrete Appl. Math.*, 100:17–48, 2000.

- [250] V.M. Demidenko. A special case of traveling salesman problems. *Izv. Akad. Navuk BSSR, Ser. Fiz.-mat. Navuk*, no. 5:28–32, 1976. (in Russian).
- [251] V.M. Demidenko. The traveling salesman problem with asymmetric matrices. *Izv. Akad. Navuk BSSR, Ser. Fiz.-Mat. Navuk*, 1:29–35, 1979. (in Russian).
- [252] V.M. Demidenko. An extension of Supnick conditions for the Asymmetric Traveling Salesman Problem and its application. Manuscript, Belorussian Academy of Sciences, 1995.
- [253] U. Derigs and U. Zimmerman. An augmenting path method for solving linear bottleneck assignment problems. *Computing*, 19:285–295, 1978.
- [254] R. Diestel. *Graph Theory*. Springer-Verlag, New York, 2000. 2nd edition, first edition 1997.
- [255] V. Dimitrijević and Z. Sarić. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Inf. Sci.*, 102:105–110, 1997.
- [256] G.A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc.*, 2:69–81, 1952.
- [257] V. Dobric and J.E. Yukich. Exact asymptotics for transportation cost in high dimensions. *J. Theor. Probab.*, 8:97–118, 1995.
- [258] J.J. Dongarra. Performance of various computers using standard linear equations software. Report CS-89-85, University of Tennessee, Knoxville, 1996.
- [259] U. Dorndorf and E. Pesch. Fast clustering algorithms. *ORSA J. Comput.*, 6:141–153, 1994.
- [260] N.N. Doroshko and V.I. Sarvanov. A minmax traveling salesman problem and Hamiltonian cycles in powers of graphs. *Vestsi Akad. Navuk BSSR, Ser. Fiz.-Mat. Navuk*, no. 6, 1981. (in Russian).
- [261] R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Springer-Verlag, Berlin, 1999.
- [262] M.E. Dyer and A.M. Frieze. On patching algorithms for random asymmetric travelling salesman problems. *Math. Program., Ser. A*, 46:361–378, 1990.
- [263] M.E. Dyer, A.M. Frieze, and C.J.H. McDiarmid. Partitioning heuristics for two geometric maximization problems. *Oper. Res. Lett.*, 3(5):267–270, 1984.
- [264] H. Edelsbrunner, G. Rote, and E. Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theoret. Comput. Sci.*, 66:157–180, 1989.
- [265] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *J. Res. Nat. Bur. Stand B*, 59:125–130, 1965.
- [266] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.
- [267] J. Edmonds. Optimum branchings. *J. Res. Natl. Bur. Stand., Sect. B*, 71B:233–240, 1967.
- [268] J. Edmonds. Matroid intersection. *Ann. Discrete Math.*, 4:39–49, 1979.
- [269] J. Edmonds and D.R. Fulkerson. Bottleneck extrema. *J. Comb. Theory*, 8:299–306, 1970.

- [270] J. Edmonds and E.L. Johnson. Matching: a Well-solved Class of Integer Linear Programs. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*. Gordon and Breach, New York, 1970.
- [271] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *Internat. Trans. Oper. Res.*, 7:5–31, 2000.
- [272] H. Emmons and K. Matur. Lot sizing in a no-wait flow shop. *Oper. Res. Lett.*, 17:159–164, 1995.
- [273] L. Engebresten. An explicit lower bound for TSP with distances one and two. Technical Report 46, Electronic Colloquium on Computational Complexity, 1998.
- [274] H. Enomoto, Y. Oda, and K. Ota. Pyramidal tours with step-backs and the asymmetric traveling salesman problem. *Discrete Appl. Math.*, 87:57–65, 1998.
- [275] D. Eppstein. Sequence comparison with mixed convex and concave costs. *J. Algorithms*, 11:85–101, 1990.
- [276] P. Erdős. Remarks on a paper by Posa. *Magyar Tud. Akad. Mat. Kut. Int. Kozl.*, 7:227–229, 1962.
- [277] P. Erdős and A. Rényi. On random graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.
- [278] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5:17–61, 1960.
- [279] P. Erdős and A. Rényi. On the strength of connectedness of a random graph. *Acta. Math. Acad. Sci. Hungar.*, 12:261–267, 1963.
- [280] P. Erdős and A. Rényi. On random matrices. *Publ. Math. Hungar. Acad. Inst. Sci.*, 8:455–461, 1964.
- [281] P. Erdős and J. Spencer. Evolution of the n -cube. *Comput. Math. Appl.*, 5:33–39, 1979.
- [282] J. Ernvall, J. Katajainen, and M. Penttonen. NP-completeness of the Hamming salesman problem. *BIT*, 25:289–292, 1985.
- [283] R. Euler and H. Le Verge. Complete linear description of small asymmetric traveling salesman polytopes. *Discrete Appl. Math.*, 62:193–208, 1995.
- [284] S.P. Fekete. Simplicity and hardness of the maximum traveling salesman problem under geometric distances. Technical Report 98.329, Center for Applied Computer Science, Universitat zu Koln, 1998.
- [285] S.P. Fekete. Simplicity and hardness of the maximum traveling salesman problem under geometric distances. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 1999)*, pages 337–345. ACM, New York, 1999.
- [286] S.P. Fekete and H. Meijer. On minimum stars and maximum matchings. *Discrete Comput. Geom.*, 23:389–407, 2000.
- [287] S.P. Fekete, H. Meijer, A. Rohe, and W. Tietze. Good and fast heuristics for large geometric maximum matching and maximum traveling salesman problems. Manuscript, 2000.
- [288] Z. Fencl. Algorithm 456 - routing problem [H]. *Commun. ACM*, 16:572–574, 1973.

- [289] T.I. Fenner and A.M. Frieze. Hamiltonian cycles in random regular graphs. *J. Comb. Theory, Ser. B*, 38:103–112, 1984.
- [290] A. Fink, G. Schneidereit, and S. Voss. Solving general ring network design problems by meta-heuristics. In M. Laguna and J.L. González Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation*. Kluwer, Boston, 2000.
- [291] G. Finke, A. Claus, and E. Gunn. A two commodity network flow approach to the traveling salesman problem. *Congres. Numer.*, 41:167–178, 1984.
- [292] S.T. Fischer. A note on the complexity of local search problems. *Inf. Process. Lett.*, 53:69–75, 1995.
- [293] F. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Oper. Res.*, 41:1055–1064, 1993.
- [294] M. Fischetti. An improved bound for the asymmetric travelling salesman problem. *Ricerca Operativa*, 37:71–85, 1986.
- [295] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Math. Oper. Res.*, 16:42–56, 1991.
- [296] M. Fischetti. Three facet lifting theorems for the asymmetric traveling salesman polytope. In E. Balas, G. Cornuéjols, and R. Kannan, editors, *Proc. Integer Program. Comb. Optimization (IPCO) 2*, pages 260–273. GSIA, Carnegie Mellon University, 1992.
- [297] M. Fischetti. Clique tree inequalities define facets of the asymmetric traveling salesman problem. *Discrete Appl. Math.*, 56:9–18, 1995.
- [298] M. Fischetti, A. Lodi, S. Martello, and P. Toth. A polyhedral approach to simplified crew scheduling and vehicle scheduling problems. *Manage. Sci.*, 47:833–850, 2001.
- [299] M. Fischetti, J.J. Salazar, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26:113–123, 1995.
- [300] M. Fischetti, J.J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Oper. Res.*, 45:378–394, 1997.
- [301] M. Fischetti, J.J. Salazar, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS J. Comput.*, 10:133–148, 1998.
- [302] M. Fischetti and P. Toth. An additive approach for the optimal solution of the prize-collecting travelling salesman problem. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 319–343. North-Holland, 1988.
- [303] M. Fischetti and P. Toth. An additive bounding procedure for combinatorial optimization problems. *Oper. Res.*, 37:319–328, 1989.
- [304] M. Fischetti and P. Toth. An additive bounding procedure for the asymmetric travelling salesman problem. *Math. Programm., Ser. A*, 53:173–197, 1992.
- [305] M. Fischetti and P. Toth. An efficient algorithm for the min-sum arborescence problem. *ORSA J. Comput.*, 5:325–352, 1993.
- [306] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Manage. Sci.*, 43:1520–1536, 1997.

- [307] M. Fischetti and D. Vigo. A branch and cut algorithm for the resource-constrained min-sum arborescence problem. *Networks*, 29:55–67, 1997.
- [308] M.L. Fisher, G.L. Nemhauser, and L.A. Wolsey. An analysis of approximations for finding a maximum weight hamiltonian circuit. *Oper. Res.*, 27(4):799–809, 1979.
- [309] L. Fleischer. Building Chain and Cactus Representations of All Minimum Cuts from Hao-Orlin in the Same Asymptotic Run Time. *J. Algorithms*, 33(1):51–72, 1999.
- [310] L. Fleischer and É. Tardos. Separating maximally violated comb inequalities in planar graphs. *Math. Oper. Res.*, 24:130–148, 1999.
- [311] B. Fleischmann. A new class of cutting planes of the symmetric traveling salesman problem. *Math. Program. Ser. A*, 40:225–246, 1988.
- [312] H. Fleischner. The square of every 2-connected graph is Hamiltonian. *J. Comb. Theory, Ser. B*, 16:29–34, 1974.
- [313] C. Fleurent, F. Glover, P. Michelon, and Z. Valli. A Scatter Search approach for unconstrained continuous optimization. In *Proc. 1996 IEEE International Confer. Evolutionary Comput.*, pages 643–648, 1996.
- [314] M.M. Flood. The traveling-salesman problem. *Oper. Res.*, 4:61–75, 1956.
- [315] F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. Technical Report OR/01/2, D.E.I.S., University of Bologna, 2001.
- [316] J. Fonlupt and A. Nachef. Dynamic programming and the graphical traveling salesman problem. *J. Assoc. Comput. Mach.*, 40(5):1165–1187, 1993.
- [317] J. Fonlupt and D. Naddef. The traveling salesman problem in graphs with excluded minors. *Math. Program. Ser. A*, 53:147–172, 1992.
- [318] R. Fourer. Software survey: Linear programming. *OR/MS Today*, 26:64–71, 1999.
- [319] K.R. Fox. *Production scheduling on parallel lines with dependencies*. PhD thesis, John Hopkins University, 1973.
- [320] A. Frank and É. Tardos. An application of simultaneous approximation in combinatorial optimization. In *Annual Symposium on Foundations of Computer Science*, pages 459–463, New York, 1985. IEEE.
- [321] G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *J. Algorithms*, 15(1):29–60, 1993.
- [322] M.L. Fredman, D.S. Johnson, L.A. McGeoch, and G. Ostheimer. Data structures for traveling salesmen. *J. Algorithms*, 18:432–479, 1995.
- [323] A. Frick. TSPGA - an evolution program for the symmetric traveling salesman problem. In H.J. Zimmermann, editor, *EUFIT'98 - 6th European Congress on Intelligent Techniques and Soft Computing*, pages 513–517. Mainz Verlag, Aachen, 1998.
- [324] A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.
- [325] A. M. Frieze. Worst-case analysis of algorithms for travelling salesman problems. *Oper. Res.-Verf.*, 32:93–112, 1979.

- [326] A.M. Frieze. On the exact solution of random travelling salesman problems with medium-sized integer costs. *SIAM J. Comput.*, 16:1052–1072, 1987.
- [327] A.M. Frieze. Parallel algorithms for finding Hamilton cycles in random graphs. *Inf. Process. Lett.*, 25:111–117, 1987.
- [328] A.M. Frieze. An algorithm for finding Hamilton cycles in random digraphs. *J. Algorithms*, 9:181–204, 1988.
- [329] A.M. Frieze. Finding Hamilton cycles in sparse random graphs. *J. Comb. Theory, Ser. B*, 44:230–250, 1988.
- [330] A.M. Frieze. Partitioning random graphs into large cycles. *Discrete Math.*, 70:149–158, 1988.
- [331] A.M. Frieze. On matchings and Hamilton cycles in random graphs. In J. Siemons, editor, *Surveys in Combinatorics, Proc. 12th British Combinatorial Conference*, pages 84–114. London Math. Soc., 1989.
- [332] A.M. Frieze, M.R. Jerrum, M. Molloy, R.W. Robinson, and N.C. Wormald. Generating and counting Hamilton cycles in random regular graphs. *J. Algorithms*, 21:176–198, 1996.
- [333] A.M. Frieze, R.M. Karp, and B. Reed. When is the assignment bound asymptotically tight for the asymmetric traveling-salesman problem? *SIAM J. Comput.*, 24:484–493, 1995.
- [334] A.M. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Struct. Algorithms*, 10:5–42, 1997.
- [335] A.M. Frieze and B.D. McKay. Multicoloured trees in random graphs. *Random Struct. Algorithms*, 5:45–56, 1994.
- [336] A.M. Frieze and G. Sorkin. The probabilistic relationship between the assignment and asymmetric traveling salesman problems. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 652–660. ACM-SIAM, 2001.
- [337] A.M. Frieze and S. Suen. Counting Hamilton cycles in random directed graphs. *Random Struct. Algorithms*, 3:235–242, 1992.
- [338] D.R. Fulkerson. Packing rooted directed cuts in a weighted directed graph. *Math. Program., Ser. A*, 6:1–13, 1974.
- [339] S.H. Fuller. An optimal drum scheduling algorithm. *IEEE Trans. Comput.*, C-21(11):1153–1165, 1972.
- [340] E.Y. Gabovich. The small traveling salesman problem. *Trudy Vychisl. Centra Tartu. Gos. Univ.*, 19:17–51, 1970. (in Russian).
- [341] E.Y. Gabovich. Constant discrete programming problems on substitution sets. *Kibernetika*, no. 5:128–134, 1976.
- [342] H. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proc. 32th IEEE Symp. Found. Comp.*, pages 812–821, 1991.
- [343] H.N. Gabow and R.E. Tarjan. Algorithms for two bottleneck optimization problems. *J. Algorithms*, 9:411–417, 1988.
- [344] N.E. Gaikov. On the minimization of a linear forms on cycles. Manuscript deposited in VINITI (AISTI) (in Russian), 1980.

- [345] A. Garcia, P. Jodra, and J. Tejel. An efficient algorithm for on-line searching of minima in Monge path-decomposable tridimensional arrays. *Inform. Process. Lett.*, 68(1):3–9, 1998.
- [346] M.R. Garey, R.L. Graham, and D.S. Johnson. Some NP-complete geometric problems. In *Proc. 8th ACM Symp. Theory Comput.*, pages 10–22, 1976.
- [347] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman, San Francisco, CA, 1979.
- [348] R.S. Garfinkel. Minimizing wallpaper waste part I: a class of traveling salesman problems. *Oper. Res.*, 25:741–751, 1977.
- [349] R.S. Garfinkel and K.C. Gilbert. The bottleneck traveling salesman problem: Algorithms and probabilistic analysis. *J. Assoc. Comput. Mach.*, 25:435–448, 1978.
- [350] B. Gavish and S.C. Graves. The traveling salesman problem and related problems. Working Paper GR-078-78, Operations Research Center, MIT, 1978.
- [351] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res.*, 40:1086–1094, 1992.
- [352] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Manage. Sci.*, 40(10):1276–1290, 1994.
- [353] M. Gendreau, A. Hertz, and G. Laporte. The traveling salesman problem with backhauls. *Comput. Oper. Res.*, 23:501–508, 1996.
- [354] M. Gendreau, G. Laporte, and F. Semet. The covering tour problem. *Oper. Res.*, 45:568–576, 1997.
- [355] M. Gendreau, G. Laporte, and F. Semet. The selective traveling salesman problem. *Networks*, 32:263–273, 1998.
- [356] G.V. Gens and E.V. Levner. Effective approximate algorithms for combinatorial problems. In *TsEMI*. Moscow, 1981.
- [357] A.M.H. Gerards and A. Schrijver. Matrices with the Edmonds-Johnson property. *Combinatorica*, 6:365–379, 1986.
- [358] H.W. Ghosh. Expected travel among random points. *Bulletin Culcatta Stat. Assoc.*, 2:83–87, 1948.
- [359] J. Giesen. Curve reconstruction, the TSP and Menger’s Theorem on length. In *Proc. 15th Annual ACM Symposium on Computational Geometry*, pages 207–216, 1999.
- [360] P.C. Gilmore and R.E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Oper. Res.*, 12:655–679, 1964.
- [361] P.C. Gilmore, E.L. Lawler, and D.B. Shmoys. Well-solved special cases. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 87–143. Wiley, Chichester, 1985.
- [362] E.Kh. Gimadi. The traveling salesman problem on a maximum: conditions for the asymptotic accuracy of the algorithm “go to the most remote city”. *Upravlyaemye Sistemy*, no. 27:11–15, 1989. (in Russian).
- [363] E.Kh. Gimadi, N.I. Glebov, and A.I. Serdyukov. An approximation algorithm for the traveling salesman problem and its probabilistic analysis. *Diskretnui*

- Analiz i Issledovanie Operatsii*, 1(2):8–17, 1994. in Russian. Translated in *Operations Research and Discrete Analysis. Translation of Discret. Anal. Issled. Oper. 1 (1994), N 1-4*, Mathematics and its Applications, Kluwer, Dordrecht, 1996; 35-43.
- [364] E.Kh. Gimadi and N.K. Maksishko. Justification of conditions for the asymptotic exactness of an approximate algorithm for solving the traveling salesman problem on a maximum in the case of a discrete distribution. *Upravlyaemye Sistemy*, no. 30:25–29, 1990. (in Russian).
- [365] E.Kh. Gimadi and V.A. Perepelitsa. On the problem of finding the minimal hamiltonian circuit on a graph with weighted arcs. *Diskretnui Analiz*, no. 15:57–65, 1969. (in Russian).
- [366] E.Kh. Gimadi and V.A. Perepelitsa. An asymptotical approach to the solution of the traveling salesman problem. *Upravlyaemye Sistemy*, 12:35–45, 1974. (in Russian).
- [367] F. Glover. A multiphase-dual algorithm for zero-one integer programming problems. *Oper. Res.*, 13:879–919, 1965.
- [368] F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sci.*, 8(1):156–166, 1977.
- [369] F. Glover. Tabu search – Part I. *ORSA J. Comput.*, 1:190–206, 1989.
- [370] F. Glover. Tabu search – Part II. *ORSA J. Comput.*, 2:4–32, 1990.
- [371] F. Glover. Multilevel Tabu Search and Embedded Search Neighborhoods for the Traveling Salesman Problem. Technical report, University of Colorado, Boulder, 1991.
- [372] F. Glover. Ejection chains, reference structures, and alternating path algorithms for traveling salesman problem. University of Colorado-Boulder, April 1992.
- [373] F. Glover. New ejection chain and alternating path methods for traveling salesman problems. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research: New Developments in Their Interfaces*, pages 491–507. Pergamon, Oxford, 1992.
- [374] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Appl. Math.*, 65:223–253, 1996.
- [375] F. Glover. Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *J. Heuristics*, 2(2):169–179, 1996.
- [376] F. Glover. A template for Scatter Search and Path Relinking. In *Lect. Notes Comput. Sci.*, volume 1363, pages 13–54. Springer, Heidelberg, 1997.
- [377] F. Glover, G. Gutin, A. Yeo, and A. Zverovich. Construction heuristics for the asymmetric TSP. *Eur. J. Oper. Res.*, 129:555–568, 2001.
- [378] F. Glover and G. Kochenberger. *State-of-the-Art Handbook in Metaheuristics*. Kluwer, Boston, to appear.
- [379] F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.
- [380] F. Glover, M. Laguna, and R. Martí. Scatter Search. In *Theory and Applications of Evolutionary Computation: Recent Trends*. Springer-Verlag, Heidelberg, to appear.

- [381] F. Glover, A. Løkketangen, and D. Woodruff. Scatter Search to Generate Diverse MIP Solutions. Technical report, University of Colorado, Boulder, 1999.
- [382] F. Glover and A.P. Punnen. The travelling salesman problem: new solvable cases and linkages with the development of approximation algorithms. *J. Oper. Res. Soc.*, 48:502–510, 1997.
- [383] M. Goemans. Worst-case comparison of valid inequalities for the TSP. *Math. Program. Ser. A*, 69:335–349, 1995.
- [384] M. Goemans and D. Bertsimas. Probabilistic analysis of the Held-Karp relaxation for the Euclidean traveling salesman problem. *Math. Oper. Res.*, 16:72–89, 1991.
- [385] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Math. Program. Ser. A*, 82:111–124, 1998.
- [386] B.L. Golden, L. Levy, and R. Dahl. Two generalizations of the traveling salesman problem. *Omega*, 9:439–441, 1981.
- [387] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Nav. Res. Logist.*, 34:307–318, 1987.
- [388] B.L. Golden and W.R. Stewart. Empirical analysis of heuristics. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.
- [389] B.L. Golden, Q. Wang, and L. Liu. A multifaceted heuristic for the orienteering problem. *Nav. Res. Logist.*, 35:359–366, 1988.
- [390] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *J. Soc. Ind. Appl. Math.*, 9:551–570, 1961.
- [391] M. Goossens, F. Mittelbach, and A. Samarin. *The LATEX Companion*. Addison-Wesley, Reading, MA, 1998. <http://www.latex-project.org/>.
- [392] L. Gouveia and S. Voss. A classification of formulations for the (time-dependent) traveling salesman problem. *Eur. J. Oper. Res.*, 83:69–82, 1995.
- [393] R.L. Graham. Personal communication with S. Arora. 1996.
- [394] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.
- [395] M. Grigni, E. Koutsoupias, and C.H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th Symp. Found. Comput. Sci.*, pages 640–645, 1995.
- [396] M. Grötschel. *Polyedrische Charakterisierungen Kombinatorischer Optimierungsprobleme*. Hain, Maisenheim am Glen, 1977.
- [397] M. Grötschel. On the symmetric traveling salesman problem: solution of 120-city problem. *Math. Program. Studies*, 12:61–77, 1980.
- [398] M. Grötschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. *Math. Program., Ser. A*, 51:141–202, 1991.
- [399] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
- [400] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1988.

- [401] M. Grötschel and M.W. M. Padberg. On the symmetric traveling salesman problem: theory and computations. In R. Henn et al., editor, *Optimization and Operations Research*, Lect. Notes Econom. Math. Systems, Vol. 157, pages 105–115. Springer, Berlin, 1978.
- [402] M. Grötschel and M.W. Padberg. Lineare charakterisierungen von traveling salesman problemen. *Zeitschrift für Operations Research*, 21:33–64, 1977.
- [403] M. Grötschel and M.W. Padberg. On the symmetric traveling salesman problem I: inequalities. *Math. Program.*, 16:265–280, 1979.
- [404] M. Grötschel and M.W. Padberg. On the symmetric traveling salesman problem II: lifting theorems and facets. *Math. Program.*, 16:281–302, 1979.
- [405] M. Grötschel and M.W. Padberg. Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and Shmoys D.B., editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 251–305. Wiley, Chichester, 1985.
- [406] M. Grötschel and W. Pulleyblank. Clique tree inequalities and the Symmetric Traveling Salesman Problem. *Math. Oper. Res.*, 11:537–569, 1986.
- [407] M. Grötschel and Y. Wakabayashi. On the structure of the monotone asymmetric traveling salesman polytope, I: Hypohamiltonian facets. *Discrete Math.*, 34:43–59, 1981.
- [408] M. Grötschel and Y. Wakabayashi. On the structure of the monotone asymmetric traveling salesman polytope, II: Hypotractable facets. *Math. Program. Study*, 14:77–97, 1981.
- [409] J. Gu and X. Huang. Efficient Local Search with Search Space Smoothing: A Case Study of the Traveling Salesman Problem. *IEEE Trans. Systems Man Cybernet.*, 24(5):728–735, 1994.
- [410] S.K. Gupta and A.K. Mittal. A minmax problem as a linear programming problem. *Opsearch*, 19:49–53, 1982.
- [411] S.K. Gupta and A.P. Punnen. k -sum optimization problems. *Oper. Res. Lett.*, 9:121–126, 1990.
- [412] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16:486–502, 1987.
- [413] D. Gusfield, R. Karp, L. Wang, and P. Stelling. Graph traversals, genes and matroids: an efficient case of the traveling salesman problem. *Discrete Appl. Math.*, 88:167–180, 1998.
- [414] G. Gutin. On an approach to solving the traveling salesman problem. In *Proceedings of the USSR Conference on System Research*, pages 184–185. Nauka, Moscow, 1984. (in Russian).
- [415] G. Gutin. On the efficiency of a local algorithm for solving the travelling salesman problem. *Autom. Remote Control*, 49(11):1514–1519, 1988.
- [416] G. Gutin. Exponential neighbourhood local search for the travelling salesman problem. *Comput. Oper. Res.*, 26:313–320, 1999.
- [417] G. Gutin and A. Yeo. TSP heuristics with large domination number. Technical Report 12, Dept Math and Stats, Brunel University, 1998.

- [418] G. Gutin and A. Yeo. Small diameter neighbourhood graphs for the traveling salesman problem: at most four moves from tour to tour. *Comput. Oper. Res.*, 26:321–327, 1999.
- [419] G. Gutin and A. Yeo. Anti-matroids. Submitted, 2001.
- [420] G. Gutin and A. Yeo. TSP tour domination and Hamilton cycle decomposition of regular digraphs. *Oper. Res. Lett.*, 28:107–111, 2001.
- [421] G. Gutin and A. Yeo. Upper bounds on ATSP neighborhood size. Technical Report TR-01-01, Dept of Computer Science, Royal Holloway Univ. London, 2001.
- [422] G. Gutin and A. Yeo. Assignment Problem based algorithms are impractical for the Generalized TSP. *Australas. J. Combinatorics*, to appear, 2002.
- [423] G. Gutin and A. Yeo. Polynomial approximation algorithms for the TSP and the QAP with factorial domination number. *Discrete Appl. Math.*, to appear.
- [424] G. Gutin, A. Yeo, and A. Zverovich. Traveling salesman should not be greedy: domination analysis of TSP greedy-type heuristics. *Discrete Appl. Math.*, to appear.
- [425] G. Gutin and A. Zverovich. Evaluation of the Contract-or-Patch Heuristic for the Asymmetric TSP. Submitted.
- [426] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28:422–437, 2000.
- [427] H. Hamacher (ed.). Operations research software exchange program. *Eur. J. Oper. Res.*, 38:118–122, 1989.
- [428] R. Häggkvist. Series of lectures on Hamilton decomposition. British Combinatorial Conference, Swansea, UK, 1981, International Conference on Convexity and Graph Theory, Israel, March 1981, and Seminar Orsey, France, 1986.
- [429] R. Häggkvist. Personal communications, 1999–2000.
- [430] M.M. Halldórsson, K. Iwano, N. Katoh, and T. Tokuyama. Finding subsets maximizing minimum structures. Research Report, Science Institute, University of Iceland, 1995.
- [431] J.H. Halton. The shoelace problem. *Math. Intell.*, 17:36–39, 1995.
- [432] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *Eur. J. Oper. Res.*, 130:449–467, 1994.
- [433] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston, 1999.
- [434] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, New York, 1979.
- [435] D. Hartvigsen. *Extensions of Matching Theory*. PhD thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA, USA, 1984.
- [436] D. Hartvigsen. The square-free 2-factor problem in bipartite graphs. In *Proc. IPCO VII, Lect. Notes Comput. Sci.*, volume 1610, pages 234–241. Springer-Verlag, Berlin, 1999.

- [437] D. Hartvigsen and W. Pulleyblank. Outer-facial graphs and the traveling salesman problem. *SIAM J. Optimization*, 1:676–689, 1994.
- [438] R. Hassin and S. Rubinstein. An approximation algorithm for the maximum traveling salesman problem. *Inform. Process. Lett.*, 67:125–130, 1998.
- [439] R. Hassin and S. Rubinstein. Better approximations for max TSP. *Inf. Process. Lett.*, 75:181–186, 2000.
- [440] R. Hassin and S. Rubinstein. A $7/8$ -approximation algorithm for metric max TSP. In *Proc. 7th Workshop Algorithms Data Struct. (WADS 2001)*. Springer-Verlag, 2001. To appear in *Lect. Notes Comput. Sci.*
- [441] X. He and Z.Z. Chen. Shortest path in complete bipartite digraph problem and its applications. In *Proc. 8th Ann. ACM-SIAM Symp. Discrete Algorithms Jan 5-7 1997*, New Orleans, LA, 1997.
- [442] K.H. Helbig-Hansen and J. Krarup. Improvements of the Held-Karp algorithm for the symmetric traveling salesman problem. *Math. Program.*, 7:87–96, 1974.
- [443] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Ind. Appl. Math.*, 10:196–210, 1962.
- [444] M. Held and R.M. Karp. The traveling salesman problem and minimal spanning trees. *Oper. Res.*, 18:1138–1162, 1970.
- [445] M. Held and R.M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees: Part II. *Math. Program.*, 1:6–25, 1971.
- [446] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.*, 12:106–130, 2000. Source code currently available from the author's <http://www.dat.ruc.dk/~keld/>.
- [447] C.S. Helvig, G. Robins, and A. Zelikovsky. Moving-target TSP and related problems. In *Lect. Notes Comput. Sci., Vol. 1461*, pages 453–464. Springer-Verlag, Berlin, 1998.
- [448] A.M. Hobbs. Powers of graphs, line graphs, and total graphs. In *Lecture Notes in Mathematics*, volume 642, pages 271–285. Springer, 1978.
- [449] D.S. Hochbaum and D.B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. Assoc. Comput. Mach.*, 33:533–550, 1986.
- [450] A.J. Hoffman. On simple linear programming problems. In *Convexity, Proc. Symposia Pure Math., AMS, Providence, RI*, pages 317–327, 1961.
- [451] A.J. Hoffman and P. Wolfe. History. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.
- [452] I. Hong, A. B. Kahng, and B. Moon. Improved large-step Markov chain variants for the symmetric TSP. *J. Heuristics*, 3:63–81, 1997.
- [453] J.E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2:135–158, 1973.
- [454] L.J. Hubert and F.B. Baker. Applications of combinatorial programming to data analysis: the traveling salesman and related problem. *Psychometrika*, 43:81–91, 1978.
- [455] C. Hurkens. Nasty TSP instances for farthest insertion. In *Integer Programming and Combinatorial Optimization*, pages 346–352. Math. Programm. Soc., 1992.

- [456] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.
- [457] M.R. Jerrum and A.J. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18:1149–1178, 1989.
- [458] R.J. Jessen. Statistical investigation of a sample farm survey. Research Bulletin 304, Iowa State College of Agriculture, 1942.
- [459] D.B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. Assoc. Comput. Mach.*, 24:1–13, 1977.
- [460] D.S. Johnson. Local optimization and the traveling salesman problem. In *Proc. 17th Colloq. Automata Languages Program.*, pages 446–461, Berlin, 1990. Lect. Notes Comput. Sci., Vol.443, Springer-Verlag.
- [461] D.S. Johnson, J.L. Bentley, L.A. McGeoch, and E.E. Rothberg. Near-optimal solutions to very large traveling salesman problems. Monograph, in preparation.
- [462] D.S. Johnson, L. McGeoch, F. Glover, and C. Rego. www.research.att.com/~dsj/chtsp/.
- [463] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, Chichester, 1997.
- [464] D.S. Johnson, L.A. McGeoch, F. Glover, and C. Rego. Website for the DIMACS Implementation Challenge on the Traveling Salesman Problem: <http://www.research.att.com/~dsj/chtsp/>.
- [465] D.S. Johnson, L.A. McGeoch, and E.E. Rothberg. Asymptotic experimental analysis for the Held-Karp traveling salesman bound. In *Proc. 7th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 341–350, 1996.
- [466] D.S. Johnson and C.H. Papadimitriou. Computational complexity. In E.L. Lawler, J.K. Lenstra, A.G.H. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. Wiley, Chichester, 1985.
- [467] S.M. Johnson. Optimal two- and three-stage production schedules with set-up times included. *Naval Res. Logist. Quart.*, 1:61–68, 1954.
- [468] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. *Contemp. Math.*, 26:189–206, 1984.
- [469] K. Jongens and T. Volgenant. The symmetric clustered traveling salesman problem. *Eur. J. Oper. Res.*, 19:68–75, 1985.
- [470] R. Jonker and T. Volgenant. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *Eur. J. Oper. Res.*, 9:83–89, 1982.
- [471] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Oper. Res. Lett.*, 2:161–163, 1983.
- [472] C. Jorgensen and S. Powell. Solving 0-1 minmax problems. *J. Oper. Res. Soc.*, 38:515–522, 1987.
- [473] M. Jünger, T. Christof, and G. Reinelt. A complete description of the traveling salesman polytope on 8 nodes. *Oper. Res. Lett.*, 10:497–500, 1991.
- [474] M. Jünger, G. Reinelt, and G. Rinaldi. The Traveling Salesman Problem. In M. Ball, T. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network*

- Models*, volume 7 of *Handbook on Operations Research and Management Sciences*, pages 225–330. North Holland, Amsterdam, 1995.
- [475] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. *Z. Oper. Res.*, 40:183–217, 1994.
- [476] M. Jünger, G. Rinaldi, and S. Thienel. Practical performance of efficient minimum cut algorithms. *Algorithmica*, 26:172–195, 2000.
- [477] M. Jünger and S. Thienel. Introduction to ABACUS—A Branch-And-CUT System. *Oper. Res. Lett.*, 22:83–95, 1998.
- [478] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software - Practice and Experience*, 30:1325–1352, 2000.
- [479] S.N. Kabadi. A solvable case of traveling salesman problem. Presented at the ORSA Conference, November 1985, Atlanta.
- [480] S.N. Kabadi. Computational complexity of some problems in graph theory. Working paper, Faculty of Administration, University of New Brunswick, 1990.
- [481] S.N. Kabadi. A general sufficiency condition for solvability of the Traveling Salesman Problem by Gilmore-Gomory patching. Working paper, Faculty of Administration, University of New Brunswick, 2001.
- [482] S.N. Kabadi. Generalization of the Gilmore-Gomory scheme and new polynomially solvable classes of the Traveling Salesman Problem. *Discrete Appl. Math.*, to appear, 2001.
- [483] S.N. Kabadi and M.F. Baki. Gilmore-Gomory type traveling salesman problems. *Comput. Oper. Res.*, 26:329–351, 1999.
- [484] S.N. Kabadi and R. Chandrasekaran. The Gilmore-Gomory Traveling Salesman Problem. Invited talk in honor of R. Gomory, President IBM Research Center, ORSA Conference, October 1989, New York, 1989.
- [485] S.N. Kabadi and R. Chandrasekaran. Generalized traveling salesman problem on a tree. Working paper 90-027, Faculty of Administration, University of New Brunswick, 1990.
- [486] S.N. Kabadi and R. Chandrasekaran. Solvable classes of generalized traveling salesman problem. *DIMACS Series Discrete Math. Theor. Comput. Sci.*, 1:49–59, 1990.
- [487] S.N. Kabadi and R. Chandrasekaran. Tree traveling salesman problem (extended abstract). In *Proc. 2nd Canad. Confer. Comput. Geometry, J.J. Urrutia (ed.)*, pages 367–371, 1990.
- [488] S.N. Kabadi and A.P. Punnen. The traveling salesman problem with two distinct tour values. Working paper 95-026, Faculty of Administration, University of New Brunswick, 1995.
- [489] S.N. Kabadi and A.P. Punnen. Weighted graphs with Hamiltonian cycles of same length. Working paper 96-021, Faculty of Administration, University of New Brunswick, 1996.
- [490] S.N. Kabadi and A.P. Punnen. New solvable cases of generalized graphical traveling salesman problem. Under preparation, 2001.
- [491] K. Kalmanson. Edgeconvex circuits and the traveling salesman problem. *Canad. J. Math.*, 27(5):1000–1010, 1975.

- [492] P.C. Kanellakis and C.H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Oper. Res.*, 28(5):1086–1099, 1980.
- [493] V. Kantabutra. Traveling salesman cycles are not always subgraphs of Voronoi duals. *Inform. Process. Lett.*, 16:11–12, 1983.
- [494] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
- [495] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
- [496] R.M. Karp. The probabilistic analysis of some combinatorial search algorithms. In J.F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 1 – 19. Academic Press, New York, 1976.
- [497] R.M. Karp. Probabilistic analysis of partitioning algorithms for the traveling-salesman in the plane. *Math. Oper. Res.*, 2:209–224, 1977.
- [498] R.M. Karp. A patching algorithm for the non-symmetric traveling salesman problem. *SIAM J. Comput.*, 8:561–573, 1979.
- [499] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th ACM Ann. Symp. Theory Comput. (STOC'80)*, pages 302–309. L.A., California, 1980.
- [500] R.M. Karp and J.M. Steele. Probabilistic analysis of heuristics. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.
- [501] S. Kataoka and S. Morito. Selection of relaxation problems for a class of asymmetric traveling salesman problem instances. *J. Oper. Res. Soc. Japan*, 34:233–249, 1991.
- [502] J. Kelly, B. Rangaswamy, and J. Xu. A Scatter Search-based learning algorithm for neural network training. *Journal of Heuristics*, 2:129–146, 1996.
- [503] H. Kesten and S. Lee. The central limit theorem for weighted minimal spanning trees on random points. *Ann. Appl. Prob.*, 6:495–527, 1996.
- [504] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. Technical Report TR95-023, ECCC, 1995.
- [505] G.A.P. Kindervater, A. Volgenant, G. De Leve, and V. van Gijlswijk. On dual solutions of the linear assignment problem. *Eur. J. Oper. Res.*, 19:76–81, 1985.
- [506] P.S. Klyaus. Generation of test problems for the traveling salesman problem. Preprint 16, Inst. Mat. Akad. Navuk BSSR, Minsk, USSR, 1976. (in Russian).
- [507] P.S. Klyaus. The structure of the optimal solution of certain classes of travelling salesman problem. *Vestsi Akad. Navuk BSSR, Ser. Fiz. -Mat. Navuk*, no. 6:95–98, 1976. (in Russian).
- [508] D.E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms (2nd Edition)*. Addison-Wesley, Reading, MA, 1981.
- [509] D.E. Knuth and S. Levy. *The CWEB System of Structured Documentation*. Addison-Wesley, Reading, MA, 1993. <http://www-cs-faculty.stanford.edu/~knuth/cweb.html>.

- [510] V.F. Kolchin. *Random Mappings*. Nauka, Moscow, 1984. (in Russian). Translated in Translation Series in Mathematics and Engineering, Optimization Software, Inc., Publications Division, New York, 1986.
- [511] J.G. Kollias, Y. Manolopoulos, and C.H. Papadimitriou. The optimum execution order of queries in linear storage. *Inform. Proc. Lett.*, 36:141–145, 1990.
- [512] J. Komlós and E. Szemerédi. Limit distributions for the existence of Hamilton circuits in a random graph. *Discrete Math.*, 43:55–63, 1983.
- [513] C. Korostensky and G.H. Gonnet. Near optimal multiple sequence alignments using a TSP approach. *SPIRE*, 99:105–114, 1999.
- [514] C. Korostensky and G.H. Gonnet. Using the TSP algorithms for evolutionary tree constructions. *Bioinformatics*, 16:619–627, 2000.
- [515] S.R. Kosaraju, J.K. Park, and C. Stein. Long tours and short superstrings. In *Proc. 35th Ann. IEEE Symp. Found. Comput. Sci. (FOCS'94)*, pages 166–177. IEEE, 1994.
- [516] E.G. Kosman. Structuring of matrices. *Soviet Math. Doklady*, 41:152–155, 1990.
- [517] A.V. Kostochka and A.I. Serdyukov. Polynomial algorithms with the estimates $3/4$ and $5/6$ for the traveling salesman problem of the maximum. *Upravlyaemye Sistemy*, no. 26:55–59, 1985. (in Russian).
- [518] U. Kousgaard. Personal communication, 2000.
- [519] E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Lect. Notes Comput. Sci., Vol. 1099*, pages 280–289. Springer-Verlag, Berlin, 1996.
- [520] M.M. Kovalev and V.M. Kotov. Analysis of algorithms for the construction of a hamiltonian cycle of maximum weight. *Vestsi Akad. Navuk BSSR, Ser. Fiz-Mat. Navuk*, no. 4:45–50, 1984. (in Russian).
- [521] M.M. Kovalev and V.M. Kotov. Estimate of the error of series of approximate algorithms. *Vestnik Belorusskogo Gosudarstvennogo Universiteta, Seriya I Fizika, Matematika, Mekhanika*, no. 3:44–48, 1986. (in Russian).
- [522] M. Krivelevich, B. Sudakov, V. Vu, and N. Wormald. Random regular graphs of high degree. *Random Struct. Algorithms*, to appear.
- [523] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–50, 1956.
- [524] S. Krynski. Graphs in which all Hamiltonian cycles have the same length. *Discrete Appl. Math.*, 55:87–89, 1994.
- [525] D.A. Kumar and C.P. Rangan. Approximation algorithms for the traveling salesman problem with range condition. *Theor. Inform. Appl.*, 34:173–181, 2000.
- [526] N.N. Kuzjurin. Asymptotic investigation on the problem of covering. *Problemy Kybernetiky*, 37:19–56, 1980. (in Russian).
- [527] M. Labbé, G. Laporte, I. Rodríguez, and J.J. Salazar. The median cycle problem. Working paper Université Libre de Bruxelles, 1999.
- [528] M. Laguna, J.P. Kelly, J.L. Gonzalez-Valarde, and F. Glover. Tabu Search for multilevel generalized assignment problems. *Eur. J. Oper. Res.*, 82:176–189, 1995.

- [529] M. Laguna, H. Lourenço, and R. Martí. Assigning proctors to exams with Scatter Search. In *Computing Tools for Modeling: Optimization and Simulation*, Interfaces in Computer Science and Operations Research, pages 215–227. Kluwer, Boston, 2000.
- [530] M. Laguna and R. Martí. Scatter Search for the Linear Ordering Problem. In *New Ideas in Optimization*, pages 331–339. McGraw Hill, Berkshire, 1999.
- [531] A. Langevin. *Planification de tournées de véhicules*. PhD thesis, Department of Applied Mathematics, École Polytechnique, Montreal, 1988.
- [532] A. Langevin, F. Soumis, and J. Desrosiers. Classification of traveling salesman problem formulations. *Oper. Res. Lett.*, 9:127–132, 1990.
- [533] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *Eur. J. Oper. Res.*, 59:231–237, 1992.
- [534] G. Laporte. Modeling and solving several classes of arc routing problems as traveling salesman problems. *Comput. Oper. Res.*, 24:1057–1061, 1997.
- [535] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized traveling salesman problem. *J. Oper. Res. Soc.*, 47:1461–1467, 1996.
- [536] G. Laporte and S. Martello. The selective traveling salesman problem. *Discrete Appl. Math.*, 26:193–207, 1990.
- [537] G. Laporte, H. Mercure, and Y. Nobert. Generalized Travelling Salesman Problem through n sets of nodes: the asymmetrical cases. *Discrete Appl. Math.*, 18:185–197, 1987.
- [538] G. Laporte and Y. Nobert. Generalized Travelling Salesman through n sets of nodes: an integer programming approach. *INFOR, Can. J. Oper. Res. Inf. Process.*, 21:61–75, 1983.
- [539] G. Laporte and I.H. Osman. *Routing Problems: A Bibliography*, volume 61. Annals of Operations Research, 1995.
- [540] G. Laporte, J. Riera, and J.J. Salazar. A branch-and-cut algorithm for the undirected travelling purchaser problem. Working paper, Université de Montreal, 2000.
- [541] G. Laporte and F. Semet. Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR, Can. J. Oper. Res. Inf. Process.*, 37:114–120, 1999.
- [542] H.T. Lau. *Finding Hamiltonian cycle in the square of a block*. PhD thesis, McGill University, 1980.
- [543] H.T. Lau. Finding EPS-graphs. *Monatsh. Math.*, 92:37–40, 1981.
- [544] H.T. Lau. *Combinatorial heuristic algorithms with FORTRAN*. Lect. Notes Econom. Math. Systems, Vol. 280. Springer Verlag, Berlin, 1986.
- [545] E.L. Lawler. The quadratic assignment problem. *Manage. Sci.*, 9:586–599, 1963.
- [546] E.L. Lawler. A solvable case of the traveling salesman problem. *Math. Program.*, 1:267–269, 1971.
- [547] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

- [548] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.
- [549] A.I. Lazebnik and I.L. Hranovic. A solution of the generalized traveling salesman problem by the branch and bound method. *Ekonom. Mat. Metody*, 9:363–364, 1973. (in Russian).
- [550] S. Lee. The central limit theorem for Euclidean minimal spanning trees I. *Ann. Appl. Prob.*, 6:996–1020, 1997.
- [551] S. Lee. Asymptotics of power-weighted Euclidean functionals. *Stoch. Proc. Appl.*, 79:109–116, 1999.
- [552] S. Lee. Worst case asymptotics of power-weighted Euclidean functionals. Preprint, 1999.
- [553] A.C. Leifer and M.B. Rosenwein. Strong linear programming relaxations for the orienteering problem. *Eur. J. Oper. Res.*, 73:517–523, 1994.
- [554] H.W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8, 1983.
- [555] J.K. Lenstra and A.H.G. Rinnooy Kan. Some simple applications of the traveling salesman problem. *Oper. Res. Quart.*, 26:717–733, 1975.
- [556] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.
- [557] V.K. Leont'ev. Investigation of an algorithm for solving the travelling salesman problem. *Zh. Vychisl. Mat. Mat. Fiz.*, no. 5, 1973. (in Russian).
- [558] V.K. Leont'ev. Stability of the traveling salesman problem. *Zh. Vychisl. Mat. Mat. Fiz.*, no. 5, 1975. (in Russian).
- [559] A. Letchford. Separating a superclass of comb inequalities in planar graphs. *Math. Oper. Res.*, 25:443–454, 2000.
- [560] L.A. Levin. Universal sorting problems. *Problems Informat. Transmission*, 9:265–266, 1973.
- [561] Y. Lien, E. Ma, and B.W.S. Wah. Transformation of the generalized traveling salesman problem into the standard traveling salesman problem. *Inf. Sci.*, 74:177–189, 1993.
- [562] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Tech. J.*, 44:2245–2269, 1965.
- [563] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.*, 21:972–989, 1973.
- [564] J.D. Litke. An improved solution to the traveling salesman problem with thousands of nodes. *Commun. ACM*, 27(12):1227–1236, 1984.
- [565] J.D.C Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for traveling salesman problem. *Oper. Res.*, 11:972–989, 1963.
- [566] A. Löbel. Vehicle scheduling in public transit and Lagrangean pricing. *Manage. Sci.*, 44:1637–1649, 1998.
- [567] R.J. Loulou. On multicommodity flow formulation for the TSP. Working paper, McGill University, Montreal, 1988.

- [568] L. Lovász. Matroid matching and some applications. *J. Comb. Theory Ser. B*, 28:208–236, 1980.
- [569] L. Lovász and M.D. Plummer. On a family of planar bicritical graphs. *Proc. London Math. Soc.*, (3)30:160–176, 1975.
- [570] L. Lovász and M.D. Plummer. *Matching Theory*. North-Holland, Amsterdam, 1986.
- [571] X. Lu. A Chvátal-Erdős type condition for Hamiltonian graphs. *J. Graph Theory*, 18:791–800, 1994.
- [572] A. Lucena. The time-dependent traveling salesman problem - The deliveryman case. *Networks*, 20:753–763, 1990.
- [573] T. Łuczak. Cycles in random graphs. *Discrete Math.*, 98:231–236, 1991.
- [574] P.D. MacKenzie and Q.F. Stout. Optimal parallel construction of Hamiltonian cycles and spanning trees in random graphs. In *Proc. 5th ACM Symp. Parallel Algorithms Architectures*, pages 224–229. Springer, 1993.
- [575] P.C. Mahalanobis. A sample survey of the acreage under jute in Bengal. *Sankhya*, 4:511–530, 1940.
- [576] K.-T. Mak and A.J. Morton. A modified Lin-Kernighan traveling-salesman heuristic. *Oper. Res. Lett.*, 13:127–132, 1993.
- [577] V. Mak and N. Boland. Heuristic approaches to the asymmetric traveling salesman problem with replenishment arcs. *Internat. Trans. Oper. Res.*, 7:431–447, 2000.
- [578] G.S. Manku. A linear time algorithm for the bottleneck biconnected spanning subgraph problem. *Inform. Process. Lett.*, 59:1–7, 1996.
- [579] Y. Manoussakis. Directed Hamiltonian graphs. *J. Graph Theory*, 16(1):51–59, 1992.
- [580] H. Marchand, O. Boivineau, and S. Lafortune. On the synthesis of optimal schedulers in discrete event control problems with multiple goals. *SIAM J. Control Optimization*, 39:512–532, 2000.
- [581] O. Marcotte and S. Suri. Fast matching algorithms for points on a polygon. *SIAM J. Comput.*, 20(3):405–422, 1991.
- [582] F. Margot, 2001. Private communication with D. Naddef.
- [583] E.S. Marks. A lower bound for the expected travel among m random points. *Ann. Math. Stat.*, 19:419–422, 1948.
- [584] K. Marriott and P.J. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, MA, 1998.
- [585] S. Martello. An enumerative algorithm for finding Hamiltonian circuits in a directed graph. *ACM Trans. Math. Softw.*, 9:131–138, 1983.
- [586] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, 1990.
- [587] O. Martin, S.W. Otto, and E.W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Syst.*, 5:299–326, 1991.
- [588] O. Martin, S.W. Otto, and E.W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Oper. Res. Lett.*, 11:219–224, 1992.

- [589] C. McDiarmid. Clutter percolation and random graphs. *Math. Program. Study*, 13:17–25, 1980.
- [590] I.I. Melamed, S.I. Sergeev, and I.Kh. Sigal. The traveling salesman problem - I. Theoretical issues. *Automat. Remote Control*, pages 1147–1173, 1989.
- [591] I.I. Melamed, S.I. Sergeev, and I.Kh. Sigal. The traveling salesman problem. Exact Algorithms. *Avtomat. Telemekh.*, 1989. Russian.
- [592] I.I. Melamed, S.I. Sergeev, and I.Kh. Sigal. The traveling salesman problem. Approximation Algorithms. *Automat. Remote Control*, pages 1459–1479, 1990.
- [593] K. Menger. Das botenproblem. *Ergebnisse Eines Mathematischen Kolloquiums*, 2:11–12, 1932.
- [594] M. Michalski. On a class of polynomially solvable travelling salesman problems. *Zastosowania Matematyki*, 19:531–539, 1987.
- [595] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems. *J. Assoc. Comput. Mach.*, 7:326–329, 1960.
- [596] D.L. Miller and J.F. Pekny. Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem. *Oper. Res. Lett.*, 8:129–135, 1989.
- [597] D.L. Miller and J.F. Pekny. Exact solution of large asymmetric traveling salesman problems. *Science*, 251:754–761, 1991.
- [598] D.L. Miller and J.F. Pekny. A staged primal-dual algorithm for perfect b-matching with edge capacities. *ORSA J. Comput.*, 7:298–320, 1995.
- [599] E. Minieka. The delivery man problem on a tree network. *Ann. Oper. Res.*, 18:261–266, 1989.
- [600] M. Misiurewicz. Lacing irregular shoes. *Math. Intell.*, 18:32–34, 1996.
- [601] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k -MST, and related problems. *SIAM J. Comput.*, 28:1298–1309, 1999.
- [602] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Comput. Oper. Res.*, 24:1097–1100, 1997.
- [603] G. Monge. Memoires sur la theorie des deblais et des remblais. In *Histoire de l'Academie Royale des Science, Annee M. DCCLXXXI, avec les Memoires de Mathematique et de Physique, pur la meme Annee, Tires des Registres de cette Academie, Paris*, pages 666–704, 1781.
- [604] C.L. Monma, B.S. Munson, and W.R. Pulleyblank. Minimum-weight two-connected spanning networks. *Math. Prog., Ser. A*, 46:153–171, 1990.
- [605] J.W. Moon. *Topics on Tournaments*. Holt, Rinehart and Winston, New York, 1968.
- [606] B. M. Moret, D. A. Bader, and T. Warnow. High-performance algorithmic engineering for computational phylogenetics. In *Proc. 2001 Int'l Conf. Comput. Sci. (ICCS 2001), San Francisco*. Lect. Notes Comput. Sci., Vol. 2073–2074, Springer-Verlag, Berlin, 2001.
- [607] B.M. Moret, S. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp.*

- Biocomputing (PSB 2001)*, Hawaii, pages 583–594. World Scientific Pub., Singapore, 2001.
- [608] G. Morton and A.H. Land. A contribution to the ‘Travelling-Salesman’ problem. *J. R. Stat. Soc., Ser. B*, 17:185–203, 1955.
- [609] T.E. Morton and D.W. Pentico. *Heuristic Scheduling Systems*. Wiley, Chichester, 1993.
- [610] K.G. Murty. An algorithm for ranking all the assignments in increasing order of cost. *Oper. Res.*, 16:682–687, 1968.
- [611] K.G. Murty. *Linear Programming*. Wiley, New York, 1986.
- [612] K.G. Murty. *Network Programming*. Prentice Hall, New York, 1992.
- [613] D. Naddef. The binested inequalities of the symmetric traveling salesman polytope. *Math. Oper. Res.*, 17:882–900, 1992.
- [614] D. Naddef. On the domino inequalities for the Symmetric Traveling Salesman Polytope. Technical report, Laboratoire ID-IMAG, 2001. To appear in *The Sharpest Cuts*, MPS/SIAM Series in Optimization.
- [615] D. Naddef and Y. Pochet. The traveling salesman polytope revisited. *Math. Oper. Res.*, to appear.
- [616] D. Naddef and G. Rinaldi. The symmetric traveling salesman polytope and its graphical relaxation: Composition of valid inequalities. *Math. Program. Ser. A*, 51:359–400, 1991.
- [617] D. Naddef and G. Rinaldi. The crown inequalities for the traveling salesman polytope. *Math. Oper. Res.*, 17:308–326, 1992.
- [618] D. Naddef and G. Rinaldi. The graphical relaxation: a new framework for the symmetric traveling salesman polytope. *Math. Program. Ser. A*, 58:53–88, 1993.
- [619] D. Naddef and G. Rinaldi. The symmetric traveling salesman polytope: New facets from the graphical relaxation. Technical report, Laboratoire ID-IMAG, 2000.
- [620] D. Naddef and S. Thienel. Efficient separation routines for the symmetric traveling salesman problem I: general tools and comb separation. Technical report, LMC-IMAG Grenoble, 1999. To appear in *Math. Program. Ser. A*.
- [621] D. Naddef and S. Thienel. Efficient separation routines for the symmetric traveling salesman problem II: separating multi handle inequalities. Technical report, LMC-IMAG Grenoble, 1999. To appear in *Math. Program. Ser. A*.
- [622] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multi-graphs and capacitated graphs. *SIAM J. Discrete Math.*, 5:54–66, 1992.
- [623] H. Nagamochi, T. Ono, and T. Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Math. Program. Ser. A*, 67:325–341, 1994.
- [624] G.L. Nemhauser and L.E. Trotter. Properties of vertex packing and independent systems polyhedra. *Math. Program.*, 6:48–61, 1974.
- [625] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, Chichester, 1988.
- [626] D. Neto. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. PhD thesis, Department of Computer Science, University of Toronto, 1999. Source code currently available at <http://www.cs.toronto.edu/~neto/research/lk/>.

- [627] A. Nijenhuis and H. Wilf. *Combinatorial Algorithms*. Acad. Press, New York, 1975.
- [628] A. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms for Computers and Calculators*. Academic Press, N.Y., 2 edition, 1978.
- [629] C.E. Noon. *The Generalized Traveling Salesman Problem*. PhD thesis, Dept Ind. Oper. Res., University of Tennessee, 1988.
- [630] C.E. Noon and J.C. Bean. A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Oper. Res.*, 39:623–632, 1991.
- [631] C.E. Noon and J.C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31:39–44, 1993.
- [632] H.L. Ong. Approximate algorithms for the travelling purchaser problem. *Oper. Res. Lett.*, 1:201–205, 1982.
- [633] H.L. Ong and J.B. Moore. Worst-case analysis of two travelling salesman heuristics. *Oper. Res. Lett.*, 2:273–277, 1984.
- [634] M. Oosten. A note on sequential lifting procedures. Research Memorandum M94-08, University of Limburg, 1994.
- [635] I. Or. *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1976.
- [636] O. Ore. Note on Hamiltonian circuits. *Am. Math. Mon.*, 67:55, 1960.
- [637] A.J. Orman and H.P. Williams. A survey of different integer programming formulations of the travelling salesman problem. Preprint, University of Southampton, 1999.
- [638] P.S. Ow and T.E. Morton. Filtered Beam Search in Scheduling. *Internat. J. Product. Res.*, 26(1):35–62, 1988.
- [639] C.O. Ozgur and J.R. Brown. A two-stage traveling salesman procedure for the single machine sequence dependent scheduling problem. *Omega*, 23:205–219, 1995.
- [640] M.W. Padberg. On the facial structure of the set packing polyhedra. *Math. Program.*, 5:199–216, 1973.
- [641] M.W. Padberg. A note on zero-one programming. *Oper. Res.*, 23:833–837, 1975.
- [642] M.W. Padberg and M. Grötschel. Polyhedral computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.
- [643] M.W. Padberg and S. Hong. On the symmetric traveling salesman problem: A computational study. *Math. Program. Study*, 12:78–107, 1980.
- [644] M.W. Padberg and M.R. Rao. Odd minimum cut-sets and b-matching. *Math. Oper. Res.*, 7:67–80, 1982.
- [645] M.W. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch-and-cut. *Oper. Res. Lett.*, 6:1–7, 1987.

- [646] M.W. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Math. Program., Ser. A*, 47:19–36, 1990.
- [647] M.W. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Math. Program., Ser. A*, 47:219–257, 1990.
- [648] M.W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.
- [649] M.W. Padberg and T.Y. Sung. A polynomial-time solution to Papadimitriou and Steiglitz's 'traps'. *Oper. Res. Lett.*, 7:117–125, 1988.
- [650] M.W. Padberg and T.Y. Sung. An analytical comparison of different formulations of the traveling salesman problem. *Math. Program. Ser. A*, 52:315–357, 1991.
- [651] C.H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoret. Comput. Sci.*, 4:237–244, 1977.
- [652] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.
- [653] C.H. Papadimitriou and P.C. Kanellakis. Flowshop scheduling with limited temporary storage. *J. Assoc. Comput. Mach.*, 27:533–549, 1980.
- [654] C.H. Papadimitriou and K. Steiglitz. Some examples of difficult traveling salesman problems. *Oper. Res.*, 26(3):434–443, 1978.
- [655] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [656] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, New York, 1998.
- [657] C.H. Papadimitriou and U.V. Vazirani. On two geometric problems related to the travelling salesman problem. *J. Algorithms*, 5:231–246, 1984.
- [658] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. Comput. System Sci.*, 43:425–440, 1991.
- [659] C.H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18:1–11, 1993.
- [660] J.K. Park. A special case of the n -vertex traveling salesman problem that can be solved in $O(n)$ time. *Inform. Process. Lett.*, 40:247–254, 1991.
- [661] R.G. Parker and R.L. Rardin. Guaranteed performance heuristics for the bottleneck traveling salesperson problem. *Oper. Res. Lett.*, 12:269–272, 1982.
- [662] W.L. Pearn and R.C. Chien. Improved solutions for the traveling purchaser problem. *Comput. Oper. Res.*, 25:879–885, 1998.
- [663] J.F. Pekny and D.L. Miller. An exact parallel algorithm for the resource constrained traveling salesman problem with application to scheduling with an aggregate deadline. Research Report EDRC-05-44-89, Institute of Complex Engineered Systems, Carnegie Mellon University, 1989.
- [664] J.F. Pekny and D.L. Miller. A parallel branch-and-bound algorithm for solving large asymmetric traveling salesman problems. *Math. Prog., Ser. A*, 55:17–33, 1992.

- [665] J.F. Pekny, D.L. Miller, and D. Stodolsky. A note on exploiting the Hamiltonian cycle problem substructure of the Asymmetric Traveling Salesman Problem. *Oper. Res. Lett.*, 10:173–176, 1991.
- [666] K. Penavic. Optimal firing sequences for CAT scans. Technical report, Dept. Appl. Math., SUNY Stony Brook, 1994.
- [667] E. Pesch and F. Glover. TSP ejection chains. *Discrete Appl. Math.*, 76:165–181, 1997.
- [668] J.M. Phillips, A.P. Punnen, and S.N. Kabadi. A linear time algorithm for the bottleneck travelling salesman problem on a Halin graph. *Inform. Process. Lett.*, 67:105–110, 1998.
- [669] J.C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one machine scheduling. *Oper. Res.*, 26:86–110, 1978.
- [670] R.D. Plante. The nozzle guide vane problem. *Oper. Res.*, 36:18–33, 1988.
- [671] L. K. Platzmann and J. J. Bartholdi, III. Spacefilling curves and the planar travelling salesman problem. *J. Assoc. Comput. Mach.*, 36:719–737, 1989.
- [672] L. Pósa. Hamiltonian circuits in random graphs. *Discrete Math.*, 14:359–364, 1976.
- [673] F.P. Preparata and M.I. Shamos. *Computational Geometry - An Introduction*. Springer, New York, 1985.
- [674] H.N. Psaraftis. A dynamic programming solution to the single-vehicle many-to-many immediate request dial-a-ride problem. *Transport. Sci.*, 14:130–154, 1980.
- [675] H.N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Oper. Res.*, 28(6):1347–1359, 1980.
- [676] W.R. Pulleyblank. Polyhedral Combinatorics. In *Handbook in OR and MS*, volume 1. North-Holland, Amsterdam, 1989.
- [677] A.P. Punnen. A linear time algorithm for the bottleneck strongly connected spanning subgraph problem. Research report, IIT Kanpur, India, 1989.
- [678] A.P. Punnen. Traveling salesman problem under categorization. *Oper. Res. Lett.*, 12:89–95, 1992.
- [679] A.P. Punnen. Combinatorial optimization with multiplicative objective function. Technical report, Department of Mathematical Sciences, University of New Brunswick-Saint John, 2000.
- [680] A.P. Punnen. The traveling salesman problem: new polynomial approximation algorithms and domination analysis. *J. Inform. Optim. Sci.*, 22:191–206, 2001.
- [681] A.P. Punnen and Y.P. Aneja. On k -sum optimization problems. *Oper. Res. Lett.*, 18:233–236, 1996.
- [682] A.P. Punnen and F. Glover. Implementing ejection chains with combinatorial leverage for the TSP. Research report, University of Colorado-Boulder, 1996.
- [683] A.P. Punnen and S. Kabadi. Domination analysis of some heuristics for the asymmetric traveling salesman problem. *Discrete Appl. Math.*, to appear, 2001.
- [684] A.P. Punnen, F. Margot, and S. Kabadi. TSP heuristics: domination analysis and complexity. Tech. Report 6, Dept. Math., University of Kentucky, 2001.

- [685] A.P. Punnen and K.P.K. Nair. A fast and simple algorithm for the bottleneck biconnected spanning subgraph problem. *Inform. Process. Lett.*, 50:283–286, 1994.
- [686] A.P. Punnen and K.P.K. Nair. Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discrete Appl. Math.*, 55:91–93, 1994.
- [687] M. Queyranne and Y. Wang. Hamiltonian path and symmetric travelling salesman polytopes. *Math. Program. Ser. A*, 58:89–110, 1993.
- [688] M. Queyranne and Y. Wang. On the Chvátal rank of certain inequalities. Tech. report, University of British Columbia, Vancouver, BC, 1994.
- [689] M. Queyranne and Y. Wang. Symmetric inequalities and their composition for asymmetric travelling salesman polytopes. *Math. Oper. Res.*, 20:838–863, 1995.
- [690] L.V. Quintas and F. Supnik. On some properties of shortest Hamiltonian circuits. *Amer. Math. Mon.*, 72:977–980, 1965.
- [691] R. Ramakrishnan, A.P. Punnen, and P. Sharma. An efficient heuristic algorithm for the bottleneck traveling salesman problem. Technical report, IIT Kanpur, India, 2001.
- [692] R. Ramesh and R.C. Brown. An efficient four-phase heuristic for the generalized orienteering problem. *Comput. Oper. Res.*, 18:151–165, 1991.
- [693] R. Ramesh, Y.-S. Yoon, and M.H. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA J. Comput.*, 4:155–165, 1992.
- [694] T. Ramesh. Traveling purchaser problem. *Opsearch*, 18:78–91, 1981.
- [695] M.R. Rao. A note on the multiple traveling salesman problem. *Oper. Res.*, 28:628–632, 1980.
- [696] S. Rao and W. Smith. Approximating geometric graphs via “spanners” and “banyans”. In *Proc. 30th Ann. ACM Symp. Theory Comput.*, pages 540–550, 1998.
- [697] R. Rardin and R.G. Parker. An efficient algorithm for producing a Hamiltonian cycle in the square of a biconnected graph. Industrial and Systems Engineering Report Series J-82, Georgia Institute of Technology, 1983.
- [698] R.L. Rardin, C.A. Tovey, and M.G. Pilcher. Analysis of random cut test instance generator for the TSP. In P.M. Pardalos, editor, *Complexity in Numerical Optimization*, pages 387–405. World Scientific, Singapore, 1993.
- [699] H.D. Ratliff and A.S. Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Oper. Res.*, 31(3):507–521, 1983.
- [700] S.S. Reddi and V. Ramamoorthy. On the flow-shop sequencing problem with no wait in process. *Oper. Res. Quart.*, 23(3):323–331, 1972.
- [701] C. Redmond. *Boundary Rooted Graphs and Euclidean Matching Algorithms*. PhD thesis, Dept. Math., Lehigh University, 1993.
- [702] C. Redmond and J.E. Yukich. Limit theorems and rates of convergence for subadditive Euclidean functionals. *Ann. Appl. Prob.*, 4:1057–1073, 1994.
- [703] C. Redmond and J.E. Yukich. Asymptotics for Euclidean functionals with power weighted edges. *Stoch. Proc. Appl.*, 61:289–304, 1996.

- [704] C. Rego. Relaxed tours and path ejections for the traveling salesman problem. *Eur. J. Oper. Res.*, 106:522–538, 1998.
- [705] C. Rego. A subpath ejection method for the vehicle routing problem. *Manage. Sci.*, 44(10):1447–1459, 1998.
- [706] C. Rego. Node ejection chains for the vehicle routing problem: Sequential and parallel algorithms. *Parallel Comput.*, 27:201–222, 2001.
- [707] C. Rego and B. Alidaee. *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. OR/CS Interfaces Series. Kluwer, Boston, to appear.
- [708] C. Rego and P. Leão. A Scatter Search Tutorial for Graph-Based Permutation Problems. Technical Report HCES-10-00, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, MS, 2000.
- [709] G. Reinelt. TSPLIB — a traveling salesman problem library. *ORSA J. Comput.*, 3:376–384, 1991. <http://www.crpc.rice.edu/softlib/tsplib/>.
- [710] G. Reinelt. Fast heuristics for large geometric traveling salesman problems. *ORSA J. Comput.*, 4:206–217, 1992.
- [711] G. Reinelt. *The Travelling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Berlin, 1994.
- [712] A.G. Reinhold. Some results on minimal covertex polygons. Manuscript written under the supervision of Fred Supnick, City College of New York, 1965.
- [713] J. Renaud, F.F. Boctor, and G. Laporte. A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS J. Comput.*, 8:134–143, 1996.
- [714] B.W. Repetto. *Upper and Lower Bounding Procedures for the Asymmetric Traveling Salesman Problem*. PhD thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, 1994.
- [715] W. Rhee. On the fluctuations of the stochastic traveling salesperson problem. *Math. Oper. Res.*, 16:482–489, 1991.
- [716] W. Rhee. On the traveling salesperson problem in many dimensions. *Random Struct. Algorithms*, 3:227–233, 1992.
- [717] W. Rhee. A matching problem and subadditive Euclidean functionals. *Ann. Appl. Prob.*, 3:794–801, 1993.
- [718] W. Rhee. On the stochastic Euclidean traveling salesperson problem for distributions with unbounded support. *Math. Oper. Res.*, 18:292–299, 1993.
- [719] W. Rhee. Boundary effects in the traveling salesperson problem. *Oper. Res. Lett.*, 16:19–25, 1994.
- [720] W. Rhee and M. Talagrand. Martingale inequalities and NP-complete problems. *Math. Oper. Res.*, 12:177–181, 1987.
- [721] W. Rhee and M. Talagrand. A sharp deviation inequality for the stochastic traveling salesman problem. *Ann. Prob.*, 17:1–8, 1989.
- [722] W. Rhee and M. Talagrand. On the long edges in the shortest tour through n random points. *Combinatorica*, 12:323–330, 1992.
- [723] F.S. Roberts. *Discrete Mathematical Models with Applications to Social, Biological, and Environmental Problems*. Prentice Hall, New Jersey, 1976.

- [724] N. Robertson and P.D. Seymour. Graph Minors X. Obstructions to tree decomposition. *J. Comb. Theory, Ser. B*, 52:153–190, 1991.
- [725] J.B. Robinson. On the Hamiltonian game: a traveling salesman problem. Technical report, RAND Research Memorandum RM-303, 1949.
- [726] R.W. Robinson and N.C. Wormald. Almost all cubic graphs are Hamiltonian. *Random Struct. Algorithms*, 3:117–126, 1992.
- [727] R.W. Robinson and N.C. Wormald. Almost all regular graphs are Hamiltonian. *Random Struct. Algorithms*, 5:363–374, 1994.
- [728] H. Rock. The three-machine no-wait flow-shop in NP-complete. *J. Assoc. Comput. Mach.*, 31:336–345, 1984.
- [729] A. Rohe, 2001. Personal communication.
- [730] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 1977.
- [731] G. Rote. *Two Solvable Cases of The Traveling Salesman Problem*. PhD thesis, Technical University Graz, Graz, Austria, 1988.
- [732] G. Rote. The n -line traveling salesman problem. *Networks*, 22:91–108, 1992.
- [733] V.I. Rublineckii. Estimates of the accuracy of procedures in the traveling salesman problem. *Numerical Mathematics and Computer Technology*, no. 4:18–23, 1973. (in Russian).
- [734] S. Sahni and T. Gonzalez. P-complete approximation problem. *J. Assoc. Comput. Mach.*, 23:555–565, 1976.
- [735] J.J. Salazar. *Algoritmi per il problema del Commesso Viaggiatore Generalizzato*. PhD thesis, Royal Spanish College in Bologna, 1992. (in Italian).
- [736] J.J. Salazar. The polytope of the cycle problem on undirected graphs. Working paper, University of La Laguna, 1994.
- [737] D. Sanders. *On Extreme Circuits*. PhD thesis, City University of New York, 1968.
- [738] V.I. Sarvanov. On the minimization of a linear form on a set of all n -elements cycles. *Vestsi Akad. Navuk BSSR, Ser. Fiz.-Mat. Navuk*, no. 4:17–21, 1976. (in Russian).
- [739] V.I. Sarvanov. The mean value of the functional of the assignment problem. *Vestsi Akad. Navuk BSSR Ser. Fiz. -Mat. Navuk*, no. 2:111–114, 1976. (in Russian).
- [740] V.I. Sarvanov. *On Quadratic Choice Problems*. PhD thesis, Inst. Mat. AN BSSR, Minsk, USSR, 1978. (in Russian).
- [741] V.I. Sarvanov. On complexity of minimization of a linear form on a set of cyclic permutations. *Dokl. Akad. Nauk SSSR*, 253:533–535, 1980. (in Russian).
- [742] V.I. Sarvanov. On the complexity of minimizing a linear form on a set of cyclic permutations. *Soviet Math. Dokl.*, 22:118–210, 1980. (in Russian).
- [743] V.I. Sarvanov and N.N. Doroshko. The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of exponential cardinality in quadratic time. In *Software: Algorithms and Programs*,

- volume 31, pages 8–11. Math. Institute of the Belorussian Acad. Sci., Minsk, 1981. (in Russian).
- [744] V.I. Sarvanov and N.N. Doroshko. The approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. In *Software: Algorithms and Programs*, volume 31, pages 11–13. Math. Institute of the Belorussian Acad. Sci., Minsk, 1981. (in Russian).
- [745] F. Scholz. Coordination hole tolerance stacking. Technical Report BCSTECH-93-048, Boeing Computer Services, 1993.
- [746] F. Scholz. Tolerance stack analysis methods. Technical Report BCSTECH-95-030, Boeing Computer Services, 1995.
- [747] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.
- [748] F. Semet and J. Renaud. A tabu search algorithm for the symmetric generalized traveling salesman problem. Working paper, University of Montreal, 1999.
- [749] M.M. Sepsri. *The symmetric generalized travelling salesman problem*. PhD thesis, University of Tennessee, Knoxville, 1991.
- [750] T. Seppäläinen and J.E. Yukich. Large deviation principles for Euclidean functionals and other nearly additive processes. *Prob. Theory Relat. Fields*, 120:309–345, 2001.
- [751] A.I. Serdyukov. An algorithm with an estimate for the traveling salesman problem of the maximum. *Upravlyaemye Sistemy*, no. 25:80–86, 1984. (in Russian).
- [752] A.I. Serdyukov. Complexity of the traveling salesman problem with a description on graphs with small degrees of vertices. *Upravlyaemye Sistemy*, 85:73–82, 1985. (in Russian).
- [753] A.I. Serdyukov. An asymptotically exact algorithm for the traveling salesman problem for a maximum in Euclidean space. *Upravlyaemye Sistemy*, no. 27:79–87, 1987. (in Russian).
- [754] A.I. Serdyukov. Polynomial time algorithms with estimates of accuracy of solutions for one class of the maximum cost TSP. In *Kombinatorno-Algebraicheskie Metody v Diskretnoi Optimizatsii, Mezhvuzovskiy sbornik*, pages 107–114. Nauka, N. Novgorod, 1991. (in Russian).
- [755] A.I. Serdyukov. The maximum-weight traveling salesman problem in finite-dimensional real spaces. *Diskretnyi Analiz i Issledovanie Operatsii*, 2(1):50–56, 1995. in Russian, Translated in *Operations research and discrete analysis. Translation of Diskret. Anal. Issled. Oper. 2 (1995), no. 1-4*, Mathematics and its Applications, 391, Kluwer, Dordrecht, 1997; 233–239.
- [756] S.I. Sergeev. Algorithms for the minmax problem of the traveling salesman I: An approach based on dynamic programming. *Autom. Remote Control*, 56:1027–1032, 1995.
- [757] S.I. Sergeev and A.B. Chernyshenko. Algorithms for the minmax problem of the traveling salesman II: Dual approach. *Autom. Remote Control*, 56:1155–1168, 1995.

- [758] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *Internat. J. Flexible Manufacturing Systems*, 4:331–358, 1992.
- [759] E. Shamir. How many random edges make a graph Hamiltonian? *Combinatorica*, 3:123–132, 1983.
- [760] M.I. Shamos. *Computational Geometry*. PhD thesis, Yale University, 1975.
- [761] D.B. Shmoys and D.P. Williamson. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Inform. Proc. Lett.*, 35:281–285, 1990.
- [762] K.N. Singh and D.L. van Oudheusden. A branch and bound algorithm for the traveling purchaser problem. *Eur. J. Oper. Res.*, 97:571–579, 1997.
- [763] S.S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory in Mathematica*. Addison-Wesley, Redwood City, CA, 1990.
- [764] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.
- [765] D.H. Smith and S. Hurley. Bounds for the frequency assignment problem. *Discrete Math.*, 167/168:571–582, 1997.
- [766] W.D. Smith. Finding the optimum N -city traveling salesman tour in the Euclidean plane in subexponential time and polynomial space. Manuscript, 1988.
- [767] S. Somhom, A. Modares, and T. Enkawa. Competition-based neural network for the multiple travelling salesman problem with minmax objective. *Comput. Oper. Res.*, 26:395–407, 1999.
- [768] J.M. Steele. Subadditive Euclidean functionals and non-linear growth in geometric probability. *Ann. Prob.*, 9:365–376, 1981.
- [769] J.M. Steele. Probabilistic algorithm for the directed traveling salesman problem. *Math. Oper. Res.*, 11:343–350, 1986.
- [770] J.M. Steele. Growth rates of Euclidean minimal spanning trees with power weighted edges. *Ann. Prob.*, 16:1767–1787, 1988.
- [771] J.M. Steele. Probabilistic and worst case analyses of classical problems of combinatorial optimization in Euclidean space. *Math. Oper. Res.*, 15:749–770, 1990.
- [772] J.M. Steele. *Probability Theory and Combinatorial Optimization*. SIAM, 1997.
- [773] J.M. Steele and T.L. Snyder. Worst case growth rates of some classical problems of combinatorial optimization. *SIAM J. Comput.*, 18:278–287, 1989.
- [774] P.C. Steven, L.G. Bruce, and E.A. Wasil. A computational study of smoothing heuristics for the Traveling Salesman Problem. *Eur. J. Oper. Res.*, 124:15–27, 1999.
- [775] P.S. Sundararaghavan. The unit capacity pickup and delivery problem on a one-way loop. *Opsearch*, 21(4):209–226, 1984.
- [776] F. Supnick. Extreme Hamiltonian lines. *Ann. Math.*, 66:179–201, 1957.
- [777] D.A. Suprunenko. The traveling-salesman problem. *Cybernetics*, 11:799–803, 1975.
- [778] D.A. Suprunenko. On the minimum of the value $v(A, t)$ on the set of cycles. Preprint 14, Inst. Mat. Akad. Nauk. BSSR, Minsk, USSR, 1976. (in Russian).

- [779] M.M. Syslo, N. Deo, and J.S. Kowalik. *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall, Englewood Cliffs, NJ, 1983.
- [780] M. Talagrand. Complete convergence of the directed TSP. *Math. Oper. Res.*, 16:881–887, 1991.
- [781] M. Talagrand. Concentration of measure and isoperimetric inequalities in product spaces. *Publications Mathématiques de l’I.H.E.S.*, 81:73–205, 1995.
- [782] M. Talagrand. A new look at independence. *Ann. Prob.*, 24:1–34, 1996.
- [783] M. Talagrand and J.E. Yukich. The integrability of the square exponential transportation cost. *Ann. Appl. Prob.*, 3:1100–1111, 1993.
- [784] A. Tamir and J.S.B. Mitchell. A maximum b -matching problem arising from median location models with applications to the roommate problem. *Math. Program.*, 80:171–194, 1998.
- [785] S.P. Tarasov. Properties of the trajectories of the appointments problem and the travelling-salesman problem. *U.S.S.R. Comput. Maths. Math. Phys.*, 21(1):167–174, 1981.
- [786] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.
- [787] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:2:146–160, 1972.
- [788] R.E. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.
- [789] A. Tetsuka. Optimal lacings. *Math. Intell.*, 19:6, 1997.
- [790] S. Thienel. *ABACUS—A Branch-And-CUt System*. PhD thesis, Universität zu Köln, 1995.
- [791] A. Thomason. A simple linear expected time algorithm for Hamilton cycles. *Discrete Math.*, 75:373–379, 1989.
- [792] T.W. Tillson. A Hamiltonian decomposition of K_{2m}^* , $2m \geq 8$. *J. Combin. Theory, Ser. B*, 29(1):68–74, 1980.
- [793] E.A. Timofeev. Minmax di-associated subgraphs and the bottleneck traveling salesman problem. *Cybernetics*, 4:75–79, 1979.
- [794] L. Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and MST. In *Proc. 29th ACM Symp. Theory Comput.*, pages 21–39, 1997.
- [795] L. Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and Steiner tree. *SIAM J. Comput.*, 30:475–485, 2000.
- [796] T. Tsiligrirides. Heuristic methods applied to orienteering. *J. Oper. Res. Soc.*, 35:797–809, 1984.
- [797] P. Underground. On graphs with Hamiltonian squares. *Discrete Math.*, 21:323, 1978.
- [798] R. van Dal. *Special Cases of the Traveling Salesman Problem*. PhD thesis, University of Groningen, 1992.
- [799] R. van Dal, J.A.A. van der Veen, and G. Sierksma. Small and large TSP: two polynomially solvable cases of the traveling salesman problem. *Eur. J. Oper. Res.*, 69:107–120, 1993.

- [800] E.S. van der Poort, M. Libura, G. Sierksma, and J.A.A. van der Veen. Solving the k -best traveling salesman problem. *Comput. Oper. Res.*, 26:409–425, 1999.
- [801] J.A.A. van der Veen. *Solvable Cases of the Traveling Salesman Problem with Various Objective Functions*. PhD thesis, University of Groningen, 1992.
- [802] J.A.A. van der Veen. An $O(n)$ algorithm to solve the bottleneck traveling salesman problem restricted to ordered product matrice. *Discrete Appl. Math.*, 47:57–75, 1993.
- [803] J.A.A. van der Veen. A new class of pyramidally solvable symmetric traveling salesman problems. *SIAM J. Discrete Math.*, 7(4):585–592, 1994.
- [804] J.A.A. van der Veen, G. Sierksma, and R. van Dal. Pyramidal tours and the traveling salesman problem. *Eur. J. Oper. Res.*, 52:90–102, 1991.
- [805] J.A.A. van der Veen and R. van Dal. Solvable cases of the no-wait flow-shop scheduling problem. *J. Oper. Res. Soc.*, 42(11):971–980, 1991.
- [806] J.A.A. van der Veen, G.J. Woenginger, and S. Zhang. Sequencing jobs that require common resources on a single machine: a solvable case of the TSP. *Math. Program. Ser. A*, 82:235–254, 1998.
- [807] J.A.A. van der Veen and S. Zhang. Low-complexity algorithms for sequencing jobs with a fixed number of job-classes. *Comput. Oper. Res.*, 23(11):1059–1067, 1996.
- [808] D. van Wieren and Q.F. Stout. Random graph algorithms for the mesh with row and column subbuses. In *Proc. 2nd Workshop Reconfigurable Architectures*, pages 1–13. Springer, 1995.
- [809] S. Vishwanathan. An approximation algorithm for the asymmetric traveling salesman problem with distances one or two. *Inform. Process. Lett.*, 44:297–302, 1992.
- [810] V.G. Vizing. Values of the target functional in a priority problem that are majorized by the mean value. *Kibernetika, Kiev*, no. 5:76–78, 1973. (in Russian).
- [811] R.V. Vohra. Probabilistic analysis of the longest hamiltonian tour. *Networks*, 18(1):13–18, 1988.
- [812] O. Vornberger. Easy and hard cycle covers. Technical report, Univerität Paderborn, Germany, 1980.
- [813] O. Vornberger. *Komplexität von Wegeproblemen in Graphen*. PhD thesis, Gesamthochschule Paderborn, West Germany, 1980.
- [814] S. Voss. Dynamic tabu search strategies for the traveling purchaser problem. *Ann. Oper. Res.*, 63:253–275, 1996.
- [815] C. Voudouris and E. Tsang. Guided local search and its application to the travelling salesman problem. *Eur. J. Oper. Res.*, 113:469–499, 1999.
- [816] K. Wagner. Bemerkungen zu hadwigers vermutung. *Math. Ann.*, 141:433–451, 1960.
- [817] D.W. Walkup. On the expected value of a random assignment problem. *SIAM J. Comput.*, 8:440–442, 1979.
- [818] C. Walshaw. A multilevel approach to the travelling salesman problem. *Oper. Res.*, to appear. Preliminary version available as Mathematics Research Report

- 00/1M/63, Computing and Mathematical Sciences Department, University of Greenwich.
- [819] R.H. Warren. Pyramidal tours for the traveling salesman. *Optimization*, 29(1):81–87, 1994.
- [820] R.H. Warren. Special cases of the traveling salesman problem. *Appl. Math. Comput.*, 60:171–177, 1994.
- [821] D.J.A. Welsh. *Matroid Theory*. Academic Press, London, 1976.
- [822] K. Wenger. Kaktus-Repräsentation der minimalen Schnitte eines Graphen und Anwendung im Branch-and-Cut Ansatz für das TSP. Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Institut für Informatik, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany, www.informatik.uni-heidelberg.de/groups/comopt, 1999. (in German).
- [823] H. Weyl. Elementare Theorie der Konvexen Polyeder. *Comment. Math. Helv.*, 7:290–306, 1935.
- [824] D.A. Wismer. Solution of the flowshop scheduling problem with no intermediate queues. *Oper. Res.*, 20:689–697, 1972.
- [825] L.A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Math. Program. Study*, 13:121–134, 1980.
- [826] L.A. Wolsey. *Integer Programming*. Wiley, Chichester, 1998.
- [827] R. Wong. Integer programming formulations of the traveling salesman problem. In *Proc. IEEE Internat. Confer. Circuits Comput.*, pages 149–152, 1980.
- [828] N. Wormald. Models of random regular graphs. In *Surveys in Combinatorics Proc. 17th British Combinatorial Conference*, pages 239–298. Cambridge Press, 1999.
- [829] J. Xu, S. Chiu, and F. Glover. Tabu Search and Evolutionary Scatter Search for 'Tree-Star' Network Problems, with Applications to Leased-Line Network Design. In *Telecommunications Optimization*. Wiley, Chichester, to appear.
- [830] M. Yagiura, T. Ibaraki, and F. Glover. An Ejection Chain Approach for the Generalized Assignment Problem. Technical report, Graduate School of Informatics, Kyoto University, 2000.
- [831] Q.F. Yang, R.E. Burkard, E. Cela, and G.J. Woeginger. Hamiltonian cycles in circulant digraphs with two stripes. *Discrete Math.*, 176:233–254, 1997.
- [832] M. Yannakakis. Computational Complexity. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–56. Wiley, Chichester, 1997.
- [833] J.E. Yukich. Asymptotics for the stochastic TSP with power weighted edges. *Prob. Theory Relat. Fields*, 102:203–220, 1995.
- [834] J.E. Yukich. Ergodic theorems for some classical optimization problems. *Ann. Appl. Prob.*, 6:1006–1023, 1996.
- [835] J.E. Yukich. *Probability Theory of Classical Euclidean Optimization Problems*, volume 1675 of *Lect. Notes Math.* Springer-Verlag, 1998.
- [836] A.Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner Problem. *Algorithmica*, 9:463–470, 1993.

- [837] W. Zhang. Truncated branch-and-bound: A case study on the asymmetric TSP. In *Proc. AAAI 1993 Spring Symp. AI and NP-Hard Problems*, pages 160–166, Stanford, CA, 1993.
- [838] W. Zhang. Depth-first branch-and-bound versus local search: A case study. In *Proc. 17th National Conf. Artif. Intell. (AAAI-2000)*, pages 930–935, Austin, TX, 2000.

List of Figures

1.1	Eyelets on a shoe	4
1.2	Zig-zag Lacing	6
1.3	Straight Lacing	6
1.4	quick Lacing	6
1.5	diagonal Lacing	6
2.1	The diamond graph	38
2.2	A graph and one of its closed walks	41
2.3	Excluded minors for Theorem 15	43
2.4	Examples of “shortcuts”	46
2.5	Examples of 1-node liftings	56
2.6	Examples of edge cloning	57
2.7	Examples of clonable and non clonable edges	58
2.8	Example of a comb ($t = 3$)	60
2.9	Example of intervals	63
2.10	Example of a path ($h = 6, t = 3$)	64
2.11	Example of minimal trace with T_2 not tight ($r = 3, s - 1 = 4$)	65
2.12	A facet inducing star (non-path) inequality; RHS=24	66
2.13	Examples of non facet and facet inducing star inequalities	66
2.14	Example of a Clique Tree	68
2.15	Example of a Bipartition	69
2.16	Traces of tight tours with 2 and 4 edges in $\delta(T)$	70
2.17	Traces of tight tours with 2, 4 and 6 edges in $\delta(T^*)$	71
2.18	Example of a ladder	72
2.19	Trace of a tight tour containing $e \in (H_1 \cap T_1 : H_2 \cap T_2)$	73
2.20	Example of a facet defining inequality	76
2.21	Some traces of Hamiltonian cycles	77
2.22	Example of domino inequality	79
2.23	Tight tours for the example of Figure 2.22	79
2.24	Example of a fractional point	88
2.25	The graph of Figure 2.24 after shrinking	88
2.26	Examples of violated combs from Figure 2.24	88
2.27	Two crossing minimum cuts	91
2.28	Edges in maximally violated comb	92
2.29	Examples of non violated comb inequalities	94

2.30	Examples of violated path inequalities with 2 handles	95
2.31	Example of a violated path inequality in ts225	95
2.32	Examples of violated clique trees	96
2.33	Example of violated clique tree	96
2.34	Violated clique tree and path inequalities from pr299	97
2.35	Example how ladders can be violated	97
2.36	Example of ladder violation in gil262	98
2.37	Example of violated ladders from pr439	98
2.38	Violated ladder from pr136 with correcting term	99
2.39	Violated ladders from pr439 with correcting term	99
2.40	Part of a fractional solution of fnl4461	101
2.41	The same solution after shrinking	102
2.42	Flowchart of a Branch-and-Cut algorithm	106
2.43	Illustration for the third branching rule	112
3.1	A clique tree with $r = 3$ handles and $s = 9$ teeth	122
3.2	Curtain inequality for the ATSP and corresponding inequality for the STSP.	123
3.3	Fork inequality for the ATSP and corresponding inequality for the STSP.	124
3.4	The support graph of a D_k^+ (left) and of a D_k^- (right) inequality when $k = 5$.	125
3.5	The support graph of a T_k inequality ($k = 3$).	127
3.6	The support graph of a C2 inequality.	127
3.7	The support graph of a C3 inequality.	128
3.8	Illustration of the merge lifting operation.	140
3.9	A T_2 inequality (a) and its canonical form (b).	145
3.10	Some odd CAT's.	147
3.11		149
3.12	The support graph of a primitive SD inequality.	151
3.13	A general SD inequality	153
3.14	Crossing and noncrossing chords. The internal nodes w.r.t. (i_a, i_b) are marked with +, and the chords are drawn in double lines.	157
3.15	A noose Q with $q = 4$.	157
3.16	The dashed lines represent chords with coefficient 0.	158
3.17	Support graphs of two lifted cycle inequalities having a single chord with coefficient 2.	159
3.18	Counterexample for the converse of Theorem 57.	161
3.19	Support graphs of two shell inequalities.	162
3.20	Support graphs of two fork inequalities.	163
3.21	2-liftable chord sets of curtain inequalities.	165
3.22	Support graphs of curtain inequalities for $ C = 5, 6, 7$.	166
4.1	Branching on subtour $G_p = (V_p, A_p)$, where $V_p = \{a, b, c\}$ and $A_p = \{(a, b), (b, c), (c, a)\}$.	173
5.1	The dissection	210
5.2	The enclosing box contains the bounding box and is twice as large, but is shifted by random amounts in the x and y directions.	211

5.3	This portal-respecting tour enters and leaves the square 10 times, and the portion inside the square is a union of 5 disjoint paths.	213
5.4	Every crossing is moved to the nearest portal by adding a “detour.”	214
5.5	The tour crossed this line segment 6 times, but breaking it and reconnecting on each side (also called “patching”) reduced the number of crossings to 2.	216
6.1	Example when $n = 6$ and $k = 3$. The path connecting the thick nodes correspond to the tour $x_1x_3x_4x_2x_6x_5x_1$.	236
7.1	A rotation with x_0 as fixed end point.	262
7.2	Patching C_1 and C_2 together	271
8.1	3-opt obtained by two successive 2-opt moves.	317
8.2	The String/Unstring Neighborhood Structure	320
8.3	Possible moves at the first level of the Lin-Kernighan process	322
8.4	The General Lin-Kernighan Procedure	326
8.5	The <i>double-bridge</i> neighborhood.	328
8.6	Two levels of a multi-node insertion ejection chain	332
8.7	Two levels of a multi-node exchange ejection chain	333
8.8	Neighborhood search iteration for node-based ejection chains.	334
8.9	The stem-and-cycle ejection chain	337
8.10	An Iteration of the Stem-and-Cycle Procedure	339
8.11	The Filter and Fan Model	343
8.12	A General Filter and Fan Procedure	346
8.13	A Scatter Search Template	358
8.14	Path Relinking	365
9.1	Running times for benchmark greedy code as a function of instance size for a variety of microprocessor-based machines. The microprocessors are listed in the order of their average times for 1000-city instances.	376
9.2	Ratios of predicted running time to actual running time on a Compaq 500-Mhz Alpha processor for the Benchmark Greedy code and for the Johnson-McGeoch implementation of Lin-Kernighan.	378
9.3	Average tradeoffs between tour quality and normalized running time for a variety of heuristics applied to 10,000-city Random Uniform Euclidean Instances. The full abbreviations for the heuristic names are given in Table 9.2 below and explained in Table 9.19 at the end of this chapter.	382
9.4	Tour length and normalized running time comparisons: Nearest Insertion versus Nearest Neighbor.	384
9.5	Tour quality comparisons for the Strip and Spacefilling Curve heuristics.	389
9.6	Average percentage excess for pure augmentation heuristics. (For an explanation of the abbreviations, see the text or Table 9.19.) Note that the ranges of N are different for the two classes of instances.	395
9.7	Tour quality comparisons for Farthest Insertion and Savings heuristics.	402
9.8	Tour quality comparisons for CCA and Savings heuristics.	403
9.9	Tour quality for Christofides with greedy shortcuts.	404

9.10	Tour quality comparisons between the Rego-Glover-Gamboia implementation of the Stem-and-Cycle variant of Lin-Kernighan with Boruvka starting tours and the Johnson-McGeoch implementation of basic Lin-Kernighan.	422
9.11	Tour quality comparisons between the Helsgaun variant on Lin-Kernighan and the Johnson-McGeoch implementation of basic Lin-Kernighan.	423
9.12	Tour quality comparisons between taking the best of ten runs of CLK-ABCC-N and taking the result of a single run of CLK-ABCC-10N.	434
9.13	Tour quality comparisons between a $0.1N$ iteration run of Helsgaun's LK variant and taking the best of ten runs of CLK-ABCC-N.	435
9.14	Tour quality comparisons between an N -iteration run of Helsgaun's LK variant and taking the best of five tour-merges, each involving ten CLK-ABCC-N tours.	437
11.1	Patching pseudograph G_p^π	495
11.2	The digraph G_π	497
11.3	Order digraph $G_O = (T^*, E_O)$	498
11.4	A simple chain on $S = \{3, 4, 5, 6, 7\}$	500
11.5	Cost structure of a pyramidal tour	502
11.6	Digraph G_B corresponding to warehouse order-picking problem	546
11.7	The convex-hull-and- k -line TSP	551
11.8	Papadimitriou-Steiglitz trap	574
12.1	Constructing a Hamiltonian cycle when n is even	592
12.2	Constructing a partial tour T_2 when n is odd	593
12.3	Patching cycles	596
12.4	Constructing P_i and Q_j	597
12.5	Patching runs into Hamiltonian cycles	604
13.1	Infeasible GTSP solution satisfying (14)–(17). The drawn edges have $x_e = 1$, the blank nodes $y_v = 0$, and the black nodes $y_v = 1/2$.	619
13.2	The layered network LN .	634
13.3	Fractional point violating a cycle cover inequality for the OP instance with $t_0=6$ and $t_e=1$ for all $e \in E$ ($0 < \alpha \leq 1/2$). Here $T = \{(1, i_1), (i_1, i_2), \dots, (i_6, 1)\}$, $x(T) = 2 + 5\alpha$, and $y(V(T)) = 3 + 4\alpha$.	645
13.4	A fractional point violating a path inequality for the OP instance with $t_0=6$ and $t_e=1$ for all $e \in E$ ($0 < \alpha \leq 1/2$).	646
14.1	A typical solution.	666
14.2	Primitive comb inequality for P^*	672
14.3	Primitive odd CAT inequalities for P^*	674
14.4	Primitive SD inequality for P^* .	676
14.5	Primitive clique tree for P_0 with two handles.	678
14.6	Primitive lifted cycle inequalities for P_0 .	680
14.7	Cloning of two nodes of T_2	682
14.8	Cloning every node of T_2	682
14.9	SD inequality obtained by clique-lifting	684
14.10	Lifted 3-cycle inequality with two quasi-clones	685
14.11	Lifted 3-cycle inequality with one quasi-clone for every node	685

14.12 Illustration of Example 19

690

List of Tables

2.1	Some statistics on $STSP(n)$	36
2.2	Computational results	115
2.3	Very large instances with Concorde	116
4.1	ATSP instances.	196
4.2	FT-b&c without and with Fractionality Persistency (10,000 seconds time limit).	198
4.3	Sensitivity of Concorde to 3- and 2-node transformations and to the chunk size parameter.	199
4.4	Comparison of branch-and-bound codes. Time limit of 1,000 seconds.	201
4.5	Comparison of branch-and-cut codes. Time limit of 10,000 seconds.	202
4.6	Comparison of branch-and-bound codes. Time limit of 1,000 seconds.	203
4.7	Comparison of branch-and-cut codes. Time limit of 10,000 seconds.	204
9.1	For instances in the Challenge test suite that have known optimal solutions, the percent by which the optimal tour length exceeds the Held-Karp bound and the normalized running times in seconds for computing each using Concorde with its default settings. (“-” indicates that the default settings did not suffice.) For random instances, suffixes 1k, 3k, and 10k stand for 1,000, 3,162, and 10,000 cities respectively. The number of cities in a TSPLIB instance is given by its numerical suffix.	380
9.2	Average tour quality and normalized running times for various heuristics on the 10,000-city instances in our Random Uniform Euclidean testbed.	382
9.3	Average comparisons between Nearest Insertion (NI) and Nearest Neighbor (NN) on our geometric testbeds. Bentley’s implementations [103] of both heuristics were used. A positive entry in the “Excess” table indicates that the NI tours are longer by the indicated percentage on average. As in all subsequent tables of this sort, the TSPLIB averages are over the following instances: pr1002, pcb1173, r11304, and nrw1379 for $N = 1,000$, pr2392, pcb3038, and fn14461 for $N = 3162$, pla7397 and bfd14051 for $N = 10K$, pla33810 for $N = 31K$, and pla85900 for $N = 100K$. These may not be completely typical samples as we had to pick instances that most codes could handle, thus ruling out the many TSPLIB instances with fractional coordinates.	383

9.4	Average normalized times for reading instances using the standard I/O routines of C, compiled for MIPS processors using gcc.	386
9.5	Average performance of the Strip heuristic.	387
9.6	Average performance of the Spacefilling Curve heuristic.	388
9.7	Normalized running times in seconds for Pure Augmentation heuristics and Random Uniform Euclidean instances.	396
9.8	Average percentage excesses over the Held-Karp bound and normalized running times for Bentley's implementations of Insertion, Addition, and Augmented Addition heuristics applied to 100,000-city Random Uniform Euclidean instances. For comparison purposes, the last column gives results for the Johnson-McGeoch implementation of Savings .	401
9.9	Results for the more powerful tour construction heuristics on Random Uniform Euclidean instances. Sav , ACh , and Chr stand for Savings , AppChristo , and Christo , respectively.	406
9.10	Average percent excess over the HK bound and normalized running times for 100,000-city Uniform and Clustered instances and for TSPLIB instance pla85900 . All codes were run on the same machine.	411
9.11	Average percent excesses over the HK bound and normalized running times for Random Matrix instances of sizes from 1,000 to 10,000 cities. All codes were run on the same machine.	412
9.12	Results for 3-Opt and four implementations of Lin-Kernighan. Averages for TSPLIB are taken over the same instances as in Figure 9.3.	419
9.13	Average percent excesses over the HK bound and normalized running times for Random Matrix instances of sizes from 1,000 to 10,000 cities.	421
9.14	Results for the Helsgaun variant on Lin-Kernighan as compared to those for the Johnson-McGeoch implementation of basic Lin-Kernighan.	424
9.15	Tour quality results for repeated local search heuristics, with Helsgaun and LK-JM included for comparison purposes.	428
9.16	Normalized running times for repeated local search heuristics. The "3M" column was omitted so the table would fit on a page. The missing entries are 1255 seconds for LK-JM and 94700 seconds for CLK-ACR-N .	429
9.17	Best results obtained for all testbed instances whose optimal solutions are known. Four heuristics sufficed to generate these results: Helsgaun-N (abbreviated in the table as h), Tourmerge-b5 (m), ILK-JM-10N (i10), and best of ten runs of ILK-JM-N (ib). Where more than one heuristic found the best solution, we list all that did.	439
9.18	Tour quality and normalized running times for TSPLIB instance d15112 . General conclusions should not be drawn from small differences in quality or time.	442

9.19	Abbreviated names used in this chapter. The symbol X stands for an abbreviation of the implementers' names: "ABCC" for Aplegate, Bixby, Chvátal, and Cook (Concorde), "ACR" for Aplegate, Cook, and Rohe, "B" for Bentley, "JM" for Johnson-McGeoch, "Neto" for Neto, and "R" for Rohe. Adding the suffix "- bn " to any name means that one is taking the best of n runs.	443
10.1	Comparisons between normalized 196 Mhz MIPS processor time and actual running time on the 500 Mhz Alpha target machine.	450
10.2	For the 100-, 316-, and 1000-city instances of each class, the average percentage shortfall of the AP bound from the HK bound and the average asymmetry and triangle inequality metrics as defined in the text. "-" stands for .000.	456
10.3	Metrics for realworld test instances.	456
10.4	Tour quality and normalized times for corresponding symmetric (-S) and asymmetric (-A) codes. Hgaun is an abbreviation for Helsgaun . LK is the Johnson-McGeoch implementation of (symmetric) Lin-Kernighan covered in Chapter 9.	465
10.5	Best average percentage above the HK bound obtained within various normalized running time thresholds for the 316-, 1,000-, and 3162-city instances of the nine asymmetric classes.	467
10.6	Tour quality and normalized running times for fast heuristics.	469
10.7	Tour quality and normalized running times for Cycle Cover Heuristics.	471
10.8	Normalized running times for reading an instance and solving the corresponding Assignment Problem with the AP codes used by COP , Patch , and Zhang .	472
10.9	Tour quality and normalized running times for KP , I3opt , and Ihyper-3 .	474
10.10	Tour quality and normalized running times for IKP and Helsgaun .	475
10.11	Results for optimization via Concorde . A "-" entry indicates that optimization was not feasible via this approach. Running times include the time for running Zhang (first four classes) or Helsgaun (last five) to obtain an initial upper bound, but not the (relatively small) time for the ATSP-to-STSP transformation.	476
10.12	Tour quality and normalized running times for classes amat (Random Asymmetric Matrices) and tmat (Random Asymmetric Matrices closed under shortest paths).	478
10.13	Tour quality and normalized running times for classes shop (50 Processor No-Wait Flowshop Scheduling) and disk (Random Disk Head Motion Scheduling).	479
10.14	Tour quality and normalized running times for classes super (Approximate Shortest Common Superstring) and crane (Stacker-Crane Scheduling).	480
10.15	Tour quality and normalized running times for classes coin (Coin-box Collection Routing) and stilt (Tilted Supnorm Routing).	481
10.16	Tour quality and normalized running times for class rtilt (Tilted Rectilinear Routing). Optima for 1000-city instances were computed by symmetric code applied to equivalent rect .	482
10.17	Robustness metrics for the heuristics in this study.	484

10.18	Results for realworld instances. Patch , Zhang , and Helsgaun are abbreviated by P , Z , and H respectively. The running time for OPT includes both the time to obtain an initial upper bound using Zhang and the time to run Concorde using its default settings on the transformed instance. The relatively small time needed to perform the ATSP to STSP transformation is not included. Instance atex600 could not be optimized in reasonable time using Concorde 's default settings.	485
13.1	Root node statistics.	639
13.2	Branch-and-cut statistics.	640
13.3	Some computational results with different clusterizations.	642
13.4	Results for problems of Class I (OP and VRP instances) with $\alpha = 0.50$.	657
13.5	Additional results for Class I ($\alpha = 0.50$) problems	658
13.6	Results for Class II (TSPLIB instances) and Generation 1 ($\alpha = 0.50$)	659
13.7	Results for Class II (TSPLIB instances) and Generation 2 ($\alpha = 0.50$)	659
13.8	Results for Class II (TSPLIB instances) and Generation 3 ($\alpha = 0.50$)	660
13.9	Average results over 10 random instances of Class III	661
13.10	Average results over 5 random instances of Class IV	661

Topic Index

- 1-1 extension, 492
- 1-arborescence, 181
- 1-arc, 188–189, 193
- 1-edge, 189
- 1-extension, 491
- 1-factor, 590
 - almost, 593
 - maximum weight, 590, 597, 602
- 1-factors, 607
- 1-graphic matroid, 23
- 1-sum, 563
- 1-tree, 636
- 2-cycle, 673
- 2-cycles, 591
- 2-factor, 589
 - 3-restricted, 590
 - 4-restricted, 589
 - 5-restricted, 590
 - k-restricted, 589
 - maximum weight, 593
 - minimum weight, 282
 - near random, 272
- 2-liftable chord, 162
- 2-liftable chords, 158, 160
- 2-liftable set, 156
- 2-matching, 182, 589
- 2-max bound, 704, 714
- 2-Opt, 462
- 2-parity problem, 536
- 2-sum, 563
 - stable, 52
- 2-sums, 51–52
- 3-Opt, 371
- 3-splicing, 569
- 3-sum, 563
- A-path, 525, 528, 715
 - lower, 525
 - upper, 525
- ABACUS, 113
- ABCC, 113
- Absolutely continuous, 297
- Adaptive memory programming, 340, 350, 360, 364
- Additive approach, 176–177
- Additive bound, 169
- Additive bounding, 177
- Admissible neighborhood, 350
- Admissible quadruplet, 556
- Affinely independent, 670
- Airline hubs, 598
- Algorithm
 - Miller and Pekny, 175
 - δ -approximate, 706
 - additive bounding, 178, 181
 - approximation, 207, 705
 - Arora's, 208
 - branch and bound, 703
 - branch and cut, 105, 184, 705
 - Carr's, 90
 - chained Lin-Kernighan, 327
 - Christofides, 207
 - deterministic, 269
 - ejection chain, 250
 - Eppstein, 527
 - extension-rotation, 272, 277
 - fixed dissection, 293
 - GENI, 319
 - GENIUS, 319, 355
 - Gilmore-Gomory, 493
 - Gilmore Gomory, 505, 539
 - greedy, 251, 279, 631, 648
 - greedy covering, 239
 - HAM, 267
 - Hartvigsen's, 589
 - Held-Karp, 255
 - Helsgaun's, 348
 - iterated, 327
 - Karp's, 279
 - Karp's dissection, 209
 - labelling, 174
 - Lawler's, 715
 - Lawler, 538
 - Lin-Kernighan, 250, 312
 - MINCUT, 193
 - minimum cut, 34
 - Nagamoshi-Ibaraki, 84
 - nearest neighbor, 278

- odd-cut separation, 648
- p-relaxation, 712
- Park, 533
- patching, 174–176, 272, 277, 279, 495
- randomized, 267, 594, 598
- rotation-closure, 279
- Sedyukov, 591
- separation, 182, 193, 629, 648
- Smith's, 208
- subtour patching, 607
- vertex insertion, 247
- Alternating path, 131
 - dynamic, 337
- Alternating paths, 337
- Alternating trail, 146
- Amplification factor, 348
- Angle TSP, 25
- Anti greedy, 227
- Antipodal points, 549
- Arborescence, 171, 180, 192
- Arborescence problem, 178
- Arc routing, 13
- Arcs
 - compatible, 156
 - geodesic, 549
 - inessential, 547
 - reduced cost, 175
- Articulation point, 180
- Articulation set, 677
- Aspiration criteria, 350
- Assignment bound, 704
- Assignment problem
 - quadratic, 330
- Assignment problem, 16, 22, 145, 169, 171–172, 231, 275
- Assignment problem, 449
- Assignment problem, 518, 537, 568, 570, 693
 - bottleneck, 715, 724
 - generalized, 328
- Assignment
 - partial, 145
- Asymmetric TSP, 169
- Asymptotics, 208, 292, 297, 305
- Attribute set, 348
- Attributive memory, 350
- Augmenting path, 174–175
- Automorphisms, 247
- B-join, 491
- Basic solution, 190
- Basis, 48
- BBSSP, 704
- Beam search, 340
 - filtered, 340
- Benchmark, 447
- Benchmark code, 370, 375, 449
- Benchmark codes, 372
- Benchmark machine, 375, 379
- Biconnected, 704
- Bijection, 39
- Binomial random variable, 259–260
- Binomial random variables, 295
- Bipartition, 68
- Black and white TSP, 25
- Block, 87
- Blocked approximation, 298
- Blocked distributions, 297
- Blossoms, 589
- Bottleneck TSP, 7, 697
- Boundary effects, 604–605
- Boundary functional, 258, 282, 284
- Boundary TSP, 286–287, 291, 295
- Bounding procedure, 179
- Branch-and-bound, 169, 451, 634, 658
- Branch-and-cut, 169, 619, 642–643, 652
- Branch and bound, 32
- Branch and Bound, 110
- Branch and bound, 113, 172, 174–176, 195, 200, 666
- Branch and cut, 35, 104, 108, 113, 181, 190, 193, 195, 200
- Branch decision, 173
- Branching, 110
- Branching, 637
- Branching node, 172–173
- Branching nodes, 176
- Branching scheme, 172
- Branching
 - on cuts, 638
 - on variables, 638
 - strong, 112
- Bricks, 44
- Bridge, 42
- BSS bound, 704
- BTGG-scheme, 724
- BTGG scheme, 721
- BTGG Scheme, 722
- BTGG scheme, 723
- BTSP, 698
- Bucket sort, 387
- Cactus, 91
- Candidate list
 - fan, 342
 - filter, 342
 - preferred attribute, 348
- Candidate lists, 338, 348
- Cardinality-constrained TSP, 328
- CAT, 185, 673
 - odd, 673
- Catalan numbers, 213
- Categorization TSP, 734
- CATs, 146
- Catter search, 360
- Cell inventory, 664
- Cellular manufacturing, 11

- Characteristic vector, 35, 188, 191
- Chemical industry, 195
- Chernoff's bound, 259–260
- Chord
 - type 1, 675
- Chunk size, 198
- Chunking, 350
- Chvátal-rank, 160
- Chvátal rank, 164
- Circuit, 495
- Clique-liftable, 136
- Clique-lifting, 138
- Clique lifting, 672, 683, 689
- Clique tree, 67, 677
- Clonable, 57
- Clones, 133
- Cloning, 133, 136, 672, 681, 689, 691, 694
 - 2-cycle, 133, 141
 - edge, 141
- Closed walk
 - almost Hamiltonian, 48
 - minimal, 45
 - minimum f -length, 52
- Closure, 569
- Clustered TSP, 8
- Clusterizations, 641
- Coboundaries, 69, 109
- Coboundary, 30, 61, 64, 70, 75–76, 79, 86, 92, 98
- Cocycle, 30
- Coded attributes, 351
- Coloring, 269
- Column generation, 103
- Comb, 54, 194, 673
- Combinatorial explosion, 352
- Combinatorial implosion, 352
- Combinatorial rank, 586, 588, 600
- Compact formulations, 30
- Compact representations, 17
- Compatible, 489
- Complexity, 180, 293, 527, 530, 532, 534, 542, 546–547, 557, 595, 604, 710, 712, 724
- Components
 - connected, 47
 - strongly connected, 242
- Concave, 705
- Concentration estimate, 286
- Concord, 197
- CONCORDE, 113
- Concorde, 205, 310
- Concorde, 372, 379, 410, 418
- Concorde, 448, 462, 476
- Configuration, 264–265
- Configuration model, 264
- Conjecture, 493
- Connected component, 519, 523, 543
- Connected components, 495, 542
- Connected subset, 589
- Consecutive ones, 93
- Constant-TSP, 492
- Constant TSP, 37, 489–491
- Constraint pool, 193
- Constraints
 - 2-matching, 21
 - circuit packing, 16
 - clique packing, 16
 - coupling, 18
 - generalized 2-matching, 631
 - indegree, 119
 - MTZ, 17
 - network flow, 17
 - outdegree, 119
 - subtour elimination, 16, 22, 119, 170
- Convex-hull-and- k -line TSP, 551–552
- Convex-hull-and- k -line TSP, 553
- Convex-hull-and-additional-point TSP, 558
- Convex-hull-and-line-TSP
 - generalized, 554, 557
- Convex-hull-and-line TSP, 551, 555
- Convex-hull TSP, 582
- Convex distance, 301
- Convex distribution, 606
- Convex dual, 303
- Convex hull, 35, 118
- Convex hull TSP, 548–549
- Convex hull
 - control distance, 301
- Cost ranges, 176
- Covering problem, 239
- Covering salesman, 27
- Covering tour problem, 616
- CPLEX, 103
- Crew scheduling, 328
- Crossover operations, 358
- Cryptography, 274
- CT-digraph, 491–492
- CT-graph, 491–492
- Cubic graphs, 264
- Curve reconstruction, 44, 575
- Cut, 30, 35
 - 2-mod, 84
 - minimum capacity, 33, 628
 - minimum crossing, 91
 - mod- k , 83
 - mod-2, 59
 - value of, 30
- Cutnode, 87
- Cuts
 - Chvatal-Gomory, 186
 - conditional, 643, 647
- Cutset, 564, 732
 - 3-edge, 564
 - k -edge, 564

- Cutter, 92
- Cutting plane, 35
- Cycle cover, 275, 458, 590
 - minimum weight, 275
 - near, 277
- Cycle extension, 268
- Cycle relaxation, 644
- D-heap, 175
- Data compression, 587
- Data structure, 193, 391
- De-Poissonize, 296
- Decrease subtours, 175
- Deficiency of node, 539
- Degree balanced, 539, 576
- Degree constraints, 44
- Delaunay triangulation, 106–107, 391, 405, 548
- Deletable, 269
- Delivery man problem, 25
- Demidenko conditions, 515, 517
- Density matrix, 493, 497
- Depth-first search, 580
- Depth, 329
- Derandomizations, 210
- Deterministic estimate, 292
- Dicycle, 687
- Dicycles, 686
- Differential encoding, 358
- Digraph
 - auxiliary, 570
 - circulant, 582, 729
 - k-regular, 250
 - random regular, 273
 - semicomplete, 580
- Dimension, 40
- Directed pseudograph, 539
- Discrete distributions, 607
- Dissection, 210
 - randomized, 211–212
- Divide and conquer, 209
- DLB approach, 353
- DNA sequencing, 537, 587
- Dominant, 130
- Domination analysis, 223, 225, 227
- Domination number, 226, 244, 252, 254
 - maximum, 245, 250
- Domination ratio, 226
- Domino, 78
 - half, 78, 80
- Don't-look bits, 409
- Double-bridge, 327, 416, 462–463
- DS-digraph, 543, 545
- DS-graph, 544
- Dual problem, 107
- Dual solution, 178, 180, 190–191
- Dual variables, 103
- Dynamic programming, 29, 209, 212–213, 219, 255, 271, 399, 432, 525, 527, 530–531, 547, 553, 577, 594, 612, 705
- Ear, 90
- EC methods, 312
- Edge-based encoding, 359
- Edge-exchanges, 314
- Edge coloring, 56
- Edge
 - cloning, 58
 - f-clonable, 57
 - power weighted, 304
 - rim, 562
 - spoke, 562
- Edges
 - crossing, 52
 - intersect, 549
 - strictly intersect, 549
- EGGTSP, 560, 563
- Ejection chain, 321, 329, 348, 354
 - multi-node, 334
 - node based, 330
 - stem-and-cycle, 336, 355
 - subpath, 335
- Ejection Chains, 312, 325
- Ejection moves, 329
- Ellipsoid algorithm, 34
- Ellipsoid method, 568
- Entropic characterization, 304
- Ergodic theorem, 288–289
 - super-additive, 290
- Error bound
 - worst case, 711
- Euclidean functional
 - smooth sub-additive, 284
- Euclidean norm, 588, 605
- Euclidean TSP, 207, 209, 256, 258, 287, 548, 571
 - higher-dimensional, 221
- Euler tour, 399, 458
- Eulerian graph, 41
- Eulerian path, 291
- Eulerian tour, 540, 709
- Eulerian traversal, 216
- Evolutionary method, 356, 360
- Evolutionary methods, 313
- Expected time, 271
- Extension-rotation, 257, 264
- Extreme points
 - integral, 43
- F-adjacent, 50
- F-connected, 50–51
- F&F process, 344
- Facet-lifting, 120
- Facet-lifting, 669
- Facet defining, 667
- Facet

- composition of, 51
- lifting, 133
- nontrivial, 130
- primitive, 672
- regular, 133, 135
- Facets, 39
 - bad, 128
 - trivial, 130
- Fan, 565
- Farkas lemma, 103
- Farthest neighbor, 605
- Fathoming, 192
- Fence, 552, 554
- Fibonacci number, 233
- Film-Copy Deliverer, 26
- Filter width, 340
- Flowshop, 454
- Formulation
 - Integer LP, 610
 - cycle shrink, 31
 - double index, 31–32, 43
 - multicommodity flow, 20
 - multistage insertion, 31
 - single commodity flow, 19
 - subtour elimination, 31
 - two commodity, 19
- Fractionality Persistency, 197
- Frequency assignment, 13
- Fubini's theorem, 296
- G-G problem, 541, 543–544, 546
- Gap, 455
- Generalized TSP, 8
- Generalized TSP, 610, 615, 617
 - E-GTSP, 615, 618
- Genetic algorithms, 357, 371
- Geometric norm, 207, 548
- Geometric STSP, 370
- Geometric TSP, 207, 547–548
- GG-TSP, 503–505, 533, 539–540
- GG scheme, 506–507, 517–518
- GG Scheme, 521
- GG scheme, 522, 524–525, 530, 532, 534, 536, 545
- Gilmore-Gomory scheme, 496, 503–504, 506
- Gilmore-Gomory TSP, 494
- Graph process, 263
- Graph
 - α -adjacency, 132
 - k -quadrant neighbor, 347
 - auxiliary, 571
 - biconnected, 87, 710
 - square, 710
 - bipartite, 627
 - elementary, 569
 - Eulerian, 576
 - Halin, 565, 569, 730
 - hypohamiltonian, 81
 - outer-facial, 566
 - Peterson, 81
 - power of, 707
 - prismatic, 570
 - rank, 624
 - series parallel, 562
- Graphical TSP, 41
 - even generalized, 560
 - generalized, 560–562, 565
- Graphs
 - isomorphic, 51
- GRASP, 371
- Greedy matching, 405
- Greedy scheme, 537
- Greedy tour, 279
- Gridpoint, 210
- Growth bounds, 283
- Hölder continuity, 284
- Hamiltonian decomposition, 250
- Hamming distance, 285, 300
- Handle, 59, 78, 121, 126, 675, 677
- Handles, 62, 73
- Heap queue, 174
- Held and Karp, 321
- Held and Karp bound, 34
- Helicopter touring, 663
- Hereditary property, 510, 716
 - lower, 509, 717
 - upper, 509, 512, 717
- Heuristic
 - Karps's dissection, 208
 - ϵ -optimal, 294
 - 2-opt, 407, 410
 - 2-Opt, 632
 - 2.5-opt, 407, 410
 - 2.5-Opt, 414
 - 2.5opt-B, 412
 - 2opt-JM, 413
 - 3-opt-JM, 413
 - 3-opt, 407, 410, 461, 632
 - 3opt-JM, 412
 - 4-opt, 407
 - AppChristo, 406
 - asymptotically optimal, 606
 - Boruvka, 393–394
 - bottleneck double tree, 708
 - boundary, 293
 - canonical, 293
 - CCA, 402, 405
 - cheapest insertion, 398, 632, 636
 - CHII, 398
 - Christo-G, 405, 412
 - Christo-HK, 405, 412
 - Christofides, 208, 215, 251, 293, 385, 403, 458
 - CI, 398
 - composite, 311

- concorde, 396
- construction, 464
- contract or patch, 459, 470
- cycle cover, 470, 473
- DENN, 392, 394, 396
- DMST, 399, 403
- double tree, 227, 251
- farthest city, 606
- farthest insertion, 632, 636
- fast, 386
- FRP, 390
- generalized 2-opt, 632
- GENI, 414
- GENIUS, 414
- greedy, 227, 375, 377, 383, 392, 396
- Greedy, 464
- Heuristic
 - greedy
 - minimum cut, 84
 - Held-Karp, 293
 - Helsgaun's, 449–450, 462
 - Helsgaun, 427, 435, 476
 - HyperOpt, 415
 - hyperopt, 462
 - insertion variants, 400
 - iterated 3-opt, 462
 - Iterated 3-Opt, 473
 - Iterated Hyperopt, 473
 - iterated KP, 462
 - k-opt, 407
 - Karp's partitioning, 399
 - KP, 461, 464, 473
 - Lin-Kernighan, 110, 321, 325, 385, 415, 714
 - chained, 385
 - Johnson-McGeoch, 377
 - Litke's, 400
 - NA, 397
 - NA+, 397
 - nearest insertion, 383, 385, 632, 636
 - nearest neighbor, 2, 227, 252, 383, 391
 - double-ended, 392
 - NI, 397
 - NN-ABCC, 413
 - NN, 396
 - Or-opt, 407, 414
 - patching, 215
 - partitioning, 607
 - Q-Boruvka, 393–394, 418
 - RA, 470
 - recursive partitioning, 386, 389
 - repeated assignment, 458
 - repeated local search, 461
 - RNN, 252
 - savings, 390, 393–394, 396, 402
 - scaled, 294
- Heuristic
 - separation
 - comb inequality, 631
 - conditional cuts, 651
 - cycle cover, 650
 - spacefilling curve, 385–386
 - strip, 385
 - Strip, 386
 - subtour patching, 606
 - Tabu Search, 431
 - tour construction, 385
 - tourmerge, 436
 - undominated, 381
 - Walshaw's, 435
 - Zhang's, 448, 459, 470
- Heuristics
 - assignment based, 447, 457
 - construction, 311, 447, 457
 - cycle cover, 458
 - cycle patching, 458
 - experimental analysis, 713
 - Greedy, 457
 - improvement, 311
 - local search, 447, 457, 460
 - Nearest Neighbor, 457
 - probability theory, 293
 - variable depth, 666
- Hitting time, 263
- HK bound, 448, 455, 464
- Homogeneity, 283
- Horizontal separation, 219
- Hungarian method, 449
- Hypergraph, 71
- Hypo-Hamiltonian, 128
- Hypo-semi-Hamiltonian, 128
- Iequality
 - path, 65
- Implementation Challenge, 338, 346, 370–371, 451
- Improvement methods, 313–314
- In-edges, 276–278
- Incidence vector, 35, 50
- Incompatibility graph, 186
- Incompatible, 673
- Incremental gap, 177
- Indirect method, 144
- Inequalities
 - D_k^- , 126
 - D_k^+ , 126
 - D_k^+ , 183
 - 2-matching, 644, 648
 - ATS, 120, 128
 - bipartition, 68
 - C2 and T_k , 140
 - C3, 142
 - CAT, 142, 145
 - chain, 58, 142
 - comb, 55, 60, 182, 618, 623

- composition of, 51
- concentration, 258
- cover, 644
- curtain, 125
- equivalent, 142
- facet defining, 120
- facet inducing, 181, 183
- fork, 125
- generalized comb, 634
- isoperimetric, 282
- lifted cycle, 120, 128, 155, 678
- lifting, 667
- maximally 2-lifted cycle, 164
- multi-handle, 98
- odd CAT, 193
- odd CAT and C_2 , 154
- primitive, 134
- SD, 142, 153
- subtour elimination, 92, 158, 618, 679
- symmetric, 182
- valid, 182
- Inequality
 - subtour elimination, 67
 - TT facet, 55, 58
 - h -canonical form, 143
 - T_2 , 144
 - T_k , 126, 138
 - 2-cycle, 133
 - 2-matching, 117
 - arithmetic-geometric mean, 238
 - binested, 73
 - bipartition, 67, 69, 71, 77, 95
 - C_2 , 126, 140
 - C_3 , 126
 - canonical, 48
 - canonical form, 144
 - CAT, 186
 - Chebyshev's, 265, 606
 - clique-tree, 121
 - clique tree, 67, 73, 95, 139, 677, 686
 - closed-set form, 73
 - comb, 59, 62, 64, 79, 109, 121
 - crown, 82
 - cycle, 183
 - cycle cover, 645
 - degree, 45
 - domino, 78–79
 - domino parity, 78
 - external, 45
 - facet defining, 122, 671
 - facet inducing, 38
 - fan, 627
 - FDA, 686
 - generalized 2-matching, 631
 - generalized Chebyshev, 258
 - generalized comb, 623
 - generalized subtour elimination, 627
 - h -liftable, 52
 - hypohamiltonian, 81
 - isoperimetric, 285, 300
 - Jensen's, 258, 288, 292
 - KP, 645
 - ladder, 71, 73, 77, 97
 - lifted 3-cycle, 686
 - lifted cycle, 159, 686
 - merge-lifted, 140
 - multiple handle, 94
 - non negativity, 62
 - odd CAT, 148, 675, 691
 - path, 63, 646, 649
 - primitive CAT, 681
 - primitive comb, 672–673, 677, 684
 - primitive odd CAT, 675, 677, 684
 - primitive SD, 675, 684, 689
 - SD, 151, 675, 683
 - SD primitive, 152
 - simple, 47
 - simple hypohamiltonian, 81
 - star, 62, 65, 76
 - subtour elimination, 31, 44, 62, 74, 101, 109, 164
 - T-lifted, 138
 - tight-triangular, 625
 - tight triangular, 45, 49, 137
 - triangular, 41
 - trivial, 45
 - TT, 47, 49, 625
 - TT facet, 49, 51, 54
 - twisted comb, 79–80
 - valid, 45, 188–189
 - ZT-clique tree, 141–142
- Instance generators, 372
- Instance
 - phylogenetic problem, 451
 - Random Matrix, 451
 - Random Uniform Euclidean, 451
- Instances
 - Additive norm, 452
 - benchmark, 447
 - code optimization, 455
 - coin collection, 454
 - common superstring, 454
 - Disk Drive, 453
 - geometric, 446
 - no-wait flowshop, 454
 - random, 446
 - random 2D Rectilinear, 451
 - random matrix, 451
 - realworld, 455
 - robotic motion, 455
 - Stacker Crane, 453
 - STSP testbed, 448
 - sup norm, 453
 - table compression, 455

- tape drive reading, 455
- Tilted Drilling Machine, 452
- TSPLIB, 446
- vehicle routing, 455
- Integer linear program, 310
- Integer program, 566, 579
- Integer programming, 109
- Integer programming, 170
- Integer programming, 668, 698
- Integrability condition, 301
- Intensification approach, 353
- Intermediate structure, 312
- Internal nodes, 731
- International TSP, 615
- International TSP, 664
- Interportal distance, 214
- Intersection graph, 121, 521
- Interval property, 63
- Isoperimetric, 258, 285
- Isoperimetric methods, 300
- Isoperimetric theory, 300
- JNRT, 114
- Join, 491
- K-cycles, 687
- K-peripatetic salesman, 27
- Kalmanson TSP, 549
- KCAP, 666
- La-path, 525
- Lacing
 - diagonal, 5
 - quick, 5
 - straight, 5
 - zig-zag, 5
- Lagrangean relaxation, 381
- Lagrangean relaxations, 321, 356
- Lagrangean subproblems, 666
- Lagrangian relaxation, 29, 635-637, 639
- Lagrangian
 - restricted, 177
- Large cycles, 278
- Large deviation principle, 303
- Large vertices, 268
- Laundry van problem, 2
- LCF-free, 552, 554
- Legitimacy conditions, 329, 337
- Legitimacy restrictions, 329
- Legitimate structure, 333
- Level, 329
- Lifting, 54, 692, 694
- Lifting coefficients, 125, 669
- Lifting procedure, 621
- Lifting sequence, 621
- Lifting set, 691
- Lifting theorem, 621
- Lifting
 - T, 133
 - 0-node, 137
 - 1-node, 55
 - 2-node, 56
 - clique, 54, 81, 133, 136, 187
 - merge, 133, 139
 - node, 55
 - sequence, 148
 - sequential, 52, 74, 125
 - T, 137-138, 142
- Liftings, 671
- Limit theorems, 258
- Lin-Kernighan, 312, 319, 321, 335, 337, 348, 353-355, 360, 367, 371, 425
- Lin-Kernighan procedure, 325
- Lin-Kernighan
 - basic, 417
 - CLK-ABC-N, 381
 - CLK-ABCC, 426
 - CLK-ACR, 426
 - Helsgaun-.1N, 381
 - Helsgaun-k, 426
 - Helsgaun, 422, 426
 - Iterated, 371
 - iterated, 425
 - ILK-JM, 426
 - LK-Neto, 426
 - LH-JM, 417
 - LK-ABCC, 417
 - LK-ACR, 418
 - LK-Neto, 417
 - MLLK-.1N, 381
- Linear equations, 36
- Linear program, 103, 105, 280, 543
 - bottleneck, 705
- Linear programming, 15, 178, 209, 448, 566, 573, 605, 705
- Linear programs, 33
- Linear relaxation, 32
- Linear representation, 3
- LK method, 325
- LK procedure, 327, 337
- LK process, 323
- Local cut, 100
- Local search, 101, 209
- Local search, 224
- Local search, 226, 249, 329, 347, 373, 385
 - chained, 425
- Local search
 - iterated
 - 3-Opt, 425
- Long chord, 552
- Longest path, 261
- Longest string, 587
- Look-ahead, 325
- Look ahead, 311
- Lookup table, 213
- Lower bound, 171-173, 178-179, 181, 191, 197, 200, 263, 279

- Assignment, 448
- Assignment, 447
- asymmetric HK, 448
- Balas and Christofides, 177
- Christofides, 177
- Held-Karp, 379, 388, 394, 399, 447
- Held and Karp, 573
- Lower minor, 717
- Lower minors, 510
- Lower
 - minor, 509
- M-salesman TSP, 8
- Machine sequencing, 9
- Makespan, 504
- Manhattan norm, 601
- Marginal value, 107
- Martingale estimates, 302
- Master tour, 582
- Matching problem, 566
- Matching
 - b-matching, 559
 - fractional 2-matching, 567
 - maximum, 231
 - maximum weight, 305, 591, 603
 - minimum cost, 531
 - minimum weight, 231
 - near perfect, 264
 - non-bipartite, 612
 - partial, 552
 - perfect, 216, 276, 281
 - minimum weight, 275, 589
 - weighted, 530
- Matrix
 - reduced cost, 174
 - AM, 517
 - BCT, 493
 - BR, 579, 581
 - Brownian, 581
 - Burkov-Rubinshtein, 579
 - constant tour, 490
 - CT, 490
 - DD-graded, 720, 727
 - Demidenko, 514–515
 - doubly graded, 720, 726
 - downward graded, 720
 - DU-graded, 720
 - generalized Kalmanson, 514–515
 - generalized Supnick, 514–515
 - Gilmore Gomory, 534
 - graded, 725
 - diagonally inward graded, 720
 - diagonally outward, 720
 - Hamming, 301
 - inward graded, 727
 - Kalmanson, 514–515, 549–550, 554, 558, 582
 - non-adjacent, 550
 - non-degenerate, 550
- Matrix
 - kalmanson
 - non-degenerate, 555
 - Klyaus, 517
 - max-distribution, 720, 723, 726
 - max-Klyaus, 721
 - max-Lawler, 714
 - Monge, 522
 - NE-staircase, 534
 - NK, 550
 - ordered product, 508, 725
 - permuted cost, 495, 721
 - product, 506, 524
 - pseudo-large, 505
 - reduced cost, 181
 - residual cost, 177
 - SDCI, 727
 - small, 535, 725
 - sparse, 176
 - Supnick, 514–515
 - Suprunenko-I, 517
 - type 2-GG, 533
 - type k-GG, 534, 536
 - UD-graded, 720
 - upper triangular, 538, 715
 - upward graded, 720
 - UU-graded, 720, 725, 727
 - Van der Veen, 516
- Matroid, 630
- Matroid intersection, 23
- Max-BK-II(k), 718
- MAX-SNP-hard, 208
- MAX TSP, 7, 586, 701
- Maximum flow, 628
- Maximum length tour, 305
- Maximum scatter TSP, 698
- MBTSP, 731
- MCNDP, 541, 543–544
- Mean execution time, 293
- Mean value, 284
- Median cycle problem, 616
- Memetic algorithm, 457
- Memory
 - frequency-based, 350, 352
 - long-term, 350
 - recency-based, 350, 352
 - residence frequency, 352
 - semi-explicit, 351
 - short-term, 350, 354
 - transition frequency, 352
- Messenger problem, 1, 7
- Meta-heuristic, 312
- Meta-heuristics, 255
- Metaheuristics, 313, 367, 371, 457
- Method
 - ejection chain, 331

- Methods
 - constructive neighborhood, 313
 - ejection chain, 327
 - filter and fan, 341
 - noising, 339
 - population-based neighborhood, 314
 - sequential fan, 340
 - transition neighborhood, 314
 - variable depth, 329
- Metric problem, 586
- Metric TSP, 207–208
- Minimal cover, 667
- Minimum-cost cover, 448
- Minimum cost flow, 568
- Minimum cut, 82, 568
- Minimum latency problem, 25
- Minimum makespan, 10
- Minimum Steiner tree, 220
- Minkowski functional, 600–601
- Minkowski norm, 207
- Minor, 43
- Mixed pseudograph, 519–520
- Monotonicity, 283
- Monotonization, 129–130, 145
 - generalized, 130
- Move
 - 2-opt, 409
 - 4-Opt, 425
 - compound, 329
 - k-opt, 407
 - transition, 329
- Moves
 - k-opt, 367
 - L-compound, 341
 - 2-opt, 322
 - 3-opt, 322
 - 4-opt, 321–322
 - 5-opt, 321
 - compound, 341
 - cycle-ejection, 336
 - double bridge, 327
 - legitimate, 344
 - LK, 353
 - multi-node exchange, 330, 332
 - multi-node insertion, 330
 - single node-insertion, 344
 - stem-ejection, 336
 - string, 321
 - unstring, 319
 - variable depth, 327
- Moving target TSP, 27
- MSTSP, 698, 706, 711, 727
- Multi-path, 525, 530, 724
- Multi-set, 265
- Multi-star, 532
- Multi-tour, 525, 530
- Multi-tree, 525, 530–531, 724
- Multigraph, 41, 85, 247, 264, 529–530, 576
 - series-parallel, 562
- Multiple visit TSP, 7
- Multiset, 216, 576
- MVI, 510, 512
- Natural formulations, 17
- Natural partition, 499
- Nearest Neighbor, 381, 464, 606
- Nearly additive, 284
- Necklace, 92
- Necklace Condition, 571
- Neighbor lists, 338
- Neighborhood, 228
- Neighborhood digraph, 240
- Neighborhood search
 - F&F, 344
 - variable, 462
- Neighborhood size, 237
 - upper bound, 238
- Neighborhood structure, 240, 311, 329, 354, 407
 - String/Unstring, 319
- Neighborhood structures, 314
 - compound, 311
- Neighborhood tree, 342
- Neighborhood
 - k-exchange, 314
 - 2-opt, 230
 - assign, 230
 - Balas-Simonetti, 232
 - cardinality, 225
 - Carrier-Villon, 249
 - compound, 330
 - constructive, 355
 - destructive, 355
 - diameter of, 240
 - exponential, 227
 - k-opt, 407
 - Or-opt, 349
 - polynomially searchable, 228
 - PVC, 230, 241
 - pyramidal, 229, 241
 - restricted 4-Opt, 415
 - shifting property, 228
 - structure, 228
- Neighborhoods
 - exponential, 223, 340, 371
 - non-sequential, 327
 - sequential, 327
- Neural nets, 371
- Neutral 2-cycle, 147
- No-wait flow shop, 10
- Node-insertion, 314
- Node deficiency, 541
- Node lifting, 47, 51
- Non-basic variables, 280
- Non-trivial subtours, 495, 519

- Normalization, 375, 447, 449
- Normalized form, 40
- Normed space, 586
- Odd CATs, 148
- Odd cut
 - minimum capacity, 631
- Odd cycle, 39
- Optimal assignment, 277
- Optimum assignment, 279
- Or-insertion, 314
- Or-opt, 312
- Or-Opt, 314
- Or-opt, 318, 335
- Order digraph, 499–500, 502
- Ordered cluster TSP, 26
- Ordered partition, 552
- Orienteering problem, 26
- Orienteering Problem, 610, 614, 643
- Orienteering problem, 664
- Out-edges, 276–278
- Outer nodes, 731
- Overload segment, 217
- Pairing, 264
- Parallel algorithms, 272
- Parallelization, 175
- Parametric technique, 173–174
- Parametrization, 174
- Parity function, 560
- Partition matroid, 23
- Partitioning heuristic, 293
- Patch edges, 276
- Patching, 529, 721
- Patching lemma, 217
- Patching
 - adjacent, 496, 503, 538
 - cycles, 271
 - GG, 503, 508
 - non-adjacent, 538
 - optimal, 604
 - pseudograph, 495–496, 522, 530, 544
 - subtour, 605
- Patchings, 725
- Path-Relinking, 364
- Path-relinking, 366
- Path Relinking, 355
- Path
 - alternating, 715
 - bottleneck, 715
 - minimum weight, 529
 - portal-respecting, 214
 - self-intersecting, 549
- PCTSP, 665
- Peak, 508, 581
 - proper, 581
 - set-back, 581
- Peaks, 718
- Penalty triplets, 731
- Pendant tooth, 142
- Perfect matching, 37
 - minimum cost, 448
- Performance bound, 709–710, 712
- Performance estimate, 607
- Performance ratio, 590–591, 595, 597, 604
- Permutation
 - cost of relative to, 501
 - cyclic, 310, 540
 - dense, 499
 - identity, 495
 - minimum cost, 538
 - product of, 495
 - random, 276
- Permutations, 228
 - cyclic, 704
- Perturbation, 210
- Pick-and-place machine, 10
- Pigeonhole principle, 283
- Points, 264
- Poisson random variable, 295
- Poisson
 - asymptotically, 263
- Poissonized version, 295
- Polyhedral norm, 560, 587–588, 599, 601
- Polyhedron
 - full dimensional, 39
 - graphical TSP, 129
 - unbounded, 41
- Polynomial 2-opt, 249
- Polynomial restriction, 254
- Polynomially equivalent, 532
- Polytope, 181, 566–567
 - monotone ATS, 128
 - 1-arborescence, 129
 - 2-matching, 39, 567, 569
 - AA, 145, 150
 - APA, 148
 - assignment, 129
 - ATS, 118, 150, 667, 681, 689, 691
 - subtour elimination, 668
 - valid inequality, 671
 - ATSP, 189
 - cycle-and-loops, 689, 694
 - cycle, 667, 686, 689, 691, 693
 - dimension, 36
 - E-GTSP, 618, 623
 - flat STS, 137
 - full dimensional, 39, 42
 - general 0-1, 135
 - GTSP, 620
 - knapsack, 667
 - monotone, 129
 - monotone ATS, 136
 - monotone STSP, 39
 - monotonization, 59
 - PCTS, 681, 689

- PCTSP, 673
- STS, 118
 - subtour elimination, 33–34, 44, 567, 573
 - symmetric TSP, 35
 - traveling salesman, 567
 - TSP, 189, 626
 - vertex packing, 146
- Population-based approach, 355
- Population-based methods, 313
- Portal, 212
 - parameter, 212
 - respecting tour, 212
- Portals, 217
- Postal routing, 615
- PQ-tree, 91
- Pre-processing, 505
- Precedence-constrained TSP, 663
- Precedence constrained TSP, 27
- Preferred attributes, 348
- Price out, 191
- Pricing, 107–108, 190–191, 193–194, 653
 - AP, 191
 - heuristic, 192
 - Lagrangian, 191
 - loop, 193
 - standard, 191–192
- Primal degeneracy, 191
- Primal dual, 173
- Primal solution, 190
- Principal submatrix, 507
- Priority queue, 391
- Prism, 570
- Prize-Collecting TSP, 610, 613–614
 - capacitated, 614
- Prize Collecting, 664
- Probability measure, 257
- Probability space, 605
- Property
 - Baki-II, 516–517
 - Baki-Kabadi-II, 517
 - Baki-Kabadi, 511
 - Baki-Kabadi III, 513
 - Baki-Kabadi IV, 513
 - G-G, 544
 - GG1, 522, 524
 - GG2, 522, 524
 - GG3, 523
 - GG4, 523, 536
 - max patching - II, 723
 - max patching I, 722
 - MPP-I, 722
 - MPP-II, 723
- Pseudo-node, 85, 101
- Pseudograph
 - patching, 722
- Pseudorandom, 451
- PTAS, 208
- Pyramidal
 - circuit, 500
 - path, 499, 508
 - permutation, 499
 - set-back tour, 581
 - solvability, 510, 513
 - tour, 502, 508, 514, 526, 533, 600
 - tours, 513
- Pyramidally solvable, 508–509, 511–514, 518, 523–525, 715, 718–720, 726
- Quad-neighbor list, 409
- Quadratic programming, 22
- Quadratic representation, 3
- Quadtree, 213
- Quasi-convex, 705
- Quasi-parallel, 551
- Quasi clone, 683, 686, 691
- Quasi clones, 681
- Quasi parallel, 553
- R-SAADP, 178, 180
- R-SAAP, 171, 178
- R-SADP, 172, 178, 180
- R-SAP, 171, 178
- Random dissection, 211
- Range, 519
- Raw data, 372
- Reduced cost, 108, 174–176, 178–179, 191–192
- Reduced costs, 193, 280
- Reference solutions, 365
- Reference structure, 322, 325
 - Hamiltonian path, 322
 - stem-and-cycle, 322, 335
- Refueling, 663
- Region, 519
- Regular graph
 - random, 272
- Regular graphs
 - random, 264
- Relaxation, 171
 - Held-Karp, 209
 - linear programming, 379
- Remote TSP, 27
- Residual cost, 178
- Resource constrained TSP, 26
- ROLL-A-ROUND, 666
- Rolling mills, 666
- Root node, 33
- Rooted tree, 278
- Rotation, 261
- Routing applications, 195
- Row generation, 33
- Row rank, 130
- Running time
 - normalized, 370, 381
- Rural Postman Problem, 541
- S-components, 562

- S-sum, 51
- Saturated set, 188
- Scatter Search, 355
- Scatter search, 356–357
- Scheduling, 535, 664
 - flow shop, 504
 - m-machine flow shop, 514
 - no wait, 504
 - robotic cell, 535
- SCSS bound, 704
- SEC, 119
- Seed node, 135
- Selective TSP, 26
- Selective TSP, 614
- Semi-regular curve, 575
- Semimetric problem, 586
- Separation, 190
- Separation problem, 82, 181, 627
- Separation procedure, 205
- Separation
 - 2-matching, 82
 - bipartition, 83
 - comb, 83
 - heuristic, 86
 - cover inequality, 650
 - domino, 83
 - heuristic, 85
 - simple comb, 82
 - subtour elimination, 82
- Sequencing, 578
- Sequential Fan tree, 340
- Sequential lifting, 668
- Serdyukov TSP, 26
- Shoelace problem, 4
- Short-cutting, 709
- Shortcut, 47
- Shortest path, 180, 233
- Shortest path condition, 137
- Shortest superstring, 587
- Shrinking, 86
- Shrinking procedure, 188–189
- Shrinking technique, 630
- Shrinkings, 85
 - legal, 85
- Sidelength, 210
- Simple chain, 500
- Simple cycle problem, 609
- Simplex method, 568
- Simulated annealing, 371
- Small cycles, 277
- Small vertex, 269
- Small vertices, 268
- Sorting network, 241
- Space smoothing, 339
- Spanner, 220
- Spanning closed walk, 40
- Spanning subgraph
 - biconnected, 710
- Spanning tree, 132, 321, 498, 503–504, 519–520, 530–531, 542, 544, 722
 - bottleneck, 709, 725
 - minimum, 630, 648
- Sparsification, 175–176
- Spherical volume, 603
- Star, 530
- Steel rolling mill, 664
- Steiner node, 541
- Steiner nodes, 541
- Steiner tree, 544
- Stem-and-cycle, 321, 338
- Stem-and-Cycle, 354, 360, 367
- Stem and cycle, 417, 421
- Stirling's formula, 233
- Stochastic convergence, 258
- Strategic oscillation, 319
- Strategic oscillation, 354
- Strong connectivity, 170, 179
- Strongly connected, 179, 579, 704
- Strongly Hamiltonian, 491
- Structuring matrices, 14
- STSP Challenge, 447
- Sub-additivity, 283–284, 295
- Sub-additivity, 295
- Sub-additivity
 - geometric, 283, 287
- Sub-cube, 283, 288, 292
- Sub-Gaussian tail, 302
- Sub-multi-tree, 531
- Sub-path, 532
- Subchains, 558
- Subgradient optimization, 636
- Submatrix
 - lower principal, 509
 - principal, 523, 536
 - upper principal, 509
- Submissive, 132
- Submoves, 341
- Subpath, 549
- Subpaths, 315, 319, 335
- Subroots, 336
- Subtour cover, 607
- Subtour elimination, 172
- Subtour merging, 173, 175
- Subtour
 - non-trivial, 498, 717
- Subtours
 - vertex disjoint, 171
 - arc disjoint, 544
- Super-additivity, 283–284, 287
 - geometric, 287, 299
 - strong, 289
- Supernodes, 335
- Superstring, 454
- Support graph, 85

- Support reduced, 130–131, 136
 - strongly, 131–132
- Surrogate constraints, 356
- Surrogate relaxations, 356
- Symmetric TSP, 169
- Tabu list, 351
- Tabu restrictions, 350
- Tabu Search, 345
- Tabu search, 350
- Tabu search, 353, 355, 371
- Tabu Search, 381, 385
- Tabu search
 - reactive, 351
- Tabu tenure, 351
- Tailing off, 110, 194
- Taxicab ripoff" problem, 601
- Taxicab ripoff problem, 585
- Teeth, 59, 61–62, 72, 78, 126, 675, 677
 - odd, 80
- Testbed, 310, 381, 449, 464
 - random, 372
 - random Euclidean, 373
 - clustered, 373
 - random matrix, 374
 - TSPLIB, 373
- Testbeds, 370, 372, 446–447, 450
- Threshold, 260
- Threshold algorithm, 701, 712, 714
 - generalized, 701
 - approximate, 714
- Threshold method, 175
- Threshold
 - Hamiltonicity, 260
- Time dependent TSP, 25
- Time window TSP, 27
- Tooth
 - degenerate, 68
- Total unimodularity, 543
- Tour merging, 436
- Tour
 - affinely independent, 36
 - compatible, 102
 - portal-respecting, 214–215
 - pyramidal, 727
- Tours
 - portal-respecting, 213
- Transformation, 195
- Translation invariance, 283
- Transportation problem, 578
- Transposition, 495
 - adjacent, 495, 501
- Traps, 44, 573
- Traveling purchaser problem, 617
 - capacitated, 617
- Traveling repairman problem, 25
- Tree-composition, 123
- Tree-like structure, 67
- Tree
 - 4-ary, 212
- Triagle inequality, 23
- Trial move, 329, 332
- Triangle inequality, 244, 508, 571, 596
 - parameterized, 23
- Troublesome vertices, 274, 276
- TSP Challenge, 310
- TSP polytope, 490, 610
 - dimension, 490
- TSPLIB, 34, 107, 411, 450
- TT-form, 49
- TT, 45
- Tunneling, 365
- Tunneling TSP, 558
- Twisted comb, 80
- Type GC, 532
- Ua-path, 525
- Upper bound, 174, 183, 197
- Upper minor, 509, 717
- Upper minors, 510
- Upperbound, 110, 112, 218
- V-compatibility graph, 624–625
- V-restriction, 624, 626
- V-tree, 572
- Valid inequality
 - primitive, 671
- Valley, 508
- Valleys, 718
- Variable-depth, 462
- Variable depth methods, 311
- Variable fixing, 107
- Variable fixing, 172, 193, 620
- Vehicle routing, 328
- Vertex
 - insertion, 228
 - k-critical, 52
 - removal, 227
- Violated cuts, 188, 193
- Vocabulary building, 350, 355
- Voronoi diagram, 572
- Wallpaper cutting, 536, 545
- Warehouse, 586
- Warehouse TSP, 598
- Weighted girth problem, 611
- Wheel, 562, 569, 731
- Whp, 258
- With high probability, 258
- Worst-case analysis, 257, 279
- Worst case analysis, 706
- Zero-lifting, 55
- Zero lifting, 104Q