Roma Tre University
Ph.D. in Computer Science and Engineering

Metaheuristic algorithms for multi-objective scheduling problems

Michele Ciavotta

Metaheuristic algorithms for multi-objective scheduling problems

A thesis presented by
Michele Ciavotta
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

February 2008

Committee:
Prof. Dario Pacciarelli

Reviewers:
Prof. Alessandro Agnetis
Prof. Rubén Ruiz

Alla mia Famiglia tutta ed a Valeria, senza il loro supporto nulla del mio lavoro avrebbe un senso

Abstract

Production optimization methods are widely used in manufacturing environments but very often just one optimization criterion is taken into account. Multi-objective optimization is without a doubt a very important research topic because of the multi-objective nature of most real-world problems and because there are still many open questions in this area.
It has been proved that exact approaches rarely work well for complex real-world cases because they often belong to the class of $\mathcal{NP}$-hard problems. Therefore the aim of this Ph.D thesis is to prove the effectiveness, adaptability and modularity of several metaheuristics applied to different real world difficult scheduling problems.
A large bibliographical section dealing with the problem of multi-objective permutation flowshop and parallel machines is presented.
Many algorithms described in literature have been re-implemented and new ones are presented. Finally computational campaigns have been performed and all results have been evaluated by means of statistical tools.

Acknowledgments

I am very grateful to all the people who have directly or indirectly contributed to the birth of this Ph.D. thesis. Special thanks to my tutor Prof. Dario Pacciarelli, to Prof. Carlo Meloni of "Politecnico di Bari" and Prof. Marco Pranzo "Università di Siena" colleagues and friends. I would also like to thank the members of *AUTORI* laboratory of the department of computer science and automation of university "Roma Tre" Rome. At last, I would like to thank Prof. Rubén Ruiz of "Universidad Politécnica de Valencia", Gerardo Minella an all the members of *SOA* group of "Instituto Tecnológico de Informática" of Valencia.

Contents

Contents viii

List of Tables x

List of Figures xii

1 Multi-objective scheduling optimization: an introduction 1
- 1.1 Introduction to Scheduling . . . 1
- 1.2 Optimality Criteria . . . 4
- 1.3 Scheduling with setups . . . 5
- 1.4 Parallel Machine Scheduling problem . . . 6
- 1.5 Flowshop scheduling problem . . . 6
- 1.6 Single-objective vs Multi-objective optimization . . . 7

2 Metaheuristics 15
- 2.1 Introduction to Metaheuristic algorithms . . . 15
- 2.2 Simulated Annealing . . . 17
- 2.3 Evolutionary Algorithms . . . 18
- 2.4 Iterated Local Search . . . 20
- 2.5 Iterated Greedy . . . 22
- 2.6 Tabu Search . . . 23
- 2.7 Variable Neighborhood Descent . . . 25
- 2.8 Rollout / Pilot Method . . . 27

3 Literature 29
- 3.1 Introduction . . . 29
- 3.2 Flowshop scheduling problems . . . 29
- 3.3 Multi-objective parallel machines scheduling problems . . . 43

3.4 Conclusions 46

4 Case Study 1: PFSP **49**
4.1 Introduction 49
4.2 Implemented algorithms 51
4.3 Multi-objective quality measures 57
4.4 Benchmark and computational evaluation details 59
4.5 Computational Evaluation 61
4.6 Comments and Conclusions 66

5 Case Study 2: PFSP with sequence-dependent setup times **81**
5.1 Introduction 81
5.2 Problem Description 82
5.3 Iterated Pareto Greedy 84
5.4 Experimental evaluation of the algorithm 89
5.5 Conclusions and future research 96

6 Case Study 3: Two-stage production system with 0-1 setups **117**
6.1 Introduction 118
6.2 Literature and applications 119
6.3 Problem description and formulation 120
6.4 Algorithms 127
6.5 Computational experiments 134
6.6 Conclusions and future research 138

7 Case Study 4: PMS problem with real-life constraints **139**
7.1 Introduction 139
7.2 Pharmaceutical manufacturing systems 141
7.3 Problem description 143
7.4 Solution methods 146
7.5 Computational results 151
7.6 Conclusions 158

8 Conclusions **161**

Bibliography **165**

List of Tables

3.1 Reviewed papers for the multi-objective flowshop. 42

4.1 Re-implemented methods for the multi-objective flowshop. 68

4.2 Details, operators and parameter values of the algorithms. 70

4.3 Results for the makespan and total tardiness criteria. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H. 71

4.4 Results for the total completion time and total tardiness criteria. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H. 72

4.5 Results for the makespan and total completion time. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H. 73

5.1 Re-implemented methods for the SDST multi-objective flowshop. . 92

5.2 Details and parameter of IPG algorithm. 93

5.3 Results for the makespan and total weighted tardiness criteria. Average quality indicator values for the 12 algorithms tested under the two different termination criteria. Instance group where setup times length is 50% that of the processing times. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H. 98
5.4 Results for the makespan and total weighted tardiness criteria. Average quality indicator values for the 12 algorithms tested under the two different termination criteria. Instance group where setup times length is 125% that of the processing times. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H. 107

6.1 Results of the three algorithms (60 seconds runs). 135
6.2 Results of the three algorithms (300 seconds runs). 135
6.3 Rank sum of (a) median computation times and (b) Pareto optimal front coverage. 137

7.1 Dispensing department: Instances with deadlines (H and H+LS) . 154
7.2 Dispensing department: Instances with deadlines (RH and RH+LS) 154
7.3 Counting department: Instances with deadlines (H and H+LS) . . 154
7.4 Counting department: Instances with deadlines (RH and RH+LS) 154
7.5 Dispensing department: Instances without deadlines (H and H+LS) 155
7.6 Dispensing department: Instances without deadlines (RH and RH+LS)155
7.7 Counting department: Instances without deadlines (H and H+LS) 156
7.8 Counting department: Instances without deadlines (RH and RH+LS)156

List of Figures

1.1 Gantt chart for a multiple machines scheduling problem 3
1.2 Example of Pareto dominance 12
1.3 Example of a Pareto set 13

2.1 Basic SA Algorithm 18
2.2 Basic EA Algorithm 20
2.3 Basic ILS Algorithm 21
2.4 Basic IG Algorithm 23
2.5 Basic TS Algorithm 25
2.6 Basic VND Algorithm 26
2.7 Basic Rollout/Pilot Algorithm 28

4.1 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria. 74
4.2 Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria. 75
4.3 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria. 76
4.4 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total tardiness criteria. 77

4.5 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment for the instance group 50×5. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total tardiness criteria. 78

4.6 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Total completion time and total tardiness criteria. 79

4.7 Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total completion time criteria. 80

5.1 The figure represents a schematic flow chart of IPG 86

5.2 Modified Crowding Distance Assignment Procedure(MCDA) . . . 87

5.3 The figure represents an asymmetric neighbour for the LS. 89

5.4 Local Search procedure (LS). 90

5.5 First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 99

5.6 First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 100

5.7 First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 101

5.8 First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 102

5.9 First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 103

5.10 First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 104

5.11 First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 105

5.12 First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 106

5.13 Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 108

5.14 Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 109

5.15 Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 110

5.16 Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 111
5.17 Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 112
5.18 Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 113
5.19 Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 114
5.20 Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria. 115

6.1 Geometric aspects of the feasible region of the *SUM-MAX* problem. 124
6.2 Geometric aspects of the solution space of the *NC-NS* problem. . 125
6.3 Bounds for the Pareto front for the *NC-NS* problem. 126
6.4 The COVER procedure. 130
6.5 The Sweep procedure. 132
6.6 The Fill procedure. 133

7.1 The main production phases in the secondary pharmaceutical manufacturing. 143
7.2 Algorithmic scheme of the Modified Jackson Schedule. 147
7.3 The scheme of the Algorithm Delta (Δ). 148
7.4 Algorithmic scheme of Pilot/Rollout 149
7.5 An illustration of the three moves. 150

7.6 Algorithmic scheme of Variable Neighborhood Descent (VND). . . 151
7.7 Average improvements in the dispensing department 157
7.8 Average improvements in the counting department 158

Chapter 1

Multi-objective scheduling optimization: an introduction

1.1 Introduction to Scheduling

Scheduling theory was introduced in the 50's [92] [98]. Since that time, more and more complex models, theorems and algorithms have been published.
In a nutshell a ***scheduling*** problem consists in the allocation of tasks (*jobs*) to resources (*machines*) in such a way that constraints are satisfied and certain goals are achieved.
Hence the elements that characterize a scheduling problem are :

- **J**, the set of n *jobs* that have to be processed
- **M**, the set of m *machines*
- **C**, the set of constraints that have to be satisfied
- **F**, the set of k (often $k = 1$) criteria to optimize

Depending on the type and number of machines, on the characteristics of jobs, constraints and criteria to optimize, different models of scheduling problems arise.
A job J_i consists of a number n_i of operation o_{ij} with $j = 1, \ldots, n_i$ and it is associated to a set of information, for example:

- *Release time*, $\mathbf{r_i}$
- *Processing time*, $\mathbf{p_{ij}}$
- *Due date*, $\mathbf{d_i}$
- *Deadline*, $\mathbf{D_i}$
- *Weight*, $\mathbf{w_i}$
- *Set of compatible machines for each operation* o_{ij} *of a job* J_i, $\mathbf{M^{ij}}$

Such information describes constraints and characteristics of the job. For example, a job J_i can not be scheduled before its release time and its operations can not be processed by machines not in M^{ij}. Processing times instead, describe how much time a process needs in a certain machine.
Solving a scheduling problem means finding the best *feasible schedule* according to a set of criteria and in turn, to find a feasible schedule means to assign to each operation o_{ij} of J_i, a time interval on a suitable machine, so that all constraints are satisfied.
Common constraints are, for example, observing release times and deadlines. This means the time interval assigned to a job on a machine can not start before the job release time and can not finish after the deadline. Another important constraint is preserving precedence relations among the jobs. Such relations mainly depend on technological constraints in the production and assembly of goods.
Schedules are represented by means of *Gantt charts* as shown in figure 1.1.
The ***machine environment*** is characterized by the type and number of machines, e. g.:

- *Single machine*
- *Parallel machines* (Identical, Uniform, Unrelated)
- *Shop environment* (Flowshop, Jobshop, Openshop)

The single machine case has been the subject of extensive research over since the early work of Jackson [92] in 1955. It can be considered the simplest class of scheduling problems. All jobs in J must be processed on the same machine. A classical example is the scheduling of several programs on the same CPU in a computer.
For some "lucky" cases a large number of theorems and properties have been

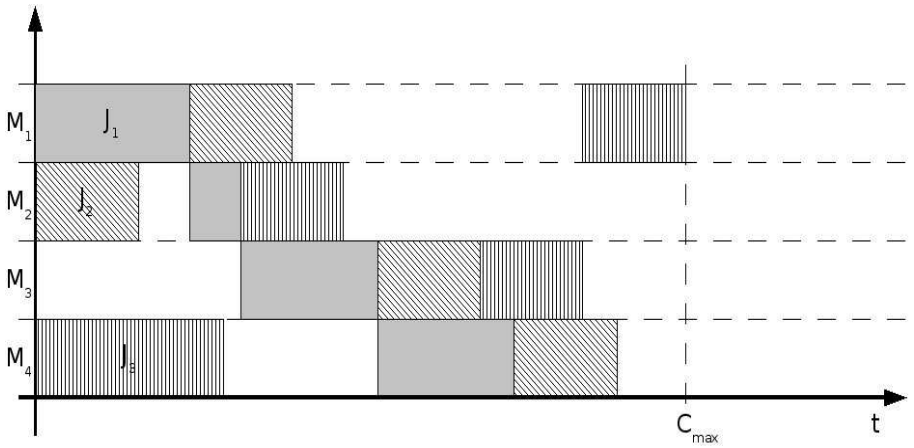

Figure 1.1: Gantt chart for a multiple machines scheduling problem

found and hence very effective algorithms have been developed but despite of the simplicity of the model many problems turn out to be $\mathcal{NP}$-hard and no efficient algorithm is known to solve them.
In the parallel machines case jobs have only one operation. Machines in M may be *identical* and this means that they need the same time to process the same operation, *uniform* processing times are different but dependent on the machine "speed", and *unrelated* i.e. processing times are different without any scale factor (see 1.4).
In the shop environment each job has more than one operation and for each operation generally a different machine is employed. Difference arises in the way the machine set M is visited. In the *flowshop* case each job has to visit all machines in the same order (see 1.5) while in *jobshop* a different but "a priori" known visit sequence is assigned to each job. In the *openshop* production is flexible and each job can be completed using whatever sequence of machines.
In this chapter, a basic description for the scheduling models for problems treated in this thesis will be provided in sections 1.4 and 1.5. More detailed models are presented for real-world case-studies addressed in chapters 4, 5, 6 and 7.

1.2 Optimality Criteria

Let denote the ***completion time*** of job J_i by C_i and its associated cost by $f_i(C_i)$, there exist essentially two types of total cost functions

$$f_{max}(C) := max_i\{f_i(C_i)|i = 1, \ldots, n\}$$

and

$$\sum f_i(C) := \sum_{i=1}^{n} f_i(C_i)$$

called **bottleneck** and **sum** objective respectively (see [26]).
The most common objective functions are:

makespan:	$max_i\{C_i\|i = 1, \ldots, n\}$	$f_i(C_i) = C_i$
total flow time:	$\sum_{i=1}^{n} C_i$	$f_i(C_i) = C_i$
total weighted flow time:	$\sum_{i=1}^{n} w_i C_i$	$f_i(C_i) = w_i C_i$

Other examples of cost functions $f_i(C_i)$ associated to J_i are:

lateness	$L_i :=$	$C_i - d_i$
earliness	$E_i :=$	$max\{0, d_i - C_i\}$
tardiness	$T_i :=$	$max\{0, C_i - d_i\}$
unit penalty	$U_i :=$	$\begin{cases} 1 & if\ C_i \leq d_i \\ 0 & otherwise \end{cases}$

As for $f_i(C_i) = C_i$ also with these functions we get at least three objectives.
Common objective functions are:

maximum lateness	$L_{max} :=$	$max_i\{L_i\}$
total (weighted) tardiness	$T(W)T :=$	$\sum_{i=1}^{n}(w_i)T_i$
total number of tardy jobs	$U :=$	$\sum_{i=1}^{n} U_i$

An objective function which is nondecreasing with respect to all variables C_i is called ***regular***. Notice that functions dealing with E_i are usually not regular. Regular functions have often properties that can lead to effective solution methods.

1.3 Scheduling with setups

Setup includes tooling, handling work-in-process, cleaning machines and work spaces, loading programs into machines and inspecting materials [141].
The majority of scheduling research assumes setups are negligible or included into processing times. Such assumption clearly simplifies the analysis and sometimes it is justified by the presence of long processing times and short setup times. On the contrary it adversely affects the solution quality for many problems that require treating explicitly setup times or costs.
The setup operations have for long been considered negligible and hence ignored, considered, for example, as part of the processing time. Although this may be somewhat justified for some scheduling problems, other situations call for explicit setup consideration.
Approaching a problem with setup means, above all, to understand how the presence of setups affect the production costs. In many cases it is sufficient consider only setup times. This occurs when setup cost is directly proportional to setup time. This is typically true when the cost is limited only to machine idle time. However, there are other situations, where the cost is relatively high for switching between certain jobs even though their switching time is relatively low.
A first classification highlights two classes of setup. In the first class, setup depends only on the job to be processed, hence it is called *sequence-independent*. In the second, setup depends on both the job to be processed and the immediately preceding one, hence it is called *sequence-dependent*. While in several cases sequence-independent setup times could be included in the processing times, the case of sequence-dependent setup times calls for an explicit treatment in models and algorithms.
A second possible classification looks at the possibility to anticipate a setup on a machine. Hence there exist *anticipatory* setups when it is possible to anticipate the execution of the setup with respect to the job and *not anticipatory* setups otherwise. Cleaning operations are a clear example of an anticipatory setup because the machine might remain idle after the cleanup. Time required to position a job on the machines is instead a obvious case of not anticipatory

setup.
The importance of setup times has been investigated in several studies. In a survey of industrial management, Panwalker et al. [143] discovered that about three quarters of the managers reported at least some operations they schedule require sequence-dependent setup times. Worthman [214] underlines the importance of considering sequence-dependent setup times for the effective management of manufacturing capacity.
For a comprehensive literature review dealing with setups see [10] and [11] .

1.4 Parallel Machine Scheduling problem

In the classical Parallel Machine Scheduling (PMS) problem, there are n jobs and m machines. Each job must be executed on one of the machines. the aim is to find the schedule that optimizes certain performance measure (in the single objective optimization) or to find a solution set which in some sense "optimize" a set of criteria (Weighted sum, Lexicographical order, Pareto optimization).
This scheduling problem involves two kinds of decisions, *job-machine assignment* (deciding on what machine a job must be processed), and *sequencing* (deciding the position in the queue).
Machines may be all *identical*, i.e. they would employ the same time to process the same job, *uniform* when they would employ times dependent on the own "speed" to process the same job otherwise they are *unrelated*.
The complexity usually grows exponentially with the number m of machines, making the problem intractable. This problem belongs to the class of Combinatorial Optimization problems, many of which are known to be $\mathcal{NP}$-hard [42], [100], [65] and [144]. What this means is that it is not likely that there exist polynomial time algorithms to solve them.
Many real-life problems can be modeled as PMS ones. On production lines, it is common to find more than one machine of each kind carrying out the production tasks. The PMS also constitutes an important issue in the field of Computer Science, due to the increments in use multiprocessor computers, which require procedures for assigning to a CPU tasks and then establish its priority.

1.5 Flowshop scheduling problem

A Flowshop Scheduling Problem (FSP) is characterized by n jobs and m machines. Each job must be processed by all machines. This model represents an

unidirectional flow of work within a production environment. In fact every job has to visit machines in the same sequence. This is the case, for instance, of production lines where similar products are manufactured.
In the general FSP each machine in the line may process a different sequence of jobs. This occurs when between two consecutive production stages there exists a temporary storage area and handling systems which can change the sequence of jobs. Hence the number of possible solutions of this problem is $(n!)^m$ and like PMS it becomes intractable even with a few jobs and machines.
In Permutation FSP (PFSP) each machine must process the same sequence of jobs. This is a clear simplification of the general problem which leads to a solution space of cardinality $n!$. Although the solution space is reduced respect to the general model, this problem remains still intractable for real-life instance sizes. This calls for an heuristic or metaheuristic approach. For more details see [75], [153] and [197].

1.6 Single-objective vs Multi-objective optimization

In the scheduling literature, most of the research works only deal with single objective problems, but many real-world scheduling problems are multi-objective by nature, i.e. several objectives should be optimized at the same time [15], [48]. Examples of such objectives are optimization of two or more of the following measures simultaneously, i.e. makespan (C_{max}), total flowtime (F), maximum tardiness (T_{max}), total tardiness (TT) as well as number of tardy jobs (U). Makespan (C_{max}) and total flowtime (F) are related to maximization of system utilization and minimization of work-in-process inventories respectively, while the remaining measures are related to job due dates. In this research we use makespan, total flowtime and total tardiness as the multiple objectives for a flowshop scheduling problem.
Since the early 50's, scheduling models have became more and more complex in order to better describe different practical situations. Although many real world problems have been well characterized, such models often employ a simplification: evaluation a solutions with respect to only one criterion. In fact, the vast majority of the papers on scheduling deals with problems in which the quality of a solution is estimated in terms of a single objective.
In production planning and scheduling, however, quality is a multidimensional concept. A decision maker, for instance, must evaluate production schedules on the basis of a number of criteria, as e.g. *work-in-progress inventories* and *observance of due dates*. If just one objective is taken into account, the the

outcome is likely to be unbalanced, independently of what criterion is considered. If he or she decides to maintain low work-in-progress inventory levels, some products are likely to be completed far beyond their due dates, while if the main goal is to have a production which respects due dates as much as possible, the consequence is that large work-in-progress inventories have to be handled. Hence in order to reach an reasonable trade-off, one has to measure the quality of a solution on the basis of more than one important criterion.
An important drawback of considering such problems lies in the difficulty of defining an appropriate notion of optimality and, given such notion, finding an optimal solution. Obviously, the situation becomes more complicated when more criteria are involved, unless the criteria are not in conflict with each other; roughly speaking, two criteria are not in conflict if a solution that performs well on one criterion is likely to perform well on the other one. If the objectives conflict, then the different solutions have to be weighted against each other. To that end, various options exist. The first one is to specify an upper bound on the value of the most important criterion: a solution is then selected that performs well on the other criteria while satisfying the bound. The second option is to aggregate the criteria into a single objective function; a solution in chosen that is optimal for this objective function. The third option is based upon an interactive version of decision making: an analyst determines a candidate solution and presents it to a decision maker, who either decides to accept it or tells the analyst on which criterion the score should be improved. Unfortunately, the determination of n candidate solutions takes more time than solving n times one of the basic single-criterion problems; sometimes it is not even possible to guarantee that one reasonable candidate solution is found In a reasonable amount of time. It is of great importance to know beforehand what the consequences are of taking extra criteria into account. If it is difficult to find a good set of candidate solutions, then one might prefer to look for a solution of somewhat lesser quality that is more easily obtained.
An important issue concerns the question of what constitutes a representative set of candidate solutions. An obvious choice is the set of all *nondominated* solutions. A solution is said to be nondominated if it outperforms any other solution on at least one criterion. If the number of nondominated solutions is large, then an analyst may impose extra restrictions upon the set of candidate solutions; for example, he or she can impose an upper bound on the value of a criterion. Over the years there have been several approaches used to deal with the multi-objective problems. Traditionally, the most common approach has been to aggregate functions according to the preferences set by the decision makers and then to find a solution that satisfies these preferences. In the re-

mainder of the chapter three multi-objective approaches, useful to understand my research work, are presented.

Weighted sum optimization

This is the simplest way to tackle a multi-objective problem and it uniquely consists in reducing the objective vector into a new single performance measure. This is an "a priori" approach because the Decision Maker first has to establish the relative importance of each objective function. He or she has to assign to each criterion a weight λ_i with $\sum_i \lambda_i = 1$ and a combination (often *linear* see 1.1) of the objective functions is optimized using a single-objective method.

$$f_{single}(s) = \sum_i \lambda_i f_i(s) \tag{1.1}$$

Hence for a solution the objective vector is calculated and then each criterion is multiplied by its weight and summed with the others, attaining in this way an unique objective to minimize.
This approach, in some cases, can be converted in a Pareto search by slightly modifying the weights and repeating the search. By running several times the solution procedure (changing each time the search direction), in fact, it is possible to approximate the global Pareto optimum set.
This method may lead to some problem if objectives have different magnitudes and notice, however, that it turns out often to be less effective and much slower when compared to multi-objective Pareto methods.

Lexicographical optimization

Another simple way to bring a multi-objective problem back to a single-objective one is the so-called *lexicographical optimization*. Like the weighted sum optimization this is an "a priori" approach to solve multi-criteria problems. Also this time the Decision Maker has to establish priorities among criteria. No weights are employed but only he or she imposes a priority ordering on the objective set.
Using this approach there does not exist any incomparability among solutions. Let s and $s^{'}$ be two different solutions for a certain Multi-Objective Optimization Problem (MOOP) and $V(s)$ and $V(s^{'})$ vectors containing in a fixed lexicographical order the values of each criterion for s and $s^{'}$. We say s is better than $s^{'}$ if $\exists\ i|\ V[i](s) \triangleleft V[i](s^{'})$ and for all $j < i\ V[i](s) = V[i](s^{'})$.

Employing this comparison technique it is possible to use a single-objective optimization method. In such a way the algorithm optimizes the first and most important criterion and when solutions with the same value of such objective are found, it selects as new best solution the one having the best value for the second criterion; in case of parity the third one is considered and so on.

Pareto optimization

Single and multi-objective scheduling problems have been studied extensively. However, in the multi-objective case, the majority of studies use the simpler "a priori" approach where multiple objectives are weighted into a single one. As mentioned, the main problem in this method is that the weights or a priority for each objective must be given. The "a posteriori" multi-objective approach is more complex since in this case, there is no single optimum solution, but rather lots of "optimum" solutions. For example, given two solutions x_1 and x_2 for a given problem with two minimization objectives f_1 and f_2 and being $f_1(\cdot)$ and $f_2(\cdot)$ the objective values for a given solution. Is x_1 better than x_2 if $f_1(x_1) < f_1(x_2)$ but at the same time $f_2(x_1) > f_2(x_2)$? It is clear than in a multi-objective scenario, neither solution is better than the other. However, given a third solution x_3 we can say than x_3 is worse than x_1 if $f_1(x_1) < f_1(x_3)$ and $f_2(x_1) < f_2(x_3)$. In order to properly compare two solutions in a MOOP some definitions are needed. Without loss of generality, let us suppose that there are M minimization objectives in a MOOP. We use the operator $\lhd$ as "better than", so that the relation $x_1 \lhd x_2$ implies that x_1 is better than x_2 for any minimization objective.
[225] present a much more extensive notation which is later extended in [145] and more recently in [104]. For the sake of completeness, some of this notation is also introduced here:

Strong (or **strict**) **domination**: A solution x_1 is said to strongly dominate a solution x_2 ($x_1 \prec\prec x_2$) if:

$$f_j(x_1) \lhd f_j(x_2) \;\; \forall j = 1, 2, \ldots, M;$$

i.e. x_1 is better than x_2 for all the objective values.

Domination: A solution x_1 is said to dominate a solution x_2 ($x_1 \prec x_2$) if the following conditions are satisfied:

$$(1) f_j(x_1) \not\trianglerighteq f_j(x_2) \quad \text{for } j = 1, 2, \ldots, M;$$
x_1 is not worse than x_2 for all objective values

$$(2) f_j(x_1) \triangleleft f_j(x_2) \quad \text{for at least one } j = 1, 2, \ldots, M$$

Weak domination: A solution x_1 is said to weakly dominate a solution x_2 ($x_1 \preceq x_2$) if:

$$f_j(x_1) \not\trianglerighteq f_j(x_2) \text{ for all } j = 1, 2, \ldots, M;$$

x_1 is not worse than x_2 for all objective values.

Incomparable solutions: Solutions x_1 and x_2 are incomparable ($x_1 \| x_2$ or $x_2 \| x_1$) if:

$$f_j(x_1) \not\trianglelefteq f_j(x_2) \text{ nor } f_j(x_2) \not\trianglelefteq f_j(x_1) \text{ for all } j = 1, 2, \ldots, M$$

These definitions can be extended to sets of solutions. Being A and B two sets of solutions for a given MOOP, we further define:

Strong (or **strict**) **domination**: Set A strongly dominates set B ($A \prec\prec B$) if:

$$\text{Every } x_i \in B \succ\succ \text{ by at least one } x_j \in A$$

Domination: Set A dominates set B ($A \prec B$) if:

$$\text{Every } x_i \in B \succ \text{ by at least one } x_j \in A$$

Better: Set A is better than set B ($A \triangleleft B$) if:

$$\text{Every } x_i \in B \succeq \text{ by at least one } x_j \in A \text{ and } A \neq B$$

Weakly domination: A weakly dominates B ($A \succeq B$) if:

$$\text{Every } x_i \in B \succeq \text{ by at least one } x_j \in A$$

Incomparability: Set A is incomparable with set B ($A \| B$) if:

$$\text{Neither } A \succeq B \text{ nor } B \succeq A$$

Non-dominated set: Among a set of solutions A, we refer to the non-dominated subset A' such as $x^{A'} \in A' \prec x^A$ with $x^{A'} \neq x^A$ and $A' \subset A$.

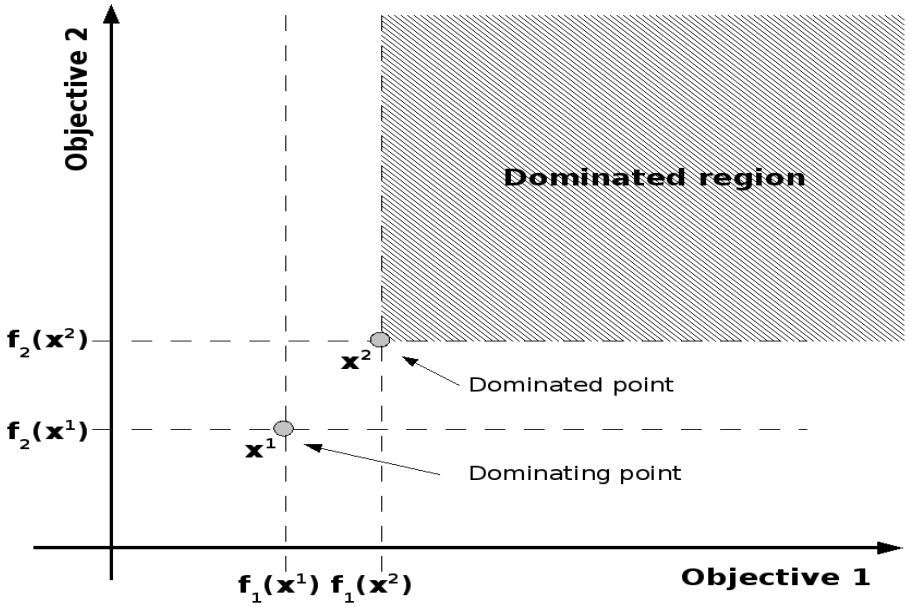

Figure 1.2: Example of Pareto dominance

Pareto global optimum solution: a solution $x \in A$ (being A a set of all feasible solutions of a problem) is a Pareto global optimum solution if and only if there is no $x' \in A$ such that $f(x') \prec f(x)$.

Pareto global optimum set: a set $A' \in A$ (being A a set of solutions of a problem) is a Pareto global optimum set if and only if it contains only and all Pareto global optimum solutions This set is commonly referred to as Pareto front.
Examples of dominance and Pareto set are given in figures 1.2 and 1.3.
Solving a MOOP means to find out the Pareto global optimum set or at least a nondominated solution set which approximates it adequately. The most important element in such process is time employed for finding optimal solutions. It might be extremely large and hence usually one choose a more functional trade-off between time and solution set quality This occurs because the hardness of an optimization problem mostly depends on solution space size if it is

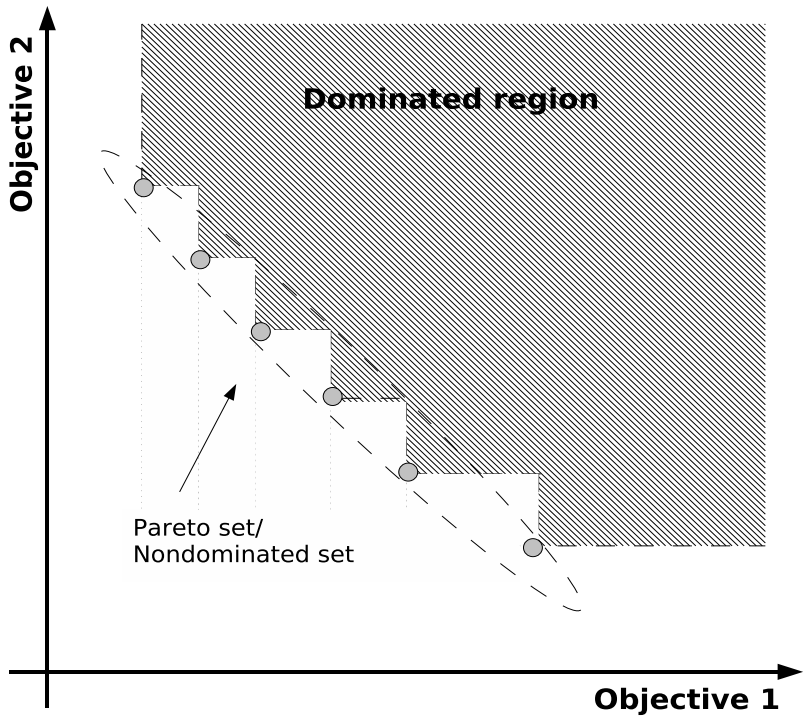

Figure 1.3: Example of a Pareto set

discrete and finite or on objective functions and constraints characteristics if such space has an infinite number of elements. Hence in several cases finding in an analytical way a solution is not possible and neither a full enumerative approach is conceivable. Consider for example, a scheduling problem where one has to sequence n elements, solution space in this case has $n!$ possible permutations. Such a problem quickly becomes intractable. Consider for example, $n = 20$ the number of possible solutions is $20! \approx 2,43 * 10^{18}$. Hence to sequence a small number of jobs one has to deal with a huge solution space.
Notice that comparing two Pareto sets which cross each other is not straightforward. In the last years this topic has been widely studied and several quality measures have been proposed [225], [145] and [104]. In section 4.3 quality

measures employed in this thesis are explained.

Chapter 2

Metaheuristics

2.1 Introduction to Metaheuristic algorithms

The name combines the Greek prefix *meta* ("beyond", here in the sense of "higher level") and *heuristic* (*heuriskein*, "to find") and it suggests a class of algorithms that are able, for a large class of optimization problems, to find near-optimal solutions working on a more general level.
In fact, while heuristic methods are invented and developed (when possible) to efficiently tackle a specific problem (usually exploiting a deep knowledge about it), metaheuristic algorithms are in many cases able, just holding a minimal understanding, to solve such a problem. In this sense, those methods are constituted by a *framework*, or a *general resolving scheme*, that can be hopefully applied to a wide set of optimization problems with relatively good results.
Beside the great advantage of being easily adaptable to many different problems, due to their scarce specific-problem information requirements (thus reducing design and implementation times), the main drawback of such methods consists in the fact that they are often much slower and less efficient with respect to *ad hoc* algorithms. For this reason, metaheuristics are generally applied to problems for which there is not any satisfactory known problem-specific exact algorithm or heuristic. This is why often metaheuristics use within their framework heuristic methods as black-boxes in order to have a trade-off between generality and performance. Moreover, for the same reason such algorithms are often hybridized with local search procedures.

Most commonly used metaheuristics are targeted to combinatorial optimization problems where the search space is extremely large and generally heuristic methods return solutions of low quality. All the scheduling problems considered during my Ph.D. experience fall into that class.
For the majority of the most effective metaheuristic approaches presented in literature it is possible to identify clearly the following two main phases:

- *Intensification*, that is the phase that tries to improve actual solution(s)
- *Diversification*, that is the phase that guide the search towards the exploration of different zones of the search space. In this way the algorithms can easily escape from local optima

In the following sections the main elements of the algorithms used in this Ph.D. thesis are discussed. Metaheuristics often belong to the class of stochastic methods. We implemented and employed the multiobjective versions of the following methods:

- *Simulated Annealing*
- *Evolutionary Algorithms*
- *Iterated Local Search*
- *Iterated Greedy* (an *Iterated Pareto Greedy* has been developed and presented for the first time in this Ph.D. thesis)

but we made use also of deterministic metaheuristic procedures:

- *Tabu Search*
- *Rollout/Pilot Method*
- *Variable Neighborhood Search*

Notice here that stochastic algorithms may potentially present different solutions in different runs. That is why it is a very common procedure to average results when using this class of procedures and why probabilities of success, percentages of search extension, normalized mean error, etc... are normally used for describing their behavior.

2.2 Simulated Annealing

Simulated Annealing (SA) is a generic stochastic metaheuristic algorithm. It was independently presented by Kirkpatrick et al. in 1983 [102], and by Černý in 1985 [32]. It originated as a generalization of a Monte Carlo method invented by Metropolis et al in 1953 [130]. This algorithm aims to emulate the annealing process in metallurgy, i.e. a technique that makes use of a heating phase followed by a cooling phase to produce crystals of bigger sizes and less defects. In fact the heating process allows the atoms to leave their initial positions in the crystals (a local minimum of the internal energy) and move randomly, the slow cooling phase gives them the possibility of finding configurations with lower internal energy respect to their initial status.
SA is commonly considered the first developed metaheuristic algorithm, moreover it was one of the first algorithms that proposed an effective scheme to avoid to get stuck in local minima. The basic idea consists in allowing moves which result in solutions of worse quality than the actual solution (*uphill moves*) in order to run away from local minima. Moreover the probability of doing such moves is not constant but it decreases during the search.
In figure 2.1 a simple pseudocode for a generic SA algorithm is presented . It starts by generating an initial random or heuristically constructed solution. The so-called *temperature* parameter T is then initialized; T will control the acceptance probability of worse quality solutions. The main loop consists in the random sampling of a candidate solution $s^{'}$ from the neighborhood $Neigh(s)$ of the actual solution s. $s^{'}$ is evaluated and it is accepted as new actual solution depending on the values of $f(s)$, $f(s^{'})$ and T. In fact $s^{'}$ replaces s if $f(s^{'}) < f(s)$ i.e. it has a better quality or, in case $f(s^{'}) >= f(s)$, with a probability that is generally computed following the Boltzmann distribution $e^{\frac{-(f(s^{'})-f(s))}{T}}$. Finally T is updated.
The main loop ends when a termination condition is met.
The temperature T decreases at the search process evolves, thus at the beginning the probability of accepting uphill moves is high and it gradually decreases, converging to a simple *iterative improvement algorithm*. Notice that the probability of accepting *uphill moves* depends also on the difference of the objective functions between the candidate and actual solution, in fact at fixed temperature, the higher the difference $f(s^{'}) - f(s)$, the lower the probability to accept a move from s to $s^{'}$ thus avoiding to accept actual solutions of very low quality even during the iterations of the algorithm.
It is easy to show how the effectiveness and the success of SAs was due to two

procedure Simulated Annealing Algorithm

$s := Generate\_Initial\_Solution()$
$T := T_0$
while termination conditions not met
$s^{'} := Rand(Neigh(s))$ % `candidate solution`
if $f(s^{'}) < f(s)$
$s := s'$
else
Accept $s^{'}$ as new solution with probability $e^{\frac{-(f(s^{'})-f(s))}{T}}$
end if
Update(T)
end while

Figure 2.1: Basic SA Algorithm

different strategies, *random walk* and *iterative improvement*, incorporated in the search process, Indeed during the first phase of the search, when T is high the algorithm permits a large exploration of the search space. This component slowly decreases with T thus leading the search to became an iterative improvement search and to converge to a possibly good local minimum.
Main advantages of this technique can be found in its proven capacity to converge to an optimum solution of a problem as well as in the easiness to transform a local search method in a simulated annealing algorithm usually having much better results. The main drawback consists in fact that SAs usually converge slower than other classes of algorithms. For more details see [1], [102],[96], [209] and [122].

2.3 Evolutionary Algorithms

The aim of evolutionary algorithms (EAs) is to mimic the the basic elements that are at the base of natural evolution. During the last decade they demon-

strated that their general framework is able to tackle and effectively solve many different class of problems. Although the deepest patterns of biological evolution are not completely understood, experimental evidence demonstrated how at the base of natural evolution there are the concepts of *chromosome*, *natural selection* and *reproduction*[131]. The evolution process operates over chromosomes rather than over organisms; chromosomes are encoding structures containing all the information essential for development and regulation of a living being. There exists a strict relation between chromosomes and the efficiency of the entity they represent. Natural selection is such a mechanism that allows those efficient organisms which are better adapted to the environment to reproduce more often than those which are not. The bulk of the evolutionary process consists in the reproduction stage. Faithful to the aim of Nature imitation, EAs coded a wide set of reproductive mechanisms. Most common ones are *mutation* that from a single individual (parent) produces an offspring with small differences in the chromosome and *recombination* that combines the chromosomes of the parents to generate off-springs. Based upon the features described above a large number of different EA have been developed. In a nutshell, an EA may be described as an iterative stochastic process that operates on a set of individuals often called *population* where each individual represents a possible solution for the considered optimization problem. An encoding/decoding mechanism transforms a possible solution into a chromosome. Frequently, the starting population is randomly generated but sometimes construction heuristics are employed to generate some good solutions with the aim of speeding up the evolutionary process.
The selection phase is performed using a fitness function which measures the degree of goodness of each individual in the population with respect to the problem under consideration. EA uses such quantitative information to guide the search towards promising zone of the search space. The most general structure of an EA is sketched in figure 2.2.
It can be noticed that the algorithm roughly is made up by three main phases: *selection*, *reproduction* and *replacement*.
The aim of the selection stage is to create a population set for reproduction where the fittest individuals (those corresponding to the solutions with better values of the fitness function) have a higher number of instances than the others in this way their promising genetic information have a greater probability to be preserved in future populations.
During the reproduction phase reproductive operators (*crossover*, *mutation*, etc...) are applied to the individuals in this population yielding a new population. Finally, during the last stage individuals of the the new created individ-

procedure Evolutionary Algorithm

Generate initial population $P(0)$
$t = 0$
while termination conditions not met **do**
$Evaluate(P(t))$
$P^{'}(t) = Selection(P(t))$
$P^{''}(t) = Reproduction(P^{'}(t))$
$P(t+1) = Replacement(P(t), P''(t))$
$t = t + 1$
end while
return best solution(s) found

Figure 2.2: Basic EA Algorithm

uals replace (partially or totally) those belonging to the previous generation. Usually the best individuals of the preceding population are maintained to avoid stochastic lose of promising chromosomes (elitism).
The whole process is repeated until a certain termination criterion is achieved, usually after a given number of iterations, after a certain amount of time or when reaching a solution near enough to a lower (upper) bound.
Notice that the EAs which have showed the best performances establishes a trade-off between *intensification* (selection phase) and *diversification* during the reproduction and replacement phases mechanisms are often used to keep a certain degree of genetic difference in the working population).
More details are reported in [86] [85] and [29]

2.4 Iterated Local Search

The Iterated Local Search algorithm (ILS) is a stochastic metaheuristic that holds tree desired characteristics: simplicity, generality and effectiveness.
The essence of the ILS metaheuristic can be described in a nutshell: one itera-

procedure Iterated Local Search Algorithm

s_0= GenerateInitialSolution
s^* =LocalSearch(s_0)
while termination condition is not met
do
  $s^{'}$ = Perturbation(s^*, history)
  $s^{*'}$ = LocalSearch($s^{'}$)
  s^* = AcceptanceCriterion(s^*,$s^{*'}$,history)
end do
return s^*

Figure 2.3: Basic ILS Algorithm

tively builds a sequence of solutions generated by a local search, leading to far better solutions than using repeated random restart of the same procedure.
This simple idea has a long history, and its rediscovery by many authors has lead to many different names like *iterated descent* [21] [22] [188], *large-step Markov chains* [126], *iterated Lin-Kernighan* [95] and *chained local optimization* [125]. Readers interested in the historical developments of these ideas should consult the review [97], while for a more detailed description of ILS see [119].

Figure 2.3 is an attempt to describe an ILS through is basic components. The algorithm starts generating an initial solution.
A Local search is then applied and an improved solution is attained. At this point the main loop begins. A condition is tested and a set of operations are repeated while a stopping criterion is not met. As in other metaheuristics this criterion is generally related to either the time elapsed or to the total number of iterations. Within this loop the first pass is the perturbation of the actual solution followed by a local search phase. In such a way the ILS tries to escape from local optima alternating an *intensification* (local search) and *diversification* phases (perturbation procedure).

Finally an acceptance criterion decides if the attained candidate solution, could be select as new actual solution.
In the most general version of such method a historical information may guide the search process while the local search procedure can be replaced by any improving procedure.
Despite the simple structure of an ILS methods it result to be very effective in solving hard optimization problems as, for example, TSP [21] [22] [101], Single Machine Total Weighted Tardiness Problem (SMTWTP) [40], Single and parallel machine scheduling [27] [28], Flow shop scheduling [187] [215], Job shop scheduling [118] [121] [106], Graph bipartitioning [124] [125], MAX-SAT [20] and Prize-collecting Steiner tree problem [30].

2.5 Iterated Greedy

Often for hard optimization problems heuristic methods have been developed that are able to find feasible solutions in short times. Those procedures, however, rarely return near optimal solutions. This is why metaheuristic frameworks as Iterated Greedy (IG) have been conceived.
Greedy procedures belong to the class of constructive methods and they generates, step by step, a complete solution for a considered problem passing through incomplete solutions.
The main idea is to use a procedure targeted to guide the search process iteratively calling a greedy heuristic. Thus if, for a certain problem, a greedy procedure is known, implementation of a IG algorithm is straightforward.
In few words, IG generates a sequence of solutions by iterating over greedy heuristics using two main phases: *Destruction* and *Construction*.
During the destruction phase some solution components are removed from a complete incumbent solution. The construction procedure, instead, applies a greedy constructive heuristic to reconstruct a new complete candidate solution.
By means of these two phases IG implements in an original fashion the *Intensification* (greedy reconstruction) and *Diversification* (destruction) phases that allow the search not be trapped in local optima. In fact the destruction phase if well designed has the ability to select a new promising zone to explore using the greedy heuristic.
Notice that if destruction turns out to be too strong the IG is equivalent to a random restart of the greedy heuristic while if it is too weak the algorithm is no more able to escape from local optima.
Once a candidate solution has been completed, an acceptance criterion decides

procedure Iterated Greedy Algorithm

s= GenerateInitialSolution
while termination condition is not met
do
$p^{'}$ = Destruction(s)
$s^{'}$ = GreedyConstruction($p^{'}$)
s = AcceptanceCriterion(s,$s^{'}$)
end do
return s^{*}

Figure 2.4: Basic IG Algorithm

whether the newly constructed solution will replace the incumbent solution.
IG iterates over these steps until some stopping criterion is met. In figure 2.4 is sketched the general structure of a IG.
IG is closely related to Iterated Local Search (ILS): instead of iterating over a local search as done in ILS [120], IG iterates in an analogous way over greedy heuristics. IG is a relatively new method but it has already been applied with success, for example, to the Set Covering Problem (SCP) [93] [123] and to the Permutation Flow Shop Problem (PFSP) [170] [171].
Notice finally, that the IG performances may be improved with the use of a local search procedures applied after the construction phase. In this way the difference between IG and ILS decreases further on.

2.6 Tabu Search

Tabu Search algorithm (TS) is a optimization method, belonging to the class of local search techniques. The main idea of TS consists in using memory structures to enhance the performance of a local search procedure and escape from local optima.
Although the primal idea of TS goes back to the 1970's, it was presented in

its actual form by Glover [69] in 1986. Moreover its basic ideas have also been suggested by Hansen [79] in the same year. A more precise formalization is reported in [70], [47] [71] and [73].
Even if there not exists a theoretical proof of its convergence to a global optimum, many computational evidences have shown that TS is one of the most used and promising metaheuristic techniques, able to compete with more recent and famous algorithms in many fields.
The general step of an *iterative procedure* consists in constructing from a current solution s a next solution$s^{'}$ and in checking whether one should stop there or perform another step.
A *Neighborhood Search* algorithm (NS) is an iterative procedure where, given an actual solution s, its neighborhood $Neigh(s)$ is explored and if a solution $s^{'}$ better than s (according to some objective function) is found, $s^{'}$ replaces s as new actual solution. This process is repeated until some stopping criterion is met or no better solution can be found in $Neigh(s)$.
To avoid to get stuck in a local optimum and explore larger regions of the search space, TS enhances an ordinary NS making use of a modified neighborhood structure that takes into account the search history. The solutions admitted to $Neigh^*(s)$ are determined through the use of simple or elaborate memory structures. The search then goes on by iteratively moving from a solution s to a different solution $s^{'}$ in $Neigh^*(s)$.
Although several types of short and long term memory structures have been proposed and developed, perhaps the first and most important type of short-term memory, is the so-called *tabu list*. In its simplest form, a tabu list contains the solutions that have been visited in the recent past (less than n moves before, where n is the size of the tabu list). Solutions in the tabu list are excluded from $Neigh^*(s)$.
Another way to implement the tabu list structure consists of storing only the information about certain attributes of visited solutions or prevent certain moves. Selected attributes in solutions recently visited are called *tabu-active*. Solutions that contain tabu-active elements are considered tabu. This type of short-term memory is often called *recency-based memory*. Tabu lists containing attributes demonstrated to be much more effective, although they suffer of a new drawback. When an attribute is forbidden, usually more than one solution turns out to be tabu, but in this way one prevents promising not yet visited path during the search.
To overcome this problem, *aspiration criteria* are introduced which allow overriding the tabu state of some solutions and include them in the allowed set $Neigh^*(s)$.

A commonly used aspiration criterion consists in admitting in $Neigh^*(s)$. solutions which are better than the currently best known solution. Figure 2.5 presents a simple TS described by means of pseudocode.

---

procedure Tabu Search Algorithm

k := 1
$s := Generate\_Initial\_Solution()$
$s^{best} := s$
while the termination criteria not met
Identify $Neigh(s)$. (Neighborhood set)
Identify $Tabu(s,k)$. (Tabu set)
Identify $A(s,k)$. (Aspirant set)
Choose best s' from $N(s,k) = N(s) - T(s,k) + A(s,k)$
$s := s^{'}$
if $f(s^{'}) < f(s^{best})$ **then**
$s^{best} := s^{best}$
end if
$k := k + 1$
end while

Figure 2.5: Basic TS Algorithm

2.7 Variable Neighborhood Descent

Variable Neighborhood Descent method is a relatively recent and simple metaheuristic framework (see [80] [81] which makes use of a set of different neighborhood to enhance the solution quality of a local search algorithm.
VND starts as a simple local search, but when a local optimum is identified a different neighborhood is selected allowing the algorithm to escape from such a point.
The basic idea is that a local optimum strictly depends on its neighborhood

and by changing it the search may be resumed.
If a better point is found the first neighborhood is restored and the search goes on, otherwise a different neighborhood is considered. The search stops when a solution that is the optimum for each neighborhood is found.
Steps of the basic VND are presented in Fig. 2.6. Let us denote with $Neigh_k$ with $(k = 1; ...; k_{max})$, a finite set of pre-selected neighborhood structures, and with $Neigh_k(s)$ the set of solutions in the kth neighborhood of s. The algorithm starts selecting the first neighborhood (i.e. $k = 1$) as long as new better solutions are found such neighborhood is maintained otherwise the subsequent is considered.
Notice that every time a new better solutions is encountered the first (and less time consuming) neighborhood is chosen again. Generally in fact the neighborhood structures are ordered in ascending order of their size. In such a way during the VND method uses more times small and fast neighborhood and only in few case the larger ones. This results in a fast and effective algorithm.

---

procedure Variable Neighborhood Descent

$s = Generate\_Initial\_Solution()$
$k = 1$
do
  $s^* = Neigh_k(s)$
  if s^* is better than s
    $s = s^*$
    $k = 1$
  else
    $k = k + 1$
until no improvement is obtained

Figure 2.6: Basic VND Algorithm

2.8 Rollout / Pilot Method

Combinatorial optimization problems are challenging in terms of finding optimal or near-optimal solutions. Often these problems turn out to be $\mathcal{NP}$-Hard. Thus the first (often the only) way to tackle them is developing quick heuristic procedures to construct approximate solutions in polynomial time making use of greedy rules. Such rules may have a myopic behavior making far from optimal choices. Generally such heuristics return feasible but not near-optimal solutions.
To enhance the solution quality of an existing heuristic, the so-called *pilot method* can be used; pilot method is a metaheuristic method that requires just little extra implementation effort and incorporates the existing heuristic as a building block.
The basic idea is a *look-ahead strategy* that exploits multiple iterations of the existing heuristic fixing, step by step, some elements of the final solution.
A similar type of look-ahead strategies has already been studied in the field of artificial intelligence. The idea is to examine all possible choices with respect to their probability to yield a future advantage cutting unpromising decisions, and choosing decisions that are most promising.
The pilot method was first conceived in the early 1990s [56]. Later, similar ideas were developed under different names. The most famous one is the *rollout method* by Bertsekas et al. in 1997 [25]. Applications are given in [25], and [129]. The latter paper proves that the rollout/pilot method could perform very well also in the difficult area of scheduling problems.
Consider a combinatorial optimization problem defined on a finite set of elements $ElemSet$ and a cost function $c : ElemSet \rightarrow \mathbb{R}$. The problem is to find a minimum cost feasible solution $S* \subset ElemSet$. Let be known a heuristic $Heur$ producing a solution which often is far from being optimal. Iteratively using $Heur$ as a *pilot heuristic*, the algorithm builds up a partial solution $MasterSol$, the *master solution*.
In order to escape from the greedy trap, the pilot method implements the following look-ahead strategy: Separately for each element $e \notin MasterSol$, the heuristic extends a copy of $MasterSol$ yielding complete solution in such a way that e is included.
Let $c(S(e))$ denote the value of the objective function of the complete solution obtained by $Heur$ for $e \in ElemSet \backslash MasterSol$, and let e_0 be the most promising element according to the heuristic, that is, $c(S(e_0)) \leq c(S(e))$ for all $e \notin MasterSol$. Element e_0 is included in the master solution $MasterSol$. Step by step a complete master solution is built up.

Different stopping criteria have been proposed, for example, the iterative process may stop when no improvements are found in the last iteration. The described algorithm is sketched in figure 2.7
The pilot method may be seen as a metaheuristic that proposes a system for heuristic repetition thus enhancing the quality of the solution returned by a single launch of such heuristic. Experiments demonstrated that the pilot method behaves competitively in comparison with other famous metaheuristics.

---

procedure Pilot Method

$k := |ElemSet|$
do

for $i = 1 : k$
$PilotSol_i = MasterSol \cup ElemSet[i] \ldots$
$\ldots \cup Heur(ElemSet \backslash ElemSet[i])$
end for

$(SelectedSol, Elem_Set[i]) = Best(PilotSol_i)$
$MasterSol = MasterSol \cup ElemSet[i]$
$ElemenSet = Elem \backslash ElemSet[i]$

while $|ElemSet| = 0$ or a termination condition is met

if $|ElemSet| > 0$ then **return** $SelectedSol$
else return $MasterSolution$

Figure 2.7: Basic Rollout/Pilot Algorithm

Chapter 3

Literature

3.1 Introduction

In this thesis we face four case studies belonging to the flowshop and parallel machines scheduling classes of problems. This chapter contains a complete and updated review of the literature for multi-objective flowshop problems which are among the most studied environments in the scheduling research area. No previous comprehensive reviews exist in the literature. Papers about lexicographical, goal programming, objective weighting and Pareto approaches have been reviewed. Exact, heuristic and metaheuristic methods have been surveyed. A comprehensive survey of multi-objective parallel machines literature is also presented.

3.2 Flowshop scheduling problems

Single and Multi-objective optimization

Single optimization criteria for the PFSP are mainly based on the completion times for the jobs at the different machines which are denoted by C_{ij}, $i \in M, j \in N$. Given a permutation π of n jobs, where $\pi_{(j)}$ denotes the job in the j-th position of the sequence, the completion times are calculated with the following expression:

$$C_{i,\pi_{(j)}} = \max\left\{C_{i-1,\pi_{(j)}}, C_{i,\pi_{(j-1)}}\right\} + p_{i\pi_{(j)}} \tag{3.1}$$

where $C_{0,\pi_{(j)}} = 0$ and $C_{i,\pi_{(0)}} = 0$, $\forall i \in M, \forall j \in N$. Additionally, the completion time of job j equals to C_{mj} and is commonly denoted as C_j in short.
By far, the most thoroughly studied single criterion is the minimization of the maximum completion time or makespan, denoted as $C_{max} = C_{m,\pi_{(n)}}$. Under this objective, the PFSP is referred to as $F/prmu/C_{max}$ according to [75] and was shown by [66] to be $\mathcal{NP}$-Hard in the strong sense for more than two machines ($m > 2$). Recent reviews and comparative evaluations of heuristics and metaheuristics for this problem are given in [61, 168] and in [84]. The second most studied objective is the total completion time or $TCT = \sum_{j=1}^{n} C_j$. The PFSP with this objective ($F/prmu/\sum C_j$) is already $\mathcal{NP}$-Hard for $m \geq 2$ according to [74]. Some recent results for this problem can be found in [58] or [163]. If there are no release times for the jobs, i.e., $r_j = 0, \forall j \in N$, then the total or average completion time equals the total or average flowtime, denoted as F in the literature.
Probably, the third most studied criterion is the total tardiness minimization. Given a due date d_j for job j, we denote by T_j the measure of tardiness of job j, which is defined as $T_j = \max\{C_j - d_j, 0\}$. As with the other objectives, total tardiness minimization results in a $\mathcal{NP}$-Hard problem in the strong sense for $m \geq 2$ as shown in [55]. A recent review for the total tardiness version of the PFSP (the $F/prmu/\sum T_j$ problem) can be found in [208]. In multi-objective optimization, two goals are usually aimed at. An approximation of the Pareto global optimum set is deemed good if it is close to this set. Additionally, a good spread of solutions is also desirable, i.e., an approximation set is good if the whole Pareto global optimum front is sufficiently covered.
One important question is concerned about the complexity of multi-objective flowshop scheduling problems. As mentioned above, the PFSP is already $\mathcal{NP}$-Hard under any of three commented single objectives (makespan, total completion time or total tardiness). Therefore, in a multi-objective PFSP, no matter if the approach is "a priori" or "a posteriori", the resulting problem is also $\mathcal{NP}$-Hard if it contains one or more $\mathcal{NP}$-Hard objectives.

Literature review on multi-objective optimization

The literature on multi-objective optimization is plenty. However, the multi-objective PFSP field is relatively scarce, specially when compared against the number of papers published for this problem that consider one single objective. The few proposed multi-objective methods for the PFSP are mainly based on evolutionary optimization and some in local search methods like simulated annealing or tabu search.

It could be argued that many reviews have been published about multi-objective scheduling. However we find that little attention has been paid to the flowshop scheduling problem. For example, the review by Nagar el al. (1995) [137] is mostly centered around single machine problems. As a matter of fact there are only four survey papers related with flowshop. In another review by T'Kindt and Billaut (2001) [196] we find about 15 flowshop papers reviewed where most of them are about the specific two machine case. Another review is given by Jones et al. (2002) [99]. However, this is more a quantification of papers in multi-objective optimization. Finally, the more recent review of Hoogeveen (2005) [88] contains mainly results for one machine and parallel machines scheduling problems. The papers reviewed about flowshop scheduling are all restricted to the two machine case. For all these reasons, in this paper we provide a complete and comprehensive review about multi-objective flowshop. However, note that we restrict ourselves to the pure flowshop setting, i.e., with no additional constraints. In the following, we will use the notation of T'Kindt and Billaut (2002) [197] to specify the technique and objectives studied by each reviewed paper. For example, a weighted makespan and total tardiness bi-criteria flowshop problem is denoted as $F//F_l(C_{max}, T)$. For more details, the reader is referred to [196] or [197].

Lexicographical and ε-constraint approaches

Lexicographical approaches have been also been explored in the literature. Daniels and Chambers (1990) [46] proposed a constructive heuristic for the m machine flowshop where makespan is minimized subject to a maximum tardiness threshold, a problem denoted by $F/prmu/\varepsilon(C_{max}/T_{max})$. This heuristic along with the one of Chakravarthy and Rajendran (1999) [33] are compared with a method recently proposed by Framinan and Leisten (2006) [62]. In this later paper, the newly proposed heuristic is shown to outperform the methods of [46] and [33] both on quality and on the number of feasible solutions found. A different set of objectives is considered in [158] were the authors minimize total flowtime subject to optimum makespan value in a two machine flowshop. Such an approach is valid for the PFSP problem since the optimum makespan can be obtained by applying the well known algorithm of Johnson. Rajendran proposes a branch and bound (B&B) method together with some heuristics for the problem. However, the proposed methods are shown to solve 24 jobs maximum.
In [140] two genetic algorithms were proposed for solving the two machine bi-criteria flowshop problem also in a lexicographical way as in [158]. The

first algorithm is based in the VEGA (Vector Evaluated Genetic Algorithm) of Schaffer (1985) [175]. In this algorithm, two subpopulations are maintained (one for each objective) and are combined by the selection operator for obtaining new solutions. In the second GA, referred to as the weighted criteria approach, a linear combination of the two criteria is considered. This weighted sum of objectives is used as the fitness value. The same problem is studied by Gupta et al. (1999) [78] where a tabu search is employed. This algorithm is finely-tuned by means of statistical experiments and shown to outperform some of the earlier existing methods. Gupta et al. (2002) [76] present some local search procedures and three metaheuristics for a two machine flowshop. The methods developed are simulated annealing, threshold accepting and tabu search. The criteria to optimize are composed of several lexicographic pairs involving makespan, weighted flowtime and weighted tardiness. The proposed methods are compared against the GA of Nepalli et al. (1996) [140] and the results discussed.
Gupta et al. (2001) [77] proposed nine heuristics for the two machine case minimizing flowtime subject to optimum makespan, i.e., $Lex(C_{max}, F)$. The authors identify some polynomially solvable cases and carry out a comprehensive analysis of the proposed heuristics. Insertion based methods are shown to give the best results. The same problem is approached by T'Kindt et al. (2002) [202] where the authors propose an ant colony optimization (ACO) algorithm. The method is compared against a heuristic from [201] and against other single objective methods form the literature. Although in some cases is slower, the proposed ACO method is shown to give higher quality results. T'Kindt et al. (2003) [201] work with the same problem. The authors propose a B&B method capable of solving instances of up to 35 jobs in a reasonable time. Some heuristics are also provided.

Weighted objectives

As mentioned, most studies make use of the "a priori" approach. This means that objectives are weighted (mostly linearly) into a single combined criterion. After this conversion, most single objective algorithms can be applied.
Nagar et al. (1995) [136] proposed a B&B procedure for solving a two machine flowshop problem with a weighted combination of flowtime and makespan as objective. The algorithm initializes the branch and bound tree with an initial feasible solution and an upper bound, both obtained from a greedy heuristic. This algorithm was able to find the optimal solutions of problems with two machines and up to 500 jobs but only under some strong assumptions and

data distributions. The same authors use this branch and bound in [138] as a tool for providing the initial population in a genetic algorithm. The hybrid B&B+GA approach is tested for the same two-job bi-criteria flowshop and it is shown to outperform the pure B&B and GA algorithms. Another genetic algorithm is presented in [184] for makespan and flowtime, including also idle time as a third criterion. The algorithm uses effective heuristics for initialization. Cavalieri and Gaiardelli (1998) [31] study a realistic production problem that they model as a flowshop problem with makespan and tardiness criteria. Two genetic algorithms are proposed where many of their parameters are adaptive. Yeh (1999) [216] proposes another B&B method that compares favorably against that of Nagar et al. (1995) [136]. For un-structured problems, Yeh's B&B is able to solve up to 14-job instances in less time than the B&B of [136]. The same author improved this B&B in [217] and finally proposed a hybrid GA in [218] showing the best results among all previous work. Note that all these papers of Yeh deal with the specific two machine case only. Lee and Chou (1998) [108] proposed heuristic methods and a mixed integer programming model for the m machine problem combining makespan and flowtime objectives. Their study shows that the integer programming approach is only valid for very small instances. A very similar work and results was given in a paper by the same authors (see [38]).
Sivrikaya-Şerifoğlu and Ulusoy (1998) [181] presented three B&B algorithms and two heuristics for the two machine flowshop with makespan and flowtime objectives. All these methods are compared among them in a series of experiments. The largest instances solved by the methods contain 18 jobs. A linear combination of makespan and tardiness is studied in [33] but in this case a Simulated Annealing (SA) algorithm is proposed. Chang et al. (2002) [34] study the gradual-priority weighting approach in place of the variable weight approach for genetic and genetic local search methods. These two methods are related to those of Murata et al. (1996) [135] and Ishibuchi and Murata (1998)[90], respectively. In numerical experiments, the gradual-priority weighting approach is shown superior. Framinan et al. (2002) [63] proposed several heuristics along with a comprehensive computational evaluation for the m machine makespan and flowtime flowshop problem. Allahverdi (2003) [8] also studies the same objectives. A total of 10 heuristics are comprehensively studied in a computational experiment. Among the studied methods, three proposed heuristics from the author outperform the others. Several dominance relations for special cases are proposed as well.
A different set of objectives, namely makespan and maximum tardiness, are studied by Allahverdi (2004) [9]. Two variations are tested, in the first one, a

weighted combination of the two objectives subject to a maximum tardiness value is studied. In the second, the weighted combination of criteria is examined. The author proposes a heuristic and compares it against the results of [46] and [33]. The proposed method is shown to outperform these two according to the results. Ponnambalam et al. (2004) [154] proposed a genetic algorithm that uses some ideas from the Traveling Salesman Problem (TSP). The implemented GA is a straightforward one that just uses a weighted combination of criteria as the fitness of each individual in the population. The algorithm is not compared against any other method from the literature and just some results on small flowshop instances are reported. Lin and Wu (2006) [115] focus on the two machine case with a weighted combination of makespan and flowtime. The authors present a B&B method that is tested against a set of small instances. The proposed method is able to find optimum solutions to instances of up to 15 jobs in all cases. Lemesre et al. (2007) [111] have studied the m machine problem with makespan and total tardiness criteria. A special methodology based on a B&B implementation, called two-phase method is employed. Due to performance reasons, the method is parallelized. As a result, some instances of up to 20 jobs and 20 machines are solved to optimality. However, the reported solving times for these cases are of seven days in a cluster of four parallel computers.

Pareto approaches

When focusing on the "a posteriori" approach the number of existing studies drops significantly. In the previously commented work of Daniels and Chambers (1990) [46], the authors also propose a B&B procedure for the C_{max} and T_{max} objectives that computes the Pareto global front for the case of two machines. A genetic algorithm was proposed by Murata et al. (1996) [135] which was capable of obtaining a Pareto front for makespan and total tardiness. This algorithm, referred to as MOGA, applies elitism by copying a certain number of individuals in the non-dominated set to the next generation. The non-dominated solutions are kept externally in an archive. The algorithm selection is based on a fitness value given to each solution on the basis of a weighted sum of the objective's values. The weights for each objective are randomly assigned at each iteration of the algorithm. The authors also test their proposed GA with three objectives including flowtime. Later, in [90] the algorithm is extended by using a local search step that is applied to every new solution, after the crossover and mutation procedures.
Sayin and Karabat (1999) [174] studied a B&B algorithm that generates the

optimum Pareto front for a two machine flowshop with makespan and flowtime objectives. The experimental evaluation compares only against heuristics like those of Johnson (1954) [98] and Rajendran (1992) [158]. Some instances of up to 24 jobs are solved to optimality. Liao et al. (1997) [114] proposed a B&B algorithm for the two machine bi-criteria optimization problem, with the objectives of minimizing makespan and number of tardy jobs and also with the objectives of makespan and total tardiness. The lower bound values are obtained by means of the Johnson algorithm for makespan, and the Moore's EDD (Early Due Date) algorithm for the number of tardy jobs. For each node of the partial schedules, two lower bounds are calculated using the above heuristics. The accepted non-dominated schedules are kept in an external set. At the end of the algorithm, this set contains optimal Pareto front for the problem. Lee and Wu (2001) [109] also study the two machine case with B&B methods but with a combination of flowtime and total tardiness criteria. The authors do not compare their proposed approach with the literature and just report the results of their algorithm. A new type of genetic algorithm is shown by Bagchi (2001) [16]. This method is based on the NSGA method by Srinivas and Deb (1994) [185]. Some brief experiments are given for a single flowshop instance with flowtime and makespan objectives. Murata et al. (2001) [134] improve the earlier MOGA algorithm of Murata et al. (1996) [135]. This new method, called CMOGA, refines the weight assignment. A few experiments with makespan and total tardiness criteria are conducted. The new CMOGA outperforms MOGA in the experiments carried out.

Ishibuchi et al. (2003) [91] present a comprehensive study about the effect of adding local search to their previous algorithm [90]. The local search is only applied to good individuals and by specifying search directions. This form of local search was shown to give better solutions for many different multi-objective genetic algorithms. In [116] many different scheduling problems are solved with different combinations of objectives. The main technique used is a multi-objective tabu search (MOTS). The paper contains a general study involving single and parallel machine problems as well. Later, in [117], a similar study is carried out, but in this case the multi-objective approach employed is the simulated annealing algorithm (MOSA).

A B&B approach is also shown by Toktaş et al. (2004) [203] for the two machine case under makespan and maximum earliness criteria. To the best of our knowledge, such combination of objectives has not been studied in the literature before. The procedure is able to solve problems of up to 25 jobs. The authors also propose a heuristic method. Suresh and Mohanasundaram (2004) [190] proposed a Pareto-based simulated annealing algorithm for makespan

and total flowtime criteria. The proposed method is compared against that of Ishibuchi et al. (2003) [91] and against an early version of the SA proposed later by Varadharajan and Rajendran (2005)[210]. The results, shown only for small problems of up to 20 jobs, show the proposed algorithm to be better on some specific performance metrics. J. Arroyo and Armentano (2004) [13] studied heuristics for several two and three objective combinations among makespan, flowtime and maximum tardiness. For two machines, the authors compare the heuristics proposed against the existing B&B methods of [46] and [114]. For the general m machine case, the authors compare the results against those of [63]. The results favor the proposed method that is also shown to improve the results of the GA of [135] if used as a seed sequence. The same authors developed a tabu search for the makespan and maximum tardiness objectives in [12]. The algorithm includes several advanced features like diversification and local search in several neighborhoods. For the two machine case, again the proposed method is compared against that of Daniels and Chambers (1990) [46] and for more than two machines against that of Ishibuchi and Murata (1998) [90]. The proposed method is shown to be competitive in numerical experiments. In a more recent paper Arroyo and Armentano (2005) [14] carry out a similar study but in this case using genetic algorithms as solution tools. Although shown to be better than other approaches, the authors do not compare this GA with their previous methods.
Makespan and total flowtime are studied by Varadharajan and Rajendran (2005) [210] with the help of simulated annealing methods. These algorithms start from heuristic solutions that are further enhanced by improvement schemes. Two versions of these SA (MOSA and MOSA-II) are shown to outperform the GA of [90]. Pasupathy et al. (2006) [148] have proposed a Pareto-archived genetic algorithm with local search and have tested it with the makespan and flowtime objectives. The authors test this approach against [90] and [34]. Apparently, the newly proposed GA performs better under some limited tests. Melab et al. (2006) [128] propose a grid-based parallel genetic algorithm aimed at obtaining an accurate Pareto front for makespan and total tardiness criteria. While the authors do not test their approach against other existing algorithms, the results appear promising. However, the running days are of 10 days in a set of computers operating as a grid. More recently, Rahimi-Vahed and Mirghorbani (2007) [157] have proposed a complex hybrid multi-objective particle swarm optimization (MOPS) method. The considered criteria are flowtime and total tardiness. In this method, a elite tabu search algorithm is used as an initialization of the swarm. A parallel local search procedure is employed as well to enhance the solution represented by each particle. This complex al-

gorithm is compared against the SPEAII multi-objective genetic algorithm of [223]. MOPS yields better results than SPEAII according to the reported computational experimentation albeit at a higher CPU time requirements. Finally, Geiger (2007) [67] has published an interesting study where the topology of the multi-objective flowshop problem search space is examined. Using several local search algorithms, the author analyzes the distribution of several objectives and tests several combinations of criteria.

Goal Programming and other approaches

There are some cases of other multi-objective methodologies like goal programming. For example, Selen and Hott (1986) [177] proposed a mixed-integer goal programming formulation for a bi-objective PFSP dealing with makespan and flowtime criteria. As with every goal programming method, a minimum desired value for each objective has to be introduced. Later, Wilson (1989) [213] proposed a different model with fewer variables but a larger number of constraints. However, both models have the same number of binary variables. The comparison between both models results in the one of [177] being better for problems with $n \geq 15$.
Many algorithms in the literature have been proposed that do not explicitly consider many objectives as in previous sections. For example, Ho and Chang (1991) [87] propose a heuristic that is specifically devised for minimizing machine idle time in a m machine flowshop. Although the heuristic does not allow for setting weights or threshold values and does not work with the Pareto approach either, the authors test it against a number of objectives. A similar approach is followed by Gangadharan and Rajendran (1994) [64] where a simulated annealing is proposed for the m machine problem and evaluated under makespan and flowtime criteria. Along with the SA method, two heuristics are also studied. Rajendran (1995) [160] proposes a heuristic for the same problem dealt with in [87]. After a comprehensive numerical experimentation, the new proposed heuristic is shown to be superior to that of Ho and Chang's. A very similar study is also presented by the same author in [159]. Ravindran et al. (2005) [164] present three heuristics aimed at minimizing makespan and flowtime. The authors test the three proposed method against the heuristic of [160] but using only very small instances of 20 jobs and 20 machines maximum. The three heuristics appear to outperform Rajendran's albeit slightly. It is difficult to draw a line in these type of papers since many authors test a given proposed heuristic under different objectives. However, the heuristics commented above were designed with several objectives in mind and therefore we have included

them in the review.
To sum up, Table 3.1 contains, in chronological order, the reviewed papers along with the number of machines (2 or m), the multi-objective approach along with the criteria as well as the type of method used. In total, 54 papers have been reviewed. Among them, 21 deal with the specific two machine case. From the remaining 33 that study the more general m machines, a total of 16 use the "a posteriori" or Pareto based approach. The results of these methods are not comparable for several reasons. First, the authors do not always deal with the same combination of criteria. Second, comparisons are many times carried out with different benchmarks and against heuristics or older methods. Last and most importantly, the quality measures employed are not appropriate as recent studies have shown.

Literature related to SDST flowshop

According to our knowledge there is no one article dealing with the multi-objective version of the permutation flowshop problem with sequence dependent setup times, hence in the following subsections we present two separated short reviews related to problems belonging to the class of single objective SDST flowshop. Compared to the regular flowshop, on which hundreds of paper have been published, the literature on the SDST counterpart is scarce. In a recent article, Ruiz and Maroto [168] carried out an extensive literature survey about this problem and here we summarize the outstanding aspects. Exact techniques for the SDST flowshop have shown rather limited results. Some heuristics and applications of stochastic local search algorithms to the SDST-FSP-C_{max} have also been proposed. Ríos-Mercado and Bard (1998) [165] proposed a modification of the well known NEH heuristic for the regular flowshop from [139] that considered setup times and they called NEH-RMB; in the same article they proposed a GRASP algorithm. In a later work, the same authors proposed a modification of the heuristics of Simons (1992) [180] resulting in a new heuristic called HYBRID [167]. Recently, Ruiz et al. [169] proposed a genetic and a memetic algorithm for the SDST-FSP-C_{max}. They carried out an extensive experimental study and compared these methods that were proposed for the FSP-C_{max}. The result was that both algorithms especially the memetic algorithm, are clearly superior to all other alternatives. On the SDST-PSP-WT, little has been published. In two similar articles [146], [147], a Simulated Annealing heuristic was proposed for the SDST flowshop problem with the objectives of minimizing the maximum weighted tardiness and the total weighted tardiness. In a more recent work, Rajendran and Ziegler (2003)

[162] introduced a new heuristic joined with a local search improvement scheme for a combined objective of total weighted flow-time and tardiness. Another similar work is [161] were only weighted flow-time is considered. Allahverdi et al. [11] have put together a much more updated and comprehensive review of scheduling research with setup times in which no other relevant papers related to the SDST flowshop can be found.

Year	Author/s	m	Approach and objectives	Comments
1986	Selen and Hott	m	$GP(C_{max}, F)$	Mixed-integer goal programming formulation
1989	Wilson	m	$GP(C_{max}, F)$	Mixed-integer goal programming formulation
1990	Daniels and Chambers	m	$\epsilon(C_{max}/T_{max})$	Heuristics
	Daniels and Chambers	2	$\#(C_{max}, T_{max})$	B&B
1991	Ho and Chang	m	(C_{max}, F)	Heuristics
1992	Rajendran	2	$Lex(C_{max}, F)$	B&B, heuristics
1994	Gangadharan and Rajendran	m	(C_{max}, F)	Simulated annealing. Heuristics
	Rajendran	m	(C_{max}, F)	Heuristics
1995	Nagar et al	2	$F_l(C_{max}, F)$	B&B
	Rajendran	m	(C_{max}, F)	Heuristics
1996	Murata et al.	m	$\#(C_{max}, T)$, $\#(C_{max}, T, F)$	Genetic algorithms
	Nagar et al.	2	$F_l(C_{max}, F)$	Genetic algorithms
	Nepalli et al.	2	$Lex(C_{max}, F)$	Genetic algorithms
			$\#(C_{max}, T, F)$	Genetic algorithms
	Sridhar and Rajendran	m	$F_l(C_{max}, F)$	Genetic algorithms
1997	Liao et al.	2	$\#(C_{max}, N_T)$, $\#(C_{max}, T)$	B&B
1998	Cavalieri and Gaiardelli	m	$F_l(C_{max}, T)$	Genetic algorithms
	Ishibuchi and Murata	m	$\#(C_{max}, T)$, $\#(C_{max}, T, F)$	Genetic algorithms
	Lee and Chou	2	$F_l(C_{max}, F)$	B&B
	Sivrikaya-Şerifoğlu and Ulusoy	2	$F_l(C_{max}, F)$	B&B, heuristics
1999	Chakravarthy and Rajendran	m	$F_l(C_{max}, F)$	Heuristics, integer programming

Year	Author/s	m	Approach and objectives	Comments
	Chou and Lee	2	$F_l(C_{max}, F)$	B&B
	Gupta et al.	2	$Lex(C_{max}, F)$	Tabu search
	Sayın and Karabatı	2	$\#(C_{max}, F)$	B&B
	Yeh	2	$F_l(C_{max}, F)$	B&B
2000	Loukil et al.	m	$\#(many)$	Tabu search. Many objectives studied
2001	Bagchi	m	$\#(C_{max}, F)$	Genetic algorithms
	Gupta et al.	2	$Lex(C_{max}, F)$	Heuristics
	Lee and Wu	2	$\#(F, T)$	B&B
	Murata et al.	m	$\#(C_{max}, T)$	Genetic algorithms
	Yeh	2	$F_l(C_{max}, F)$	B&B
2002	Chang et al.	m	$F_l(C_{max}, T)$, $F_l(C_{max}, T, F)$	Genetic algorithms
	Framinan et al.	m	$F_l(C_{max}, F)$	Heuristics
	Gupta et al.	2	$Lex(F, C_{max})$, $Lex(T, C_{max})$,	Various methods. Weighted functions
	T?kindt et al.	2	$Lex(C_{max}, F)$	Ant colony optimization
	Yeh	2	$F_l(C_{max}, F)$	Hybrid genetic algorithm
2003	Allahverdi	m	$F_l(C_{max}, F)$	Heuristics
	Ishibuchi et al.	m	$\#(C_{max}, T)$, $\#(C_{max}, T, F)$	Genetic algorithms and local search
	T?kindt et al.	2	$Lex(C_{max}, F)$	B&B, heuristics
2004	Allahverdi	m	$\epsilon(F_l(C_{max}, T_{max}), T_{max})$ $F_l(C_{max}, T_{max})$	Heuristics

Year	Author/s	m	Approach and objectives	Comments
	Armentano and Arroyo	m	$\#(C_{max}, T_{max})$	Tabu search
	Arroyo and Armentano	m	$\#(C_{max}, T_{max}, F)$	Heuristics, combinations of the three objectives
	Ponnambalam et al.	m	$F_l(C_{max}, F)$	Genetic algorithms
	Suresh and Mohanasundaram	m	$\#(C_{max}, F)$	Simulated Annealing
	Toktaş et al.	2	$\#(C_{max}, E_{max})$	B&B, heuristics
2005	Arroyo and Armentano	m	$\#(C_{max}, T_{max})$, $\#(C_{max}, T)$	Genetic algorithms
	Loukil et al.	m	$\#(many)$	Simulated annealing. Many objectives studied
	Ravindran et al.	m	(C_{max}, T)	Heuristics
	Varadharajan and Rajendran	m	$\#(C_{max}, F)$	Simulated annealing
2006	Framinan and Leisten	m	$\epsilon(C_{max}/T_{max})$	Heuristics
	Lin and Wu	2	$F_l(C_{max}, F)$	B&B
	Melab et al.	m	$\#(C_{max}, T)$	Parallel genetic algorithms
	Pasupathy et al.	m	$\#(C_{max}, F)$	Genetic algorithms
2007	Geiger2007	m	$\#(many)$	Local search
	Lemesre et al.	m	$F_l(C_{max}, T)$	B&B. Parallelism
	Rahimi-Vahed and Mirghorbani	m	$\#(F, T)$	Hybrid particle swarm optimization

Table 3.1: Reviewed papers for the multi-objective flowshop.

3.3 Multi-objective parallel machines scheduling problems

Although the parallel machine scheduling problem is one of the most studied in literature, only few papers concern the multi-objective version of such problem. In the following subsections we report several papers using the same classification we employed for FSP. Such papers are well known in literature and more information can be found in the reviews of T'Kindt and Billaut [196] [197], Allahverdi et al. [10] [11] and Pfund et al. [149]

Lexicographical and ε-constraint approaches

Lexicographical approaches have been widely explored in the literature. Leung and Young (1989) [112] proposed a $O(n\log(n))$ algorithm to optimally solve $P/pmtn/Lex(\bar{C}, C_{max})$ where $\bar{C}$ represents the average completion time. Such algorithm can be also used to find a lower bound for the problem without preemption that has been demonstrated to be $\mathcal{NP}$-Hard.
In 1995 Mc Cormick and Pinedo [43] proposed a polynomial algorithm ($O(m^3n)$) to solve $Q/pmtn/\varepsilon(\bar{C}, C_{max})$. Their procedure computes all the strict Pareto optimal points that correspond to extreme points of the solution set.
T'Kindt et al. in 1997 [199] propose a pseudo-polynomial time implementation in $O(n^4\log_2(C^*_{max}))$ for $P/pmtn, d_i/Lex(C_{max}, L_{max})$. Other two optimal algorithm for polynomially solvable parallel machines problems are presented by Tuzikov et al. in 1998 [206]. Authors study a scheduling problem where n jobs have to be scheduled on a set of machines with different speeds. In the first problem the aim is minimize $f_{max} = \max_{i=1,\dots,n}(\phi_i(C_i))$ subject to $g_{max} = \max_{i=1,\dots,n}(\psi_i(C_i))$ $(\varepsilon(f_{max}/g_{max}))$ while in the second one they optimize $\bar{g} = \sum_{i=1}^{n}\psi_i(C_i)$ subject to f_{max}. Notice that ψ_i and ϕ_i are generic increasing functions.
Mohri et al. (1999) [132] proposed an exact method to solve in polynomial time the two identical parallel machines problem where the maximum lateness is minimized subject to a makespan threshold, $P2/pmtn, d_i/\varepsilon(L_{max}/C_{max})$. In the same paper they extended their approach to polynomially solve the $P3/pmtn, d_i/\varepsilon(L_{max}/C_{max})$ problem.
Sarim and Hariharan (2000) [173] tackle the problem of scheduling n independent jobs on two parallel machines, when no preemption is allowed. Maximum tardiness and the number of tardy jobs are minimized in lexicographical way. Since $P2/d_i/T_{max}$ is $\mathcal{NP}$-Hard, the bicriteria problem is also. Therefore they propose an heuristic enumeration algorithm based on a branch and bound

scheme to solve it.
The problem $P/pmtn, d_i/Lex(L^w_{max}, \bar{C}^w)$ with $L^w_{max} = \max_{i=i,\ldots,n}(w_i L_i)$ has been addressed by T'Kindt et al. (2000) [198]. Such a problem is strongly $\mathcal{NP}$-Hard and the authors proposed a heuristics approach for it. The same authors in 2001 [200] are interested in a scheduling problem connected with the production of glass bottles. The aim is to minimize a linear combination of $I_{max} = C_{max} - C_{min}$ (the maximum difference of machines workload) and $\bar{M} = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{i,j} \times m_{i,j}$ (Total Margin) where $p_{i,j} = \dfrac{x_{i,j}}{k_{i,j}}$ is the processing time of job J_i on M_j and $x_{i,j}$ is the associated quantity of glass, subject to C_{max} bounded by a date ϵ. Preemption of jobs occurs at real times.A polynomial algorithm is presented.

Weighted objectives

The most direct way to consider several criteria is to reduce them in a new single performance measure. This is an "a priori" approach because Decision Maker first has to establish the relative importance of each objective function. Geoffrion and Graves (1976) [68] considered the case of identical parallel machines with sequence dependent setup times and developed a quadratic assignment formulation of the problem to minimize the sum of the changeover, production and time-constraint penalty costs.
The problem $P/d_i = d \geq \sum_{i=1}^{n} p_i, nmit/F_l(\bar{E}, \bar{T})$ with $F_l(\bar{E}, \bar{T}) = \bar{E} + \bar{T}$ belongs to the class $\mathcal{P}$ and Sundararaghavan and Ahmed (1984) [189] presented an polynomial algorithm for it. A more general version of the same problem has been tacked by Emmons (1987) [59] were $/F_l(\bar{E}, \bar{T}) = \alpha T + \beta E$ while the problem with machines having different non job dependent speed is also tackled by Emmons, however the complexity of this problem remains open.
Dietrich (1989) [54] considered the unrelated parallel machine problem with major and minor setups and developed a two-phase to minimize a linear combination of makespan and total flow time.
Alidaee and Ahmadian (1993) [7] studied $R_m/p_{i,j} \in [\underline{p}_{i,j}, \bar{p}_{i.j}]/F_l(\bar{C}, \bar{C}\bar{C}^w)$ and $R_m/p_{i,j} \in [\underline{p}_{i,j}, \bar{p}_{i.j}], d_i = d, d\ unknown/F_l(\bar{T}, \bar{E}, \bar{C}\bar{C}^w)$ where the objective function takes into account crashing costs. These problems can be reducted to the assignment problem and therefore they are polynomially solvable.
The problem $P/d_i = d \geq \sum_{i=1}^{n} p_i / \max_{1 \leq i \leq n}(w_i(E_i + T_i))$ is studied by Li and Cheng [113]. The authors show that this problem is strongly $\mathcal{NP}$-hard and propose an $O(mn^2)$ heuristic. The problem $R/d_i = d, d\ unknown, p_i = p, nmit/F_l(\bar{E}, \bar{T}, d)$ is polynomially solvable. Cheng and Chen [35] present an

algorithm to solve it.
$R_m/d_i = d/\sum(w_eE_j+w_tT_j)$ has been studied by Adamopoulos and Pappis [2]. This problem is $\mathcal{NP}$-Hard, therefore the authors chose to develop a polynomial time heuristic procedure that provides efficient solutions.
Heady and Zhu (1998) [83] addressed the $P/SDST/\sum(E_i+T_i)$ problem, where some machines may not be able to process some jobs. They proposed a heuristic to minimize the sum of earliness and tardiness costs for the problem. For small-sized problems, they also compared the performance of the proposed heuristic with the optimal solution obtained from using integer programming formulation. Sivrikaya-Şerifoğlu and Ulusoy (1999) [182] addressed the problem of $Q/SDST;rj/w_E\sum E_i + w_T\sum T_i$, where there are two types of machines with different speeds. Here $w_E\sum E_i + w_T\sum T_i$ means that the weights for earliness and tardiness penalties are common to all the jobs. The authors presented two types of genetic algorithms, namely one with a crossover operator and one without crossover operator. They showed that the genetic algorithm with a crossover operator performs better for difficult and large-sized problems. Balakrishnan et al. (1999) [17] considered the general case of uniform machines with objective function of minimizing $\sum(w_iE_i + w_iT_i)$. They presented a mixed integer programming formulation for the problem. Zhu and Heady (2000) [221] addressed the $R/SDST/\sum(w_iE_i + w_iT_i)$ problem. They developed a mixed integer programming formulation for the problem, which can provide an optimal solution in reasonable time for nine jobs and three machines.Bank and Werner (2001) [18] consider the same problem with the presence of release dates, they presented several constructive algorithms, an iterative algorithm and a neighborhood search technique testing them against simulated annealing.
Radhakrishnan and Ventura (2000) [156] addressed the $P/SDST/\sum(E_i + T_i)$ problem, presented a mathematical programming formulation that can be used for limited-sized problems, and proposed a simulated annealing algorithm for large-sized problems.
In paper manufacturing, Akkiraju et al. (2001) [5] observed a model generalizing the $R/BatchSDST$ problem with multiple objectives such as $\sum w_iT_i$, $\sum w_iE_i$, and TST. They suggested a heuristic approach based on the so-called Asynchronous Team architecture. Initial solutions are first generated by different experts and computer programs. Then these solutions are perturbed and improved. Finally, a set of Pareto optimal solutions is presented to a decision maker. Yi and Wang (2003) [219] considered the $P/BatchSIST/\sum w_iE_i + \sum w_iT_i$ problem, where the jobs have a common due date. They proposed a fuzzy logic embedded genetic algorithm (called soft computing) to solve

the problem. Recently Feng and Lau (2005) [60] addressed the more general $P/SDST/\sum(w_iE_i + w_iT_i)$ problem and proposed a meta-heuristic called Squeaky Wheel Optimization. The authors showed that their heuristic outperforms that of Radhakrishnan and Ventura.

Pareto approaches

In literature we found only few cases using a Pareto optimization approach in the context of parallel machine scheduling. Shmoys and Tardos (1993) [179] studied the bicriteria problem of minimizing C_{max} and the cost of the schedule considering the case where there is a range of possible processing times for each machine-job pair, and the cost linearly increases as the processing time decreases.
Suresh and Chaudhuri (1996) [191] studied $R//C_{max}, T_{max}$ problem. They proposed a tabu search algorithm where the initial solution is found using a heuristic called GAP/EDD.
The minimization of criteria $\bar{C}$ and $\bar{U}$ on m identical machines has been tacked by Ruiz-Torres et al . (1997) [172] who tackle the enumeration of strict Pareto optima. As the $P/d_i/\bar{U}$ is $\mathcal{NP}$-hard the bicriteria problem is also and the authors propose four heuristic to solve it.
Jansen and Porkolab (1999) [94] studied the bicriteria problem of minimizing C_{max} and the cost of the schedule and presented a modified version of their PTAS algorithm to solve it.
Yu et al. [220] tackle the $R//C_{max}, \sum C_j, \sum T, \sum U$. This problem arises from a printed wafer board manufacturing facility's drilling operation. To solve this problem, the authors developed a two-stage solution procedure, LRH. In the first step a heuristic assigns some jobs to machines while in the second phase a unimodular integer programming formulation is solved. Pfund et al. [150] extended this research to determine the robustness of various dispatching rules for the multi-criteria unrelated parallel-machine scheduling problem in presence processing time uncertainty and machine breakdowns. Two dispatching rules are presented in this work.

3.4 Conclusions

In this chapter we have conducted a comprehensive survey of the multi-objective literature for the flowshop and parallel machines, that are two of the most common and thoroughly studied problems in the scheduling field. This survey

follows and complements others like by those of [137], [196] [197] [88], [10] [11] and [149].

Chapter 4

An evaluation of multi-objective algorithms for PFSP

In this chapter a complete computational evaluation of multi-objective algorithms is carried out. A total of 23 different algorithms including both flowshop-specific methods as well as general multi-objective optimization approaches have been tested under three different two-criteria combinations with a comprehensive benchmark. All methods have been studied under recent state-of-the-art quality measures. Parametric and non-parametric statistical testing is profusely employed to support the observed performance of the compared methods. As a result, we have identified the best performing methods from the literature which constitutes a reference work for further research.

4.1 Introduction

In the field of scheduling, the flowshop problem has been thoroughly studied for decades. The flowshop problem is defined by a set of $N = 1, 2, \ldots, n$ jobs that have to be processed on a set of $M = 1, 2, \ldots, m$ machines. The processing time of each job $j \in N$ on each machine $i \in M$ is known in advance and is denoted by p_{ij}. All N jobs visit the machines in the same order, which, without loss of generality, can be assumed to be $1, 2, \ldots, m$. The objective is to find a processing sequence of the jobs so that a given criterion is optimized. In general, the number of possible solutions results from the product of all pos-

sible job permutations across all machines, i.e. $(n!)^m$ solutions or "schedules". However, in the flowshop literature it is very common to restrict the solution space by having the same permutation of jobs for all machines. The resulting problem is referred to as Permutation Flowshop Problem or PFSP in short where $n!$ schedules are possible.
The majority of the literature for the PFSP is centered around a single optimization criterion or objective. However, a single criterion is deemed as insufficient for real and practical applications. Multi-objective optimization is without a doubt a very important research topic not only because of the multi-objective nature of most real-world problems, but also because there are still many open questions in this area. Over the last decade, multi-objective optimization has received a big impulse in Operations Research. Some new techniques have been developed in order to deal with functions and real-world problems that have multiple objectives, and many approaches have been proposed.
The easiest way of dealing with a multi-objective problem is the so called "a priori" approach where two or more objectives are weighted and combined into a single measure, usually linear. For example, given two optimization criteria F_1 and F_2, a single-objective problem is derived with a combined $\alpha F_1 + (1-\alpha)F_2$ function, where $0 \leq \alpha \leq 1$. However, the major drawback in this approach is that α must be given a priori. If α is not known, an alternative is to obtain several solutions with varying values of α but even in this case, if F_1 and F_2 are measured in different scales, this approach might be challenging.
A more desirable approach is the "a posteriori" method. In this case, the aim is to obtain many solutions with as many associated values as objectives. In such cases, the traditional concept of "optimum" solution does not apply. A given solution A might have a better F_1 value than another solution B, but at the same time worse F_2 value. In this context, a set of solutions is obtained where their objective values form what is referred to as the Pareto front. In the Pareto front all solutions are equally good, since there is no way of telling which one is better or worse. In other words, all solutions belonging to a Pareto front are the "best" solutions for the problem in a multi-objective sense.
There are no comprehensive reviews in the literature about multi machine flowshops with several objectives. Hence I carried out this work and presented it in chapter 3. In the past years a number of interesting algorithms have been proposed. However, new proposals are hardly validated against the existing literature and when done, the quality indicators used are not appropriate. Additionally, the multi-objective literature is rich on advanced methods that have not been applied to the PFSP before. Therefore, an objective of this chapter

is also to adapt proposed methods from the general multi-objective optimization field to the PFSP. As we have seen in chapter 3, the literature is marred with different multi-objective proposals, many of which have not been tested for scheduling problems. As a result we identify the most promising methods for the most common criteria combination. We evaluate a total of 23 methods, from local search metaheuristics such as tabu search or simulated annealing to evolutionary approaches like genetic algorithms. Furthermore, we use the latest Pareto-compliant quality measures for assessing the effectiveness of the tested methods. Careful and comprehensive statistical testing is employed to ensure the accuracy of the conclusions.
This chapter is organized as follows: In section 4.2 there is a short description of all the re-implemented algorithms. The benchmark set and comparison evaluation details are presented in section 4.4. Section 4.5 deals with the comparative evaluation of all the studied algorithms. Finally in section 4.6 some conclusions and further research topics are given.

4.2 Implemented algorithms

In this work we have implemented not only algorithms specifically proposed for the multi-objective PFSP but also many other multi-objective optimization algorithms. In these cases, some adaptation has been necessary. In the following we go over the algorithms that have been considered.

Pareto approaches for the flowshop problem

We now detail the algorithms that have been re-implemented and tested among those proposed specifically for the flowshop scheduling problem. These methods have been already reviewed in section 3.2 and here we extend some details about them and about their re-implementation.
The MOGA algorithm of [135] was designed to tackle the multi objective flowshop problem. It is a simple genetic algorithm with a modified selection operator. During this selection, a set of weights for the objectives are generated. In this way the algorithm tends to distribute the search toward different directions. The authors also incorporate an elite preservation mechanism which copies several solutions from the actual Pareto front to the next generation. We will refer to our MOGA implementation as MOGA_Murata. Chakravarthy [33] presented a simple simulated annealing algorithm which tries to minimize the weighted sum of two objectives. The best solution between those generated

by the Earliest Due Date (EDD), Least Static Slack (LSS) and NEH methods is selected to be the initial solution. The adjacent interchange scheme (AIS) is used to generate a neighborhood for the actual solution. Notice that this algorithm, referred to as SA_Chakravarty, is not a real Pareto approach since the objectives are weighted. However, we have included it in the comparison in order to have an idea of how such methods can perform in practice. We "simulate" a Pareto approach by running SA_Chakravarty 100 times with different weight combinations of the objectives. All the 100 resulting solutions are analyzed and the best non dominated subset is given as a result.
Bagchi [16] proposed a modification of the well known NSGA procedure (see next section) and adapted it to the flowshop problem. This algorithm, referred to as ENGA, differentiates from NSGA in that it incorporates elitism. In particular, the *parent* and *offspring* populations are combined in a unique set, then a non dominated sorting is applied and the 50% of the non dominated solutions are copied to the *parent* population of the following generation.
Murata [134] enhanced the original MOGA of [135]. A different way of distributing the weights during the run of the algorithm is presented. The proposed weight specification method makes use of a cellular structure which permits to better select weights in order to find a finer approximation of the optimal Pareto front. We refer to this later algorithm as CMOGA.
Suresh [190] proposed a Pareto archived simulated annealing (PASA) method. A new perturbation mechanism called "segment-random insertion (SRI)" scheme is used to generate the neighborhood of a given sequence. An archive containing the non dominated solution set is used. A randomly generated sequence is used as an initial solution. The SRI is used to generate a neighborhood set of candidate solutions and each one is used to update the archive set. A fitness function that is a scaled weighted sum of the objective functions is used to select a new current solution. A restart strategy and a re-annealing method are also implemented. We refer to this method as MOSA_Suresh.
Armentano [12] developed a multi-objective tabu search method called MOTS. The algorithm works with several paths of solutions in parallel, each with its own tabu list. A set of initial solutions is generated using a heuristic. A local search is applied to the set of current solutions to generate several new solutions. A clustering procedure ensures that the size of the current solution set remains constant. The algorithm makes also use of an external archive for storing all the non dominated solutions found during the execution. After some initial experiments we found that under the considered stopping criterion (to be detailed later), less than 12 iterations were carried out. This together with the fact that the diversification method is not sufficiently clear from the

original text has resulted in our implementation not including this procedure. The initialization procedure of MOTS takes most of the allotted CPU time for large values of n. Considering the large neighborhood employed, this all results in extremely lengthy computations for larger n values.

Arroyo [14] proposed a genetic local search algorithm with the following features: preservation of population's diversity, elitism (a subset of the current Pareto front is directly copied to the next generation) and usage of a multi-objective local search. The concept of Pareto dominance is used to assign fitness (using the non dominated sorting procedure and the crowding measure both proposed for the NSGAII) to the solutions and in the local search procedure. We refer to this method as MOGALS_Arroyo.

A multi-objective simulated annealing (MOSA) is presented in [210]. The algorithm starts with an initialization procedure which generates two initial solutions using simple and fast heuristics. These sequences are enhanced by three improvement schemes and are later used, alternatively, as the solution of the simulated annealing method. MOSA tries to obtain non dominated solutions through the implementation of a simple probability function that attempts to generate solutions on the Pareto optimal front. The probability function is varied in such a way that the entire objective space is covered uniformly obtaining as many non dominated and well dispersed solutions as possible. We refer to this algorithm as MOSA_Varadharajan.

Pasupathy [148] proposed a genetic algorithm which we refer to as PGA_ALS. This algorithm uses an initialization procedure which generates four good initial solutions that are introduced in a random population. PGA_ALS handles a working population and an external one. The internal one evolves using a Pareto-ranking based procedure similar to that used in NSGAII. A crowding procedure is also proposed and used as a secondary selection criterion. The non dominated solutions are stored in the external archive and two different local searches are then applied to half of archive's solutions for improving the quality of the returned Pareto front.

Finally, we have also re-implemented PILS from [67]. This new algorithm is based on iterated local search which in turn relies on two main principles, *intensification* using a variable neighborhood local search and *diversification* using a perturbation procedure. The Pareto dominance relationship is used to store the non dominated solutions. This scheme is repeated through successive iterations to reach favorable regions of the search space.

Notice that among the 16 multi-objective PFSP specific papers reviewed in section 3.2, we are re-implementing a total of 10. We have chosen not to re-implement the GAs of [90] and [91] since they were shown to be inferior to

the multi-objective tabu search of [12] and some others. Loukil in [116] and [117] have presented some rather general methods applied to many scheduling problems. This generality and the lack of details have deterred us from trying a re-implementation. Arroyo [13] proposed just some heuristics and finally, the hybrid Particle Swarm Optimization (PSO) proposed by Rahimi-Vahed [157] in incredibly complex, making use of parallel programming techniques and therefore we have chosen not to implement it.

Other general Pareto algorithms

The multi-objective literature is marred with many interesting proposals, mainly in the form of evolutionary algorithms, that have not been applied to the PFSP before. Therefore, in this section we review some of these methods that have been re-implemented and adapted to the PFSP.
Deb [185] proposed the well known non dominated sorting genetic algorithm, referred to as NSGA. This method differs from a simple genetic algorithm only for the way the selection is performed. The Non Dominated Sorting procedure (NDS) iteratively divides the entire population into different Pareto fronts. The individuals are assigned a fitness value that depends on the Pareto front they belong to. Furthermore, this fitness value is modified by a factor that is calculated according to the number of individuals crowding a portion of the objective space. A sharing parameter σ_{share} is used in this case. All other features are similar to a standard genetic algorithm. Zitzler [224] presented another genetic algorithm referred to as SPEA. The most important characteristic of this method is that all non dominated solutions are stored in an external population. Fitness evaluation of individuals depend on the number of solutions from the external population they dominate. The algorithm also incorporates a clustering procedure to reduce the size of the non dominated set without destroying its characteristics. Finally, population's diversity is maintained by using the Pareto dominance relationship. Later, Zitzler [223] proposed an improved SPEAII version that incorporates a different fine-grained fitness strategy to avoid some drawbacks of the SPEA procedure. Other improvements include a density estimation technique that is an adaptation of the k-th nearest neighbor method, and a new complex archive truncation procedure.
Knowles [103] presented another algorithm called PAES. This method employs local search and a population archive. The algorithm is composed of three parts, the first one is the candidate solution generator which has an archive of only one solution and generates a new one making use of random mutation. The

second part is the candidate solution acceptance function which has the task of accepting or discarding the new solution. The last part is the non dominated archive which contains all the non dominated solutions found so far. According to the authors, this algorithm represents the simplest nontrivial approach to a multi-objective local search procedure. In the same paper, the authors present an enhancement of PAES referred to as $(\mu + \lambda)-$PAES. Here a population of μ candidate solutions is kept. By using a binary tournament, a single solution is selected and λ mutant solutions are created using random mutation. Hence, a $\mu + \lambda$ population is created and a dominance score is calculated for each individual. μ individuals are selected to update the candidate population while an external archive of non dominated solutions is maintained. Another genetic algorithm is proposed by Corne [45]. This method, called PESA uses an external population EP and an internal one IP to pursuit the goal of finding a well spread Pareto front. A selection and replacement procedure based on the degree of crowding is implemented. A simple genetic scheme is used for the evolution of IP while EP contains the non dominated solutions found. The size of the EP is upper bounded and a hyper-grid based operator eliminates the individuals in the more crowded zones. Later, in [44] a enhanced PESAII method is provided. This algorithm differs from the preceding one only in the selection technique in which the fitness value is assigned according to a hyperbox calculation in the objective space. In this technique, instead of assigning a selective fitness to an individual, it is assigned to the hyperboxes in the objective space which are occupied by at least one element. During the selection process, the hyperbox with the best fitness is selected and an individual is chosen at random among all inside the selected hyperbox.
In [49] an evolution of the NSGA was presented. This algorithm, called NSGAII, uses a new Fast Non Dominated Sorting procedure ($FNDS$). Unlike the NSGA, here a rank value is assigned to each individual of the population and there is no need for a parameter to achieve fitness sharing. Also, a crowding value is calculated with a fast procedure and assigned to each element of the population. The selection operator uses the rank and the crowding values to select the better individuals for the mating pool. An efficient procedure of elitism is implemented by comparing two successive generations and preserving the best individuals. This NSGAII method is extensively used in the multi objective literature for the most varied problem domains. Later, Deb [50] introduced yet another GA called CNSGAII. Basically, in this algorithm the *crowding* procedure is replaced by a *clustering* approach. The rationale is that once a generation is completed, the previous generation has a size of P_{size} (*parent* set) and the current one (*offspring* set) is also of the same size. Com-

bining both populations yields a $2P_{size}$ set but only half of them are needed for the next generation. To select these solutions the non dominated sorting procedure is applied first and the *clustering* procedure second.
Deb [50] studied another different genetic algorithm. This method, called $\varepsilon-$MOEA uses two co-evolving populations, the regular one called P and an archive A. At each step, two parent solutions are selected, the first from P and the second from A. An offspring is generated, and it is compared with each element of the population P. If the offspring dominates at least a single individual in P then it replaces this individual. The offspring is discarded if it is dominated by P. The offspring individual is also checked against the individuals in A. In the archive population the $\varepsilon-$dominance is used in the same way. For example, and using the previous notation, a solution x_1 strongly $\varepsilon-$dominates another solution x_2 ($x_1 \prec_\varepsilon x_2$) if $f_j(x_1) - \varepsilon \lhd f_j(x_2)$.
Zitzler [222] proposed another method called $B-$IBEA. The main idea in this method is defining the optimization goal in terms of a binary quality measure and directly using it in the selection process. B-IBEA performs binary tournaments for mating selection and implements environmental selection by iteratively removing the worst individual from the population and updating the fitness values of the remaining individuals. An $\varepsilon-$indicator is used. In the same work, an adaptive variation called $A-$IBEA is also presented. An adapted scaling procedure is proposed with the goal of making the algorithm's behavior independent from the tuning of the parameter k used in the basic $B-$IBEA version. Finally, Kollat [105] proposed also a NSGAII variation referred to as $\varepsilon-$NSGAII by adding $\varepsilon-$dominance archiving and adaptive population sizing. The ε parameter establishes the size of the grid in the objective space. Inside each cell of the grid no more than one solution is allowed. Furthermore, the algorithm works by alternating two phases. It starts using a very small population of 10 individuals and several runs of NSGAII are executed. During these runs all the non dominated solutions are copied to an external set. When there are no further improvements in the current Pareto front, the second phase starts. In this second phase the $\varepsilon-$dominance procedure is applied on the external archive.
The 23 re-implemented algorithms, either specific for the PFSP or general multi-objective proposals, are summarized in Table 4.1.

4.3 Multi-objective quality measures

As commented in previous chapters, comparing the solutions of two different Pareto approximations coming from two algorithms is not straightforward. Two approximation sets A and B can be even incomparable. Recent studies like those of [225], [145] or more recently, [104] are an example of the enormous effort being carried out in order to provide the necessary tools for a better evaluation and comparison of multi-objective algorithms. However, the multi-objective literature for the PFSP frequently uses quality measures that have been shown to be misleading. For example, in the two most recent papers reviewed (157 and 67) some metrics like generational distance or maximum deviation from the best Pareto front are used. These metrics, among other ones are shown to be non Pareto-compliant in the study of [104], meaning that they can give a better metric for a given Pareto approximation front B and worse for another front A even in a case where $A \prec B$. What is worse, in the comprehensive empirical evaluation of quality measures given in [104], it is shown that the most frequently used measures are non Pareto-compliant and are demonstrated to give wrong and misleading results more often than not. Therefore, special attention must be given to the choice of quality measures to ensure sound and generalizable results.

Knowles [104] propose three main approaches that are safe and sound. The first one relies on the Pareto dominance relations among sets of solutions. It is possible to rank a given algorithm over another based on the number of times the resulting Pareto approximation fronts dominate (strong, regular or weakly) each other. The second approach relies on quality indicators, mainly the hypervolume I_H and the Epsilon indicators that were already introduced in [224] and [225], respectively. Quality indicators usually transform a full Pareto approximation set into a real number. Lastly, the third approach is based on empirical attainment functions. Attainment functions give, in terms of the objective space, the relative frequency that each region is attained by the approximation set given by an algorithm. These three approaches range from straightforward and easy to compute in the case of dominance ranking to the not so easy and computationally intensive attainment functions.

In this paper, we choose the hypervolume (I_H) and the unary multiplicative Epsilon (I_ε^1) indicators. The choice is first motivated by the fact that dominance ranking is best observed when comparing one algorithm against another. By doing so, the number of times the solutions given by the first algorithm strongly, regularly or weakly dominate those given by the second gives a direct picture of the performance assessment among the two. The problem is that with 23 algo-

rithms compared in this chapter (see next Section) the possible two-algorithms pairs is 253 and therefore this type of analysis becomes unpractical. The same conclusion can be reached for the empirical attainment functions because these have to be compared in pairs. Furthermore, the computation of attainment functions is costly and the outcome has to be examined graphically one by one. As a result, such type of analysis is not useful in our case.
According to [104], I_H and I_ε^1 are Pareto-compliant and represent the state-of-the-art as far as quality indicators are concerned. Additionally, combining the analysis of these two indicators is a powerful approach since if the two indicators provide contradictory conclusions for two algorithms, it means that they are incomparable. In the following we give some additional details on how these two indicators are calculated.
The hypervolume indicator I_H, first introduced by [224] just measures the area (in the case of two objectives) covered by the approximated Pareto front given by one algorithm. A reference point is used for the two objectives in order to bound this area. A greater value of I_H indicates both a better convergence to as well as a good coverage of the optimal Pareto front. Calculating the hypervolume can be costly and we use the algorithm proposed in [48]. This algorithm already calculates a normalized and scaled value.
The binary epsilon indicator I_ε proposed initially by [225] is calculated as follows: Given two approximation sets A and B produced by two algorithms, the binary multiplicative epsilon indicator $I_\varepsilon(A, B)$ equals to

$$\max_{x_B} \min_{x_A} \max_{1 \leq j \leq M} \frac{f_j(x_A)}{f_j(x_B)}$$

where x_A and x_B are each of the solutions given by algorithms A and B, respectively. Notice that such a binary indicator would require to calculate all possible pairs of algorithms. However, in [104], a unary I_ε^1 version is proposed where the approximation set B is substituted by the best known Pareto front. This is an interesting indicator since it tells us how much worse (ε) an approximation set is w.r.t. the best known Pareto front in the best case. Therefore "ε" gives us a direct performance measure. Note however that in our case some objectives might take a value of zero (for example tardiness). Also, objectives must be normalized. Therefore, for the calculation of the I_ε^1 indicator, we first normalize and translate each objective, i.e., in the previous calculation, $f_j(x_A)$ and $f_j(x_B)$ are replaced by $\frac{f_j(x_A)-f_j^-}{f_j^+-f_j^-} + 1$ and $\frac{f_j(x_B)-f_j^-}{f_j^+-f_j^-} + 1$, respectively, where f_j^+ and f_j^- are the maximum and minimum known values for a given objective j, respectively. As a result, our normalized I_ε^1 indicator will take

values between 1 and 2. A value of one for a given algorithm means that its approximation set is not dominated by the best known one.

4.4 Benchmark and computational evaluation details

Each one of the 23 proposed algorithms is tested against a new benchmark set. There are no known comprehensive benchmarks in the literature for the multi-objective PFSP. The only reference we know of is the work of [19] where a small set of 14 instances is proposed. In order to carry out a comprehensive and sound analysis, a much larger set is needed. We augment the well known instances of [193]. This benchmark is organized in 12 groups with 10 instances each. The groups contain different combinations of the number of jobs n and the number of machines m. The $n \times m$ combinations are: $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$ and 500×20. The processing times (p_{ij}) in Taillard's instances are generated from a uniform distribution in the range $[1, 99]$. We take the first 110 instances and drop the last 10 instances in the 500×20 group since this size is deemed as too large for the experiments. As regards the due dates for the tardiness criterion we use the same approach of [82]. In this work, a tight due date d_j is assigned to each job $j \in N$ following the expression: $d_j = P_j \times (1 + random \cdot 3)$ where $P_j = \sum_{i=1}^{m} p_{ij}$ is the sum of the processing times over all machines for job j and $random$ is a random number uniformly distributed in $[0, 1]$. This method of generating due dates results in very tight to relatively tight due dates depending on the actual value of $random$ for each job, i.e., if $random$ is close to 0, then the due date of the job is going to be really tight as it would be more or less the sum of its processing times. As a result, the job will have to be sequenced very early to avoid any tardiness. These 110 augmented instances can be downloaded from `http://soa.iti.es`. Each algorithm has been carefully re-implemented following all the explanations given by the authors in the original papers. We have re-implemented all the algorithms in Delphi 2006. It should be noted that all methods share most structures and functions and the same level of coding has been used, i.e., all of them contain most common optimizations and speed-ups. Fast Non-Dominated Sorting ($FNDS$) is frequently used for most methods. Unless indicated differently by the authors in the original papers, the crossover and mutation operators used for the genetic methods are the two point order crossover and insertion mutation, respectively. Unless explicitly stated, all algorithms incorporate a duplicate-deletion procedure in the populations as well as in the non-dominated archives. Table 4.2 shows further implementation details of all

compared algorithms, including operators and parameter values.
The stopping criterion for most algorithms and is given by a time limit depending on the size of the instance. The algorithms are stopped after a CPU running time of $n \cdot m/2 \cdot t$ milliseconds, where t is an input parameter. Giving more time to larger instances is a natural way of separating the results from the lurking "total CPU time" variable. Otherwise, if worse results are obtained for large instances, it would not be possible to tell if it is due to the limited CPU time or due to the instance size. Every algorithm is run 10 different independent times (replicates) on each instance with three different stopping criteria: $t = 100$, 150 and 200 milliseconds. This means that for the largest instances of 200×20 a maximum of 400 seconds of real CPU time (not wall time) are allowed. For every instance, stoping time and replicate we use the same random seed as a common variance reduction technique.
We run every algorithm on a cluster of 12 identical computers with Intel Core 2 Duo E6600 processors running at 2.4 GHz with 1 Gbyte of RAM. For the tests, each algorithm and instance replicate is randomly assigned to a single computer and the results are collected at the end. According to Section 3.2, the three most common criteria for the PFSP are makespan, total completion time and total tardiness. All these criteria are of the minimization type. Therefore, all experiments are conducted for the three following criteria combinations: 1) makespan and total tardiness, 2) total completion time and total tardiness and finally, 3) makespan and total completion time.
A total of 75,900 data points are collected per criteria combination if we consider the 23 algorithms, 110 instances, 10 replicates per instance and three different stopping time criteria. In reality, each data point is an approximated Pareto front containing a set of vectors with the objective values. In total there are $75,900 \cdot 3 = 227,700$ Pareto fronts taking into account the three criteria combinations. The experiments have required approximately 5,100 CPU hours. From the $23 \cdot 10 \cdot 3 = 690$ available Pareto front approximations for each instance and criteria combination, a $FNDS$ is carried out and the best non-dominated Pareto front is stored. These "best" 110 Pareto fronts for each criteria combination are available for future use of the research community and are also downloadable from `http://soa.iti.es`. Additionally, a set of best Pareto fronts are available for the three different stopping time criteria. These last Pareto fronts are also used for obtaining the reference points for the hypervolume (I_H) indicator and are fixed to 1.2 times the worst known value for each objective. Also, these best Pareto fronts are also used as the reference set in the multiplicative epsilon indicator (I_ε^1).

4.5 Computational Evaluation

In this section we present a wide analysis of the huge set of experimental data collected. Statistical parametric and non-parametric tests have been largely employed to carry out a careful and sound analysis. In the following subsections three couple of objective functions are considered and for each of them the most performing algorithms are identified.

Makespan and total tardiness results

According to the review carried out in sections 3.2 , makespan and total tardiness are two common criteria. Furthermore, a low makespan increases machine utilization and throughput. However, the best possible makespan might sacrifice due dates and therefore both objectives are not correlated. We test all 23 re-implemented methods for these two criteria. Table 4.3 shows the average hypervolume (I_H) and epsilon indicator (I_ε^1) values for all the algorithms. Notice that the results are divided into the three different stopping criteria. Also, the methods are sorted in descending order of hypervolume value.
Although later we will conduct several statistical experiments, we proceed now to comment on the results. First and foremost, both quality indicators are contradictory in very few cases. We can observe that as hypervolume values decrease, the epsilon indicator increases. There are just some exceptions and these occur between consecutive algorithms with very similar hypervolume values, like for example SPEA and CNSGAII in positions 10 and 11 in the 200ms time columns. Another interesting result is that the ranking of the algorithms does not practically change as the allowed CPU time is increased and when it does it is motivated by small differences to start with. However, these are the observed average values across all instances and as we will mention later, there are more pronounced differences when focusing on specific instance sizes.
Since the objective values are normalized and the worst solutions are multiplied by 1.2, the maximum hypothetical hypervolume is $1.2^2 = 1.44$. As we can see, MOSA_Varadharajan is very close to this value in all three stopping criteria. Similarly, the minimum, possible epsilon indicator is one. Most interestingly, PESA and PESAII algorithms outperform PGA_ALS and MOTS by a noticeable margin. It has to be reminded that PESA and PESAII have just been re-implemented and adapted to the PFSP problem since both methods were proposed in the general multi-objective optimization literature, i.e., they were not built for the PFSP. GA_ALS and MOTS are PFSP-specific algorithms and in the case of MOTS, even for the same two objectives that have been

tested. It is also interesting how MOGA_Murata, is the 7th best performer in the comparison, although it is more than 10 years old and one of the first multi-objective algorithms proposed for the PFSP. This algorithm manages to outperform CMOGA, proposed also by the same authors and claimed to be better to MOGA_Murata. It has to be reminded that our re-implementations have been carried out according to the details given in all the reviewed papers. CMOGA might result to be better to MOGA_Murata under different CPU times or under specific optimizations. However, for the careful and comprehensive testing in this chapter, this is not the case.
In a much more unfavorable position are the remaining PFSP-specific methods (MOSA_Suresh, SA_Chakravarthy, PILS and ENGA). The bad performance of SA_Chakravarthy is expected, since it has to be recalled that this method uses the "a priori" approach by weighting the objectives and here we have run it for 100 different times with varying weights. Therefore, using this type of methods in such a way for obtaining a Pareto front is not advisable. ENGA is, as indicated by the original authors, better than NSGA but overall significantly worse than earlier methods like MOGA_Murata. Another striking result is the poor performance of the recent PILS method. Basically PILS is an iterative improvement procedure and is extremely costly in terms of CPU time. Therefore, in our experimental setting, it is outperformed by most other methods.
It should be specified that not all methods stop by the allotted CPU time as a stopping criterion. Some methods carry out some local searches after completing the iterations and some others just cannot be properly modified to stop at a given point in time. In any case, all CPU times stay within a given acceptable interval. For example, for 100 milliseconds stopping time and for the largest 200×20 instances tested, all methods should stop at 3.33 minutes. However, SA_Chakravarty stopped at 1.39 minutes. MOGALS_Arroyo at 3.36, MOTS at 3.52, PILS at 3.36, MOSA_Suresh at 12.8 and MOSA_Varadharajan at 2.23. For 200 milliseconds stopping time all methods should stop at 6.67 minutes for the largest instances. In this case, MOTS required 7.05 minutes, PILS 6.72, SA_Chakravarty 1.39, MOSA_Suresh 12.92 and MOSA_Varadharajan 2.16. Interestingly, MOSA_Varadharajan, the best performer of the test is actually a fast method and other not so well performers like MOSA_Suresh take a much longer time. All other methods can be controlled to stop at the specified time. These discrepancies in stopping times are only important for the largest instances as for the smallest ones more or less all CPU times are similar.
Table 4.3 contains just average values and many of them are very similar. Although each average is composed of a very large number of data points, it is still

necessary to carry out a comprehensive statistical experiment to assess if the observed differences in the average values are indeed statistically significant. A total of 12 different experiments are carried out. We do design of experiments (DOE) and parametric ANOVA analyses as well as non-parametric Friedman rank-based tests on both quality indicators and for the three different stopping criteria. The utility of showing both parametric as well as non-parametric tests is threefold. First, in the Operations Research and Computer Science literature it is common to disregard parametric testing due to the fact that this type of tests are based on assumptions that the data has to satisfy. Non-parametric testing is many times preferred since it is "distribution-free". However, non-parametric testing is nowhere as powerful as parametric testing. Second, in non-parametric testing a lot of information is lost since the data has to be ranked and the differences in the values (be these large or small) are transformed into a rank value. Third, ANOVA techniques allow for a much deeper and richer study of the data. Therefore we also compare both techniques in this chapter to support these claims. For more information the reader is refereed to [41] and [133]. We carry out six multi-factor ANOVAS where the type of instance is a controlled factor with 11 levels (instances from 20×5 to 200×20). The algorithm is another controlled factor with 23 levels. The response variable on each experiment is either the hypervolume or the epsilon indicator. Lastly, there is one set of experiments for each stopping time. Considering that each experiment contains 25,300 data points, the three main hypotheses of ANOVA: normality, homocedasticity and independence of the residuals are easily satisfied. To compare results, a second set of six experiments are performed. In this case, non-parametric Friedman rank-based tests are carried out. Since there are 23 algorithms and 10 different replicates, the results for each instance are ranked between 1 and 230. A rank of one represents the best result for hypervolume or epsilon indicator. We are performing four different statistical tests on each set of results (for example, for 100 CPU time we are testing ANOVA and Friedman on both quality indicators). Therefore, a correction on the confidence levels must be carried out since the same set of data is being used to make more than one inference. We take the most conservative approach, which is the Bonferroni adjustment, and we set the adjusted significance level α_s to $\frac{\alpha}{4} = \frac{0.05}{4} \simeq 0.01$. This means that all the tests are carried out at a 0.01 adjusted confidence level for a real confidence level of 0.05.
Figures 4.1 and 4.2 show the means plot for the factor algorithm in the ANOVA for the epsilon indicator response variable and the means plot for the ranks of the epsilon indicator, respectively. Both figures refer to the 100 CPU time stopping criterion. For the parametric tests we use Tukey's Honest Significant

Difference (HSD) intervals which counteract the bias in multiple pairwise comparisons. Similar Honest Significant Difference (HSD) intervals are used for the non-parametric tests. As can be seen, the non-parametric test is less powerful. Not only are the intervals much wider (recall that overlapping intervals indicate a non-statistically significant difference) but ranking neglects the differences in the response variables. PILS is shown to be better in rank than MOSA_Suresh and SA_Chakravarthy when we have already observed that it has worse average hypervolume and epsilon indicator values. The reason behind this behavior is that PILS is better for many small instances with a small difference in epsilon indicator than MOSA_Suresh and SA_Chakravarthy. However, it is much worse for some other larger instances. When one transforms this to ranks, PILS obtains a better rank more times and hence it appears to be better, when in reality it is marginally better more times but significantly worse many times as well. Concluding the discussion about parametric vs. non-parametric, if the parametric hypothesis are satisfied (even if they are not strictly satisfied, as thoroughly explained in 133) it is much better to use parametric statistical testing. Notice that we are using a very large dataset, with medium or smaller datasets, the power of non-parametric tests drops significantly.

The relative ordering, as well as most observed differences of the algorithms are statistically significant as we can see from Figure 4.1. As a matter of fact, the only non-statistically significant differences are those between $(\mu + \lambda)-$PAES and $\varepsilon-$NSGAII and between the algorithms CNSGAII, SPEA and NSGAII. MOGALS_Arroyo, PESA and PESAII are also equivalent.

Figure 4.3 shows the parametric results also for CPU time of 100 but for the hypervolume quality indicator. Notice that the Y-axis is now inverted since a larger hypervolume indicates better results. The relative ordering of the algorithms is almost identical to that of Figure 4.1, the only difference being MOGALS_Arroyo PESA and PESAII. This means that these algorithms show a very similar performance and on average are incomparable. A more in depth instance-by-instance analysis would be needed to tell them apart.

As we have shown, increasing CPU time does not change, on average, the relative ordering of the algorithms. Figure 4.4 shows the parametric results for the epsilon indicator and for 200 CPU time. We can observe the slight improvement on PILS but although there is an important relative decrease on the average epsilon indicator, it is not enough to improve the results to a significant extent. More or less all other algorithms maintain their relative positions with little differences.

We commented before that the average performance is not constant for all instance sizes. For example, PILS can be a very good algorithm for small

problems. Figure 4.5 shows the parametric results for the epsilon indicator and for 200ms CPU time but focused on the 50×5 instances. As we can see, this group of instances is "easy" in the sense that a large group of algorithms is able to give very low epsilon indicator values. Most notably, PILS statistically outperforms MOSA_Varadharajan albeit by a small margin. This performance however is not consistent. For instances of the size 50×10, PILS is the third best performer and for instances if size 100×5 PILS results to be the worst algorithm in the comparison. A worthwhile venue of research would be to investigate the strengths of PILS and to speed it up so that the performance is maintained for a larger group of instances.

Total completion time and total tardiness results

Total completion time criterion is related to the amount of work-in-progress or WIP. A low total completion time minimizes WIP and cycle time as well. As it was the case with the makespan criterion, total completion time is not correlated with total tardiness. Therefore, in this section we report the results for these two objectives.
Table 4.4 shows the corresponding average hypervolume and epsilon indicator values.
One should expect that different criteria combinations should result in different performance for the algorithms tested. However, comparing the results of Tables 4.3 and 4.4 gives a different picture. MOSA_Varadhrajan and MOGALS_Arroyo produce the best results with independence of the allowed CPU time. MOGA_Murata as well as many others keep their relative position and most other PFSP-specific methods are still on the lower positions despite the new criteria combination. The only noteworthy exception is the $(\mu+\lambda)-$PAES method which is ranking 7th where for the other criteria combinations it was ranking around 12th. As it was the case with the previous makespan and total tardiness criteria combination, both quality indicators and both parametric as well as non-parametric statistics give consistent results. For example, Figure 4.6 shows the parametric results for the epsilon indicator and for 200ms CPU time across all instances.
As can be seen, most of the observed differences from Table 4.4 are indeed statistically significant.

Makespan and total completion time results

Finally, we study the combination of makespan and total completion time. Contrary to the two previous criteria combinations, these two objectives are correlated. It is straightforward to see that a low makespan value will also result in a reduced total completion time. However, there are some scheduling scenarios where this is not true. In our case, the best approximated Pareto fronts obtained throughout the tests contain several points and in the case of large instances, Pareto fronts are composed of hundreds of points. Therefore, it seems that these two objectives are not as correlated as they seem. Table 4.4 shows the observed average values. Once again, the results are remarkably similar with respect to the other criteria combinations. MOSA_Varadhrajan and MOGALS_Arroyo are the best performers for the three criteria combinations and the non PFSP-specific methods PESA and PESAII are very good performers as well. As in the other cases, most observed differences in the average values are statistically significant as shown in Figure 4.7.

4.6 Comments and Conclusions

In this chapter we have conducted a comprehensive computational evaluation of 23 different metaheuristics proposed for the Pareto or "a posteriori" multi-objective approach. Recent and state-of-the-art quality measures have been employed in an extensive experiment where makespan, total completion time and total tardiness criteria have been studied in three different two-criteria combinations. The comparative evaluation not only includes flowshop-specific algorithms but also adaptations of other general methods proposed in the multi-objective optimization literature. A new set of benchmark instances, based on the well known benchmark of [193] has been proposed and is currently available online along with the best known Pareto fronts for the tested objectives.
A comprehensive statistical analysis of the results has been conducted with both parametric as well as non-parametric techniques. We have shown the preferred qualities of the parametric tests over the non-parametric counterparts, contrary to what is mainstream in the literature. The array of statistical analyses soundly support the observed performances of the evaluated algorithms.
As a result, we have identified the best algorithms from the literature, which, along with the survey, constitute an important study and reference work for further research. Overall, the multi-objective simulated algorithm of [210], MOSA_Varadharajan can be regarded as the best performer under our experimental settings. Another consistent performer is the genetic local search

method MOGALS_Arroyo of [14]. Our adapted versions of PESA and PESAII from [45] and [44], respectively, have shown a very good performance over many other flowshop-specific algorithms. The recent PILS method from [67] has shown a promising performance for small instances. In our study we have also shown that different stopping criteria as well as different criteria combinations result in little changes, i.e., the algorithms that give best results do so in a wide array of circumstances.

Tables and figures

Acronym	Year	Author/s	Type
NSGA	1994	Srinivas and Deb	Genetic algorithm. General
MOGA_Murata	1996	Murata et al.	Genetic algorithm. Specific
SPEA	1999	Zitzler and Thiele	Genetic algorithm. General
SA_Chakravarty	1999	Chakravarthy and Rajendran	Simulated annealing. Specific
PAES	2000	Knowles and Corne	Population local search. General
$(\mu + \lambda)-$PAES	2000	Knowles and Corne	Population local search. General
PESA	2000	Corne et al.	Genetic algorithm. General
SPEAII	2001	Zitzler et al.	Genetic algorithm. General
PESAII	2001	Corne et al.	Genetic algorithm. General
ENGA	2001	Bagchi	Genetic algorithm. Specific
CMOGA	2001	Murata et al.	Genetic algorithm. Specific
NSGAII	2002	Deb	Genetic algorithm. General
CNSGAII	2003	Deb et al.	Genetic algorithm. General
$\varepsilon-$MOEA	2003	Deb et al.	Genetic algorithm. General
$B-$IBEA	2004	Zitzler and Künzli	Genetic algorithm. General
$A-$IBEA	2004	Zitzler and Künzli	Genetic algorithm. General
MOSA_Suresh	2004	Suresh and Mohanasundaram	Simulated annealing. Specific
MOTS	2004	Armentano and Arroyo	Tabu search. Specific
$\varepsilon-$NSGAII	2005	Kollat and Reed	Genetic algorithm. General
MOGALS_Arroyo	2005	Arroyo and Armentano	Genetic algorithm. Specific
MOSA_Varadharajan	2005	Varadharajan and Rajendran	Simulated annealing. Specific
PGA_ALS	2006	Pasupathy et al.	Genetic algorithm. Specific
PILS	2007	Geiger	Iterated local search. Specific

Table 4.1: Re-implemented methods for the multi-objective flowshop.

Parameters	NSGA	MOGA_Mur.	SPEA	SA_Chakr.	PAES	$(\mu+\lambda)-$PAES	PESA	SPEAII
IniPop	R	R	R	PR	R	R	R	R
P_s	100	100	100	3	10	10	100	100
Selection	T	R	T	-	T	T	R	R
Crossover	OP	TP	TP	-	-	-	UX	OP
P_c	1	0,01	0,01	-	-	-	0	0,04
Mutation	-	S	S	-	S	S	S	S
P_m	-	0	0	-	0,04	$1/n$	$1/n$	6
Archive_s	-	-	20	-	50	50	100	100
# Elite	-	5	-	-	-	-	-	-
Others	σ_{share}=0.1 α=1	-	-	T_i=475 T_e=5 P_w=0.15	G_s=5	G_s=5	$G_s = 32 \times 32$	-

Parameters	PESAII	ENGA	CMOGA	NSGAII	CNSGAII	$\varepsilon-$MOEA	$B-$IBEA	$A-$IBEA
IniPop	R	R	R	R	R	R	R	R
P_{size}	100	200	100	10	10	100	100	100
Selection	HR	R	R	T	T	R	T	T
Crossover	UX	OP	TP	OP	TP	TP	OP	OP
P_{cross}	0	0,01	0,01	0,01	0,04	0,04	0,01	0,01
Mutation	S	-	S	S	S	S	S	S
P_{mut}	$1/n$	-	0	$1/n$	$1/n$	$1/n$	0	0
Archive size	100	-	-	-	-	-	-	-
# Elite	-	-	3	-	-	-	-	-
Others	$G_s = 32 \times 32$	σ_{share}=0.2 α=1	MCD=20		- M_i=100 M_n=5 P_w=0.8	ϵ=0.001	K=0.05 I_ε^+	K=0.05 I_ε^+

Parameters	MOSA_Sur.	MOTS	$\varepsilon-$NSGAII	MOGALS_Arr.	MOSA_Varadh.	PGA_ALS	PILS
IniPop	R	H	R	R	H	H	R
P_{size}	1	10	Dynamic	100	4	100	Dynamic
Selection	-	-	T	R	-	R	-
Crossover	-	-	TP	TP	-	OP	-
P_{cross}	-	-	0,04	0,01	-	0,04	-
Mutation	-	-	S	S	-	S	-
P_{mut}	-	-	$1/n$	0	-	0	-
Archive size	10	-	-	-	500	-	-
# Elite	-	-	-	20	-	-	-
Others	T_i=285	-	ϵ=0.01	NP=10	T_i=575	-	NN=3
	T_e=5				T_e=20		
	C_r=0.9				C_r=0.9		
					M_i=30		

Table 4.2: Details, operators and parameter values of the algorithms.

IniPop= population initialization: R=Random, PR=Priority rules, H=Heuristic. P_s=Population size. Selection: T=Tournament, R=Roulette, HR=Hyperbox roulette. Crossover: OP=One point order crossover, TP=Two point order crossover, UX=Uniform order crossover. P_c=Crossover probability. P_m=Mutation probability. Archive_s=Archive size. # Elite=Number of elite solutions. T_i=Initial temperature (SA), T_e=Final temperature (SA), P_w=Probability of accepting worse solutions. G_s=Grid size. MCD=Maximum cell distance. I_ε^+=Aditive epsilon indicator. C_r=Cooling rate. M_i=Maximum number of iterations at the same temperature. M_n=Maximum number of non-improving iterations. NP=Number of paths to be explored. NN=Number of neighbourhoods.

#	Time 100 Method	I_H	I_ε^1	150 Method	I_H	I_ε^1	200 Method	I_H	I_ε^1
1	MOSA_Varadharajan	1.366	1.049	MOSA_Varadharajan	1.360	1.053	MOSA_Varadharajan	1.357	1.056
2	MOGALS_Arroyo	1.290	1.089	MOGALS_Arroyo	1.299	1.082	MOGALS_Arroyo	1.301	1.080
3	PESA	1.285	1.086	PESA	1.287	1.084	PESA	1.291	1.081
4	PESAII	1.280	1.089	PESAII	1.284	1.086	PESAII	1.288	1.084
5	PGA_ALS	1.273	1.116	PGA_ALS	1.267	1.116	PGA_ALS	1.255	1.118
6	MOTS	1.224	1.135	MOTS	1.232	1.131	MOTS	1.235	1.129
7	MOGA_Murata	1.178	1.148	MOGA_Murata	1.187	1.143	MOGA_Murata	1.195	1.138
8	CMOGA	1.150	1.162	CMOGA	1.169	1.150	CMOGA	1.181	1.143
9	NSGAII	1.142	1.168	NSGAII	1.156	1.160	NSGAII	1.165	1.155
10	SPEA	1.141	1.169	SPEA	1.154	1.161	SPEA	1.164	1.155
11	CNSGAII	1.137	1.169	CNSGAII	1.153	1.160	CNSGAII	1.162	1.154
12	$\varepsilon-$NSGAII	1.091	1.195	$\varepsilon-$NSGAII	1.106	1.187	$\varepsilon-$NSGAII	1.115	1.182
13	$(\mu+\lambda)-$PAES	1.086	1.197	$(\mu+\lambda)-$PAES	1.101	1.188	$(\mu+\lambda)-$PAES	1.110	1.183
14	PAES	1.049	1.219	$\varepsilon-$MOEA	1.051	1.222	$\varepsilon-$MOEA	1.061	1.216
15	$\varepsilon-$MOEA	1.045	1.225	PAES	1.035	1.227	PAES	1.028	1.230
16	MOSA_Suresh	0.976	1.290	MOSA_Suresh	0.960	1.301	MOSA_Suresh	0.955	1.303
17	SA_Chakravarty	0.894	1.395	SA_Chakravarty	0.882	1.402	SA_Chakravarty	0.870	1.410
18	PILS	0.803	1.409	PILS	0.843	1.379	PILS	0.866	1.363
19	ENGA	0.588	1.522	$A-$IBEA	0.592	1.518	$A-$IBEA	0.599	1.514
20	$A-$IBEA	0.578	1.528	ENGA	0.570	1.534	ENGA	0.559	1.542
21	SPEAII	0.537	1.544	SPEAII	0.524	1.550	SPEAII	0.522	1.549
22	NSGA	0.520	1.571	NSGA	0.502	1.584	NSGA	0.490	1.593
23	$B-$IBEA	0.411	1.640	$B-$IBEA	0.411	1.641	$B-$IBEA	0.417	1.637

Table 4.3: Results for the makespan and total tardiness criteria. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H.

#	Time	100			150			200	
	Method	I_H	I_ε^1	Method	I_H	I_ε^1	Method	I_H	I_ε^1
1	MOSA_Varadharajan	1.381	1.034	MOSA_Varadharajan	1.377	1.036	MOSA_Varadharajan	1.375	1.036
2	MOGALS_Arroyo	1.330	1.057	MOGALS_Arroyo	1.326	1.060	MOGALS_Arroyo	1.324	1.061
3	PGA_ALS	1.311	1.076	PGA_ALS	1.313	1.075	PGA_ALS	1.315	1.074
4	MOTS	1.296	1.073	MOTS	1.302	1.071	MOTS	1.309	1.067
5	PESA	1.263	1.088	PESA	1.262	1.088	PESA	1.263	1.088
6	PESAII	1.263	1.088	PESAII	1.261	1.088	PESAII	1.262	1.087
7	$(\mu+\lambda)-$PAES	1.190	1.126	$(\mu+\lambda)-$PAES	1.196	1.123	$(\mu+\lambda)-$PAES	1.200	1.121
8	MOGA_Murata	1.183	1.128	MOGA_Murata	1.191	1.124	MOGA_Murata	1.198	1.120
9	CMOGA	1.163	1.138	CMOGA	1.178	1.130	CMOGA	1.188	1.125
10	NSGAII	1.158	1.143	NSGAII	1.170	1.137	NSGAII	1.176	1.134
11	CNSGAII	1.148	1.146	CNSGAII	1.162	1.139	CNSGAII	1.171	1.135
12	SPEA	1.137	1.151	SPEA	1.149	1.145	SPEA	1.157	1.141
13	$\varepsilon-$NSGAII	1.111	1.165	$\varepsilon-$NSGAII	1.131	1.155	$\varepsilon-$NSGAII	1.144	1.148
14	$\varepsilon-$MOEA	1.100	1.170	$\varepsilon-$MOEA	1.107	1.167	$\varepsilon-$MOEA	1.111	1.164
15	PAES	1.081	1.190	PAES	1.071	1.195	PAES	1.066	1.197
16	MOSA_Suresh	1.065	1.207	MOSA_Suresh	1.053	1.214	MOSA_Suresh	1.048	1.216
17	PILS	0.824	1.388	PILS	0.865	1.357	PILS	0.900	1.334
18	SA_Chakravarty	0.612	1.480	SA_Chakravarty	0.592	1.493	SA_Chakravarty	0.584	1.498
19	SPEAII	0.453	1.572	SPEAII	0.446	1.577	SPEAII	0.445	1.576
20	ENGA	0.426	1.609	ENGA	0.406	1.628	ENGA	0.397	1.635
21	NSGA	0.368	1.658	NSGA	0.348	1.679	$A-$IBEA	0.339	1.670
22	$A-$IBEA	0.333	1.676	$A-$IBEA	0.337	1.674	NSGA	0.338	1.686
23	$B-$IBEA	0.332	1.676	$B-$IBEA	0.337	1.673	$B-$IBEA	0.338	1.671

Table 4.4: Results for the total completion time and total tardiness criteria. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H.

#	Time 100 Method	I_H	I_ε^1	150 Method	I_H	I_ε^1	200 Method	I_H	I_ε^1
1	MOSA_Varadharajan	1.362	1.054	MOSA_Varadharajan	1.356	1.058	MOSA_Varadharajan	1.350	1.061
2	MOGALS_Arroyo	1.337	1.066	MOGALS_Arroyo	1.337	1.064	MOGALS_Arroyo	1.336	1.064
3	MOTS	1.309	1.087	MOTS	1.312	1.085	MOTS	1.312	1.083
4	PGA_ALS	1.271	1.120	PESA	1.273	1.091	PESA	1.275	1.091
5	PESA	1.269	1.092	PGA_ALS	1.272	1.121	PGA_ALS	1.272	1.122
6	PESAII	1.262	1.095	PESAII	1.267	1.093	PESAII	1.268	1.093
7	MOGA_Murata	1.160	1.154	MOGA_Murata	1.171	1.148	MOGA_Murata	1.176	1.146
8	CMOGA	1.134	1.167	CMOGA	1.155	1.155	CMOGA	1.167	1.149
9	CNSGAII	1.111	1.180	CNSGAII	1.126	1.171	CNSGAII	1.133	1.168
10	NSGAII	1.106	1.184	NSGAII	1.121	1.177	NSGAII	1.128	1.173
11	SPEA	1.099	1.186	SPEA	1.116	1.176	SPEA	1.122	1.173
12	$(\mu+\lambda)-$PAES	1.059	1.206	$(\mu+\lambda)-$PAES	1.071	1.199	$(\mu+\lambda)-$PAES	1.077	1.196
13	$\varepsilon-$NSGAII	1.051	1.213	$\varepsilon-$NSGAII	1.068	1.204	$\varepsilon-$NSGAII	1.077	1.199
14	$\varepsilon-$MOEA	1.027	1.233	$\varepsilon-$MOEA	1.036	1.228	$\varepsilon-$MOEA	1.040	1.226
15	PAES	1.005	1.238	PAES	0.992	1.244	PAES	0.980	1.252
16	MOSA_Suresh	0.950	1.296	MOSA_Suresh	0.936	1.303	MOSA_Suresh	0.922	1.314
17	SA_Chakravarty	0.813	1.410	PILS	0.836	1.383	PILS	0.861	1.367
18	PILS	0.794	1.413	SA_Chakravarty	0.798	1.422	SA_Chakravarty	0.781	1.431
19	ENGA	0.512	1.575	ENGA	0.492	1.590	ENGA	0.477	1.604
20	SPEAII	0.475	1.586	SPEAII	0.467	1.590	SPEAII	0.458	1.600
21	NSGA	0.446	1.625	$A-$IBEA	0.426	1.626	$B-$IBEA	0.425	1.631
22	$A-$IBEA	0.416	1.634	NSGA	0.426	1.642	$A-$IBEA	0.424	1.631
23	$B-$IBEA	0.416	1.634	$B-$IBEA	0.423	1.629	NSGA	0.409	1.658

Table 4.5: Results for the makespan and total completion time. Average quality indicator values for the 23 algorithms tested under the three different termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H.

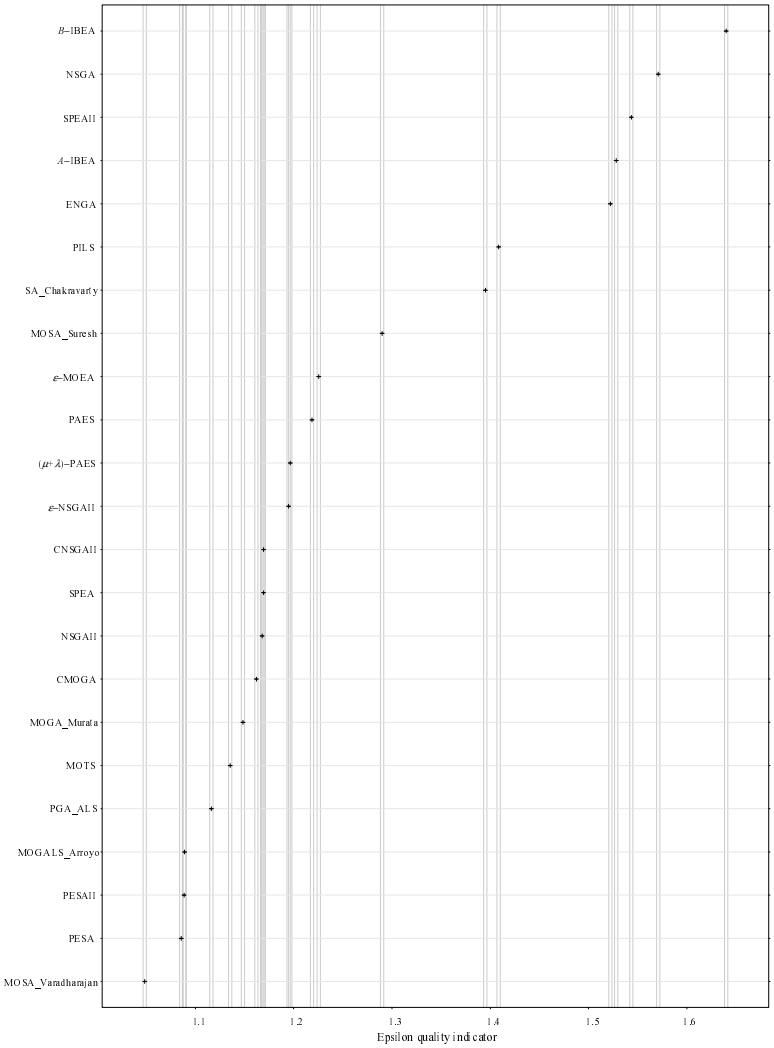

Figure 4.1: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria.

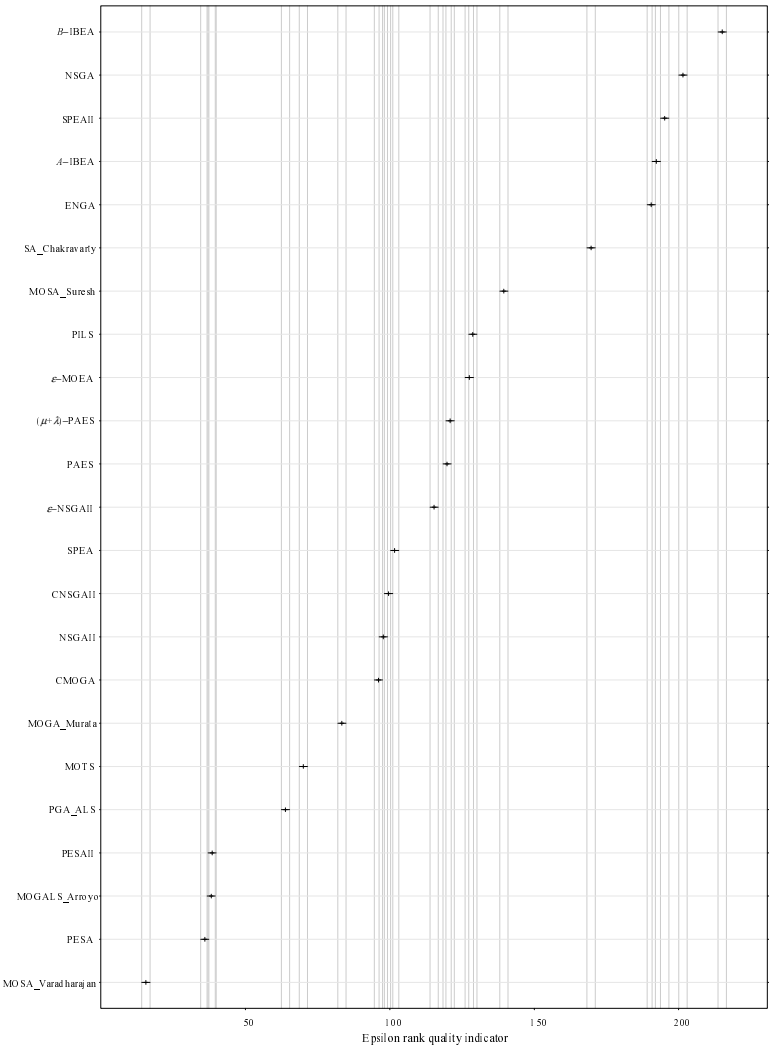

Figure 4.2: Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria.

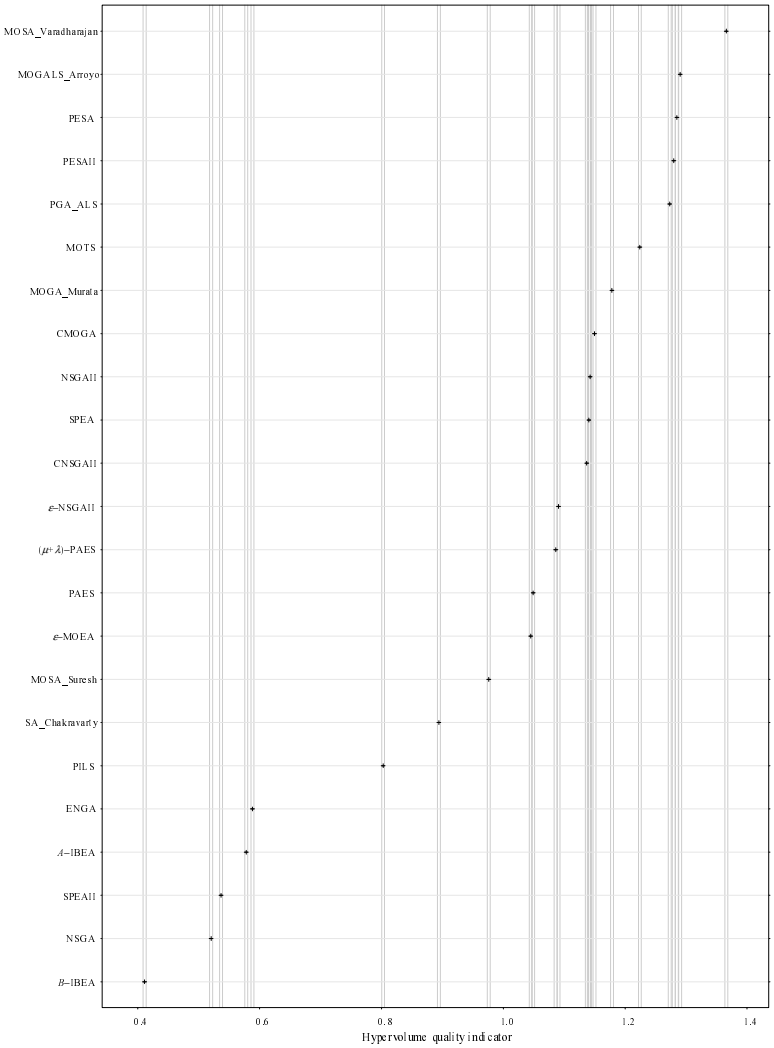

Figure 4.3: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 100 CPU time stopping criterion. Makespan and total tardiness criteria.

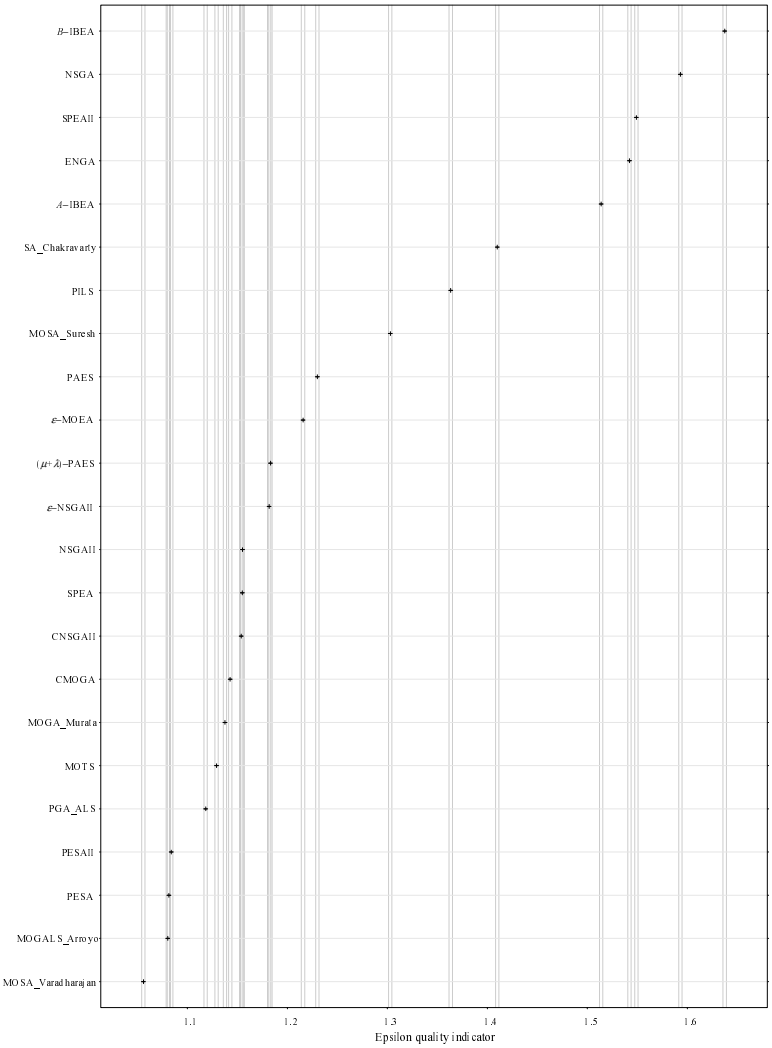

Figure 4.4: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total tardiness criteria.

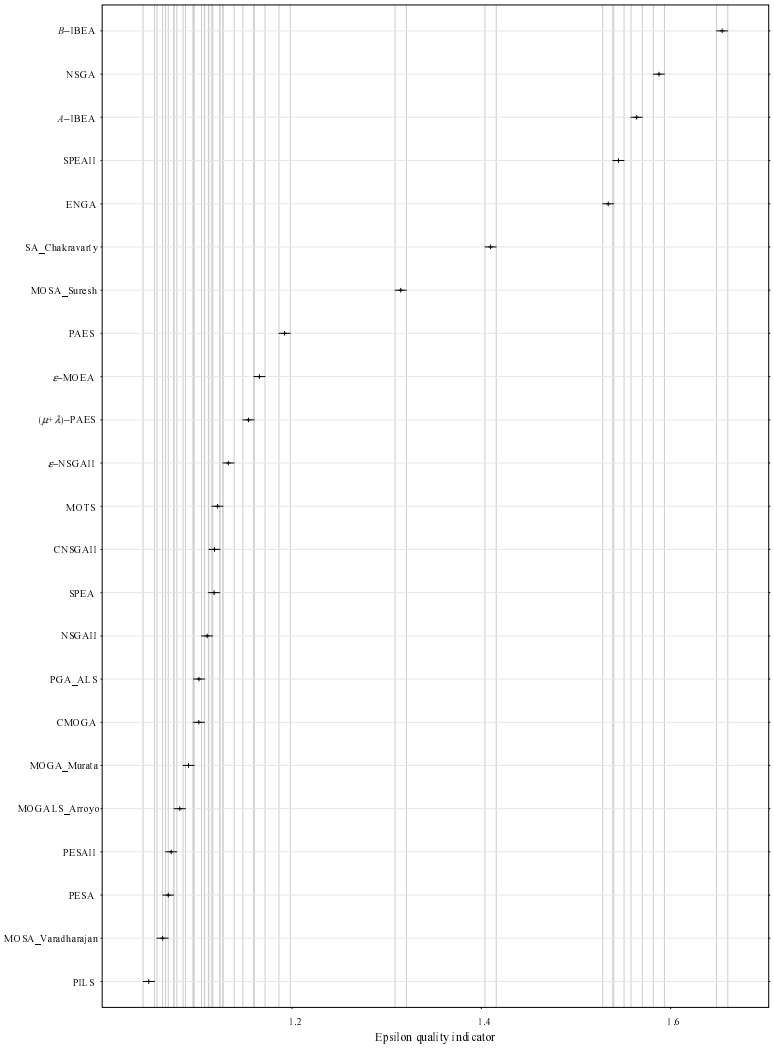

Figure 4.5: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment for the instance group 50×5. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total tardiness criteria.

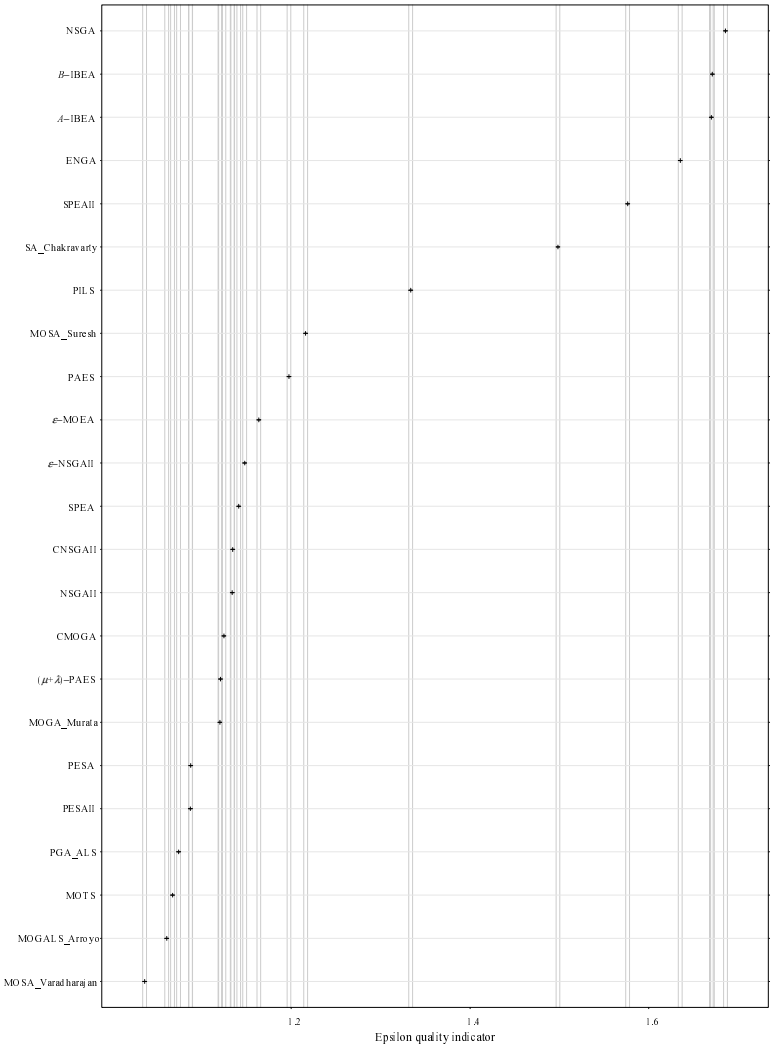

Figure 4.6: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Total completion time and total tardiness criteria.

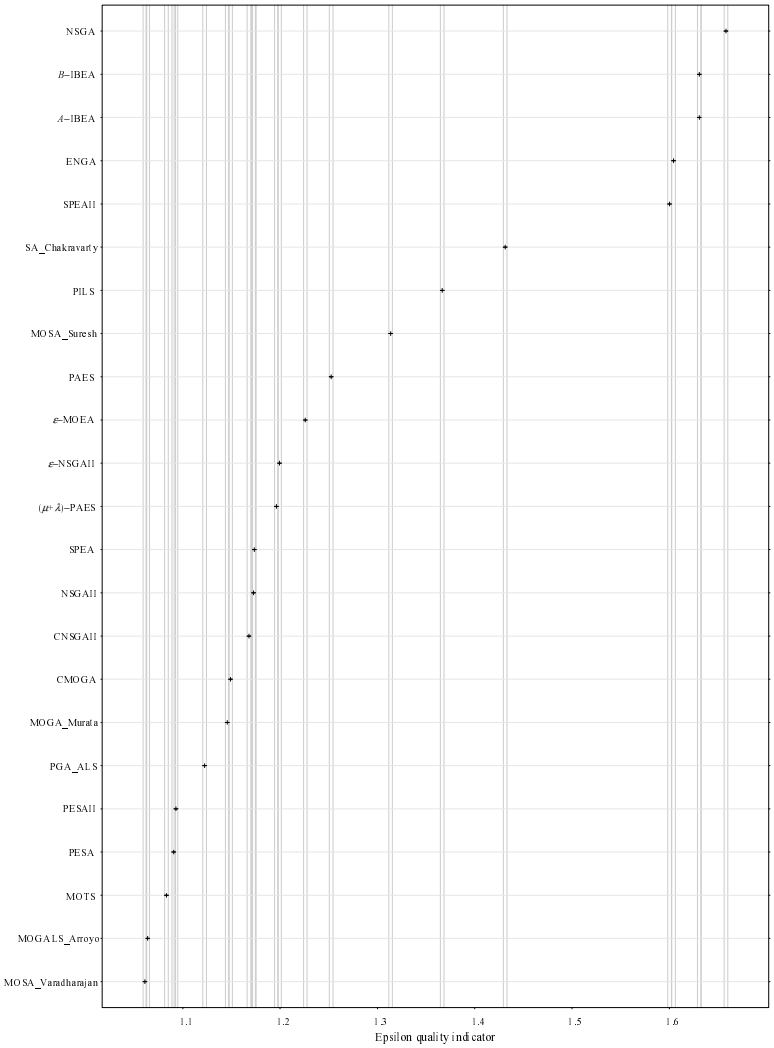

Figure 4.7: Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total completion time criteria.

Chapter 5

Pareto Iterated Greedy Algorithm for PFSP with setups

In this chapter we extend the model presented in chapter 4 considering the presence of sequence dependent setup times.
In the literature, many papers have faced the flowshop scheduling problem with setups, but according to our knowledge, nothing has been published dealing with both the optimization of more than one objective and sequence dependent setup times, although it is definitely a relevant topic. Here we tackle such a problem by means of an innovative algorithm (IPG). We compare this new approach with the highest performig classical approaches, proposed for multi-objective flowshop problem or general purpose. Finally, statistical techniques are employed to prove that IPG widely outperforms all other approaches.

5.1 Introduction

The aim of this chapter is to introduce the problem of multi-objective permutation flowshop with sequence dependent setup times, to present a new, highly effective, algorithm and to demonstrate its superiority when compared with the best published approaches.
The well known flowshop scheduling problem (FSP) is considered. In the multi-objective version of this problem the target is to find a set of processing sequences of jobs so that a given set of different criteria is optimized. In this

chapter we consider two well known independent objectives, the Maximum Completion Time also called Makespan (C_{max}) and the Total Weighted Tardiness (TWT). In addition, in real life production environments it is often required to consider the presence of setup times. We can roughly classify them in two main categories. In the first one there are those which are *Sequence Independent* (SIST) i.e. the changeover time for a machine only depends on the current job in execution. In the second category there are the *Sequence Dependent* setup times (SDST). In this case setup time for a machine depends either on the actual job in process and on the preceding one. Since the second group is more general, more often present in real cases and less studied, we limited our investigation to this field. The presence of setup times affects with delay the completion time $c_{j,k}$ of each job j on each machine k and hence the objectives considered.
Although in some cases it is possible to consider the setup costs, in terms of money and time, included in the processing costs, in the majority of industrial contexts it is not possible to ignore them. In the last decade the relevance of both multi-objective and setup topics has been increasing. This is why we decided to face with this more general and complex problem, using an algorithm of new conception. We proved its effectiveness against the set of the best multi-objective algorithms, general purpose or proposed expressly for the PFSP, modified for handling setup times. Furthermore, we carefully selected and employed basic as well as advanced effectiveness measures. Careful and comprehensive statistical testing is carried out to ensure the confidence of the conclusions.
The remainder of this chapter is organized as follows: section 5.2 presents a mathematical model and an accurate description of the considered problem. In section 5.3 we give a careful description of the proposed algorithm. In section 5.4 results of a wide campaign of experiments are shown and analyzed. Finally in section 5.5 some conclusions and further research topics are given.

5.2 Problem Description

Let $\mathcal{J} = \{J_1, J_2 \dots J_n\}$ the set of jobs to be processed in a production environment that consists in the set $\mathcal{M} = \{M_1, M_2 \dots M_m\}$ machines and let $p_{i,j}$, the processing time of job i on machine j for each $i \in N = \{1, 2 \dots n\}$ and $j \in M = \{1, 2 \dots m\}$, be known in advance. Each job J_i has assigned a due date d_i, that represents the delivery date agreed with the costumer, it deals with the completion time of the job on the last machine. If a job is terminated

after this time, then it is considered late. For each couple of jobs in $\mathcal{J}$ and for each machine in $\mathcal{M}$ a setup time value $s_{i,j,k}$ is given. Without loss of generality we rearranged indexes of machines in such a way that the first machine in the line has index 1, the second has index 2 and so on.
In the most general version of the flowshop scheduling problem (FSP) each machine processes a possibly different sequence of jobs, but in this paper we consider only the case a single sequence is processed by all the machines in the production line. PFSP has been demonstrated to belong to the $\mathcal{NP}$-hard class for many optimization criteria as shown in chapter 3. And the presence of setup times makes PFSP also more difficult to solve. Setups in fact have to be added to processing times in order to calculate the completion time of each job on each machine, furthermore in our model they are *sequence dependent* (SDST) i.e the time needed for a setup on a machine depends on both the actual job in process and the preceding one. Hence a permutation that is the optimum for the classical PFSP might be far from the optimum for the same criterion but considering setup times. In this chapter we face a bi-objective version of PFSP with SDST where optimization criteria are: the minimization of the maximum completion time of every job on the last machine (Makespan or C_{max}) (eq. 5.2) and the minimization of the Total Weighted Tardiness (eq. 5.3). Hereinafter we denote our problem as PSP-SDST-(C_{max}, TWT). In what follows a short description of this problem is presented.

$$\text{PFSP-SDST-}(C_{max}, WT) \; : \min_{Pareto} (C_{max}, WT) \tag{5.1}$$

with

$$C_{max} = \max_{i=1\ldots n} c_{i,m} \tag{5.2}$$

$$WT = \sum_{i=1}^{n} w_i \times T_i = \sum_{i=1}^{n} [w_i \times \max(0, c_{i,m} - d_i)] \tag{5.3}$$

subject to

$$c_{i,j} = \max(c_{i,j-1}, c_{i-1,j} + s_{i-1,i,j}) + p_{i,j}, \; (i,j) \in N \times M \tag{5.4}$$

$$c_{0,j} = 0, c_{i,0} = 0, s_{0,i} = 0$$

Expression 5.1 indicates the problem type, i.e the minimization of criteria presented in 5.2 and 5.3 according to Pareto relationship. Finally equation 5.4 shows how to calculate the completion time of job i on machine j, notice that such value depends either on the completion time of the same job on the previous machine and on the one of the previous job on the same machine.

5.3 Iterated Pareto Greedy

A new algorithm named *Iterated Pareto Greedy* (IPG) is presented in this section. As can be deduced form the name, a greedy strategy is iteratively applied over an archive of nondominated solutions. Such procedure, presented for the first time in this chapter, is an evolution of the NEH heuristic of Nawaz et al. [139] making use of the Pareto relationship to generate a set of solutions which do not dominate each other.
The Iterated Greedy (IG) algorithm is a rather new metaheuristic approach which has recently demonstrated to be the state-of-the-art for single objective optimization of PFSP with [171] and without setups [170]. The IPG is a particular IG where the greedy procedure returns a set of nondominated solutions. The main idea is very simple, a greedy procedure is iteratively invoked to generate a set of complete nondominated solutions starting from a partial solution. Roughly it is possible to subdivide it into four phases. in the first phase (*Initialization*) an initial set of possibly good solutions is generated using an heuristic approach. Each heuristic employed is able to attain good values for only a single criterion. The remaining three phases are iteratively repeated and constitute the bulk of the algorithm. The second phase called *Selection* provides one or more solutions of the current archive to the next phase. The *Pareto improvement* phase is applied on a selected solution. During this phase a current solution is disrupted (*Destruction*) eliminating some jobs from the sequence and a greedy procedure (*Construction*) is applied, it reinserts the eliminated jobs into partial sequences returning hopefully a set of improved nondominated solutions. Finally the *Local search* phase is applied on a solution of the solution archive. Figure 5.1 is a flow chart of IPG algorithm. Following subsections describe in details each phase

Algorithm Initialization

In order to obtain a good initial solution set (ISS) for IPG we used an initialization procedure proposed in [210] which attained good results. It makes

use of NEH heuristic of Nawaz et al. [139] and of a different one presented by Rajendran in [160] both designed for the optimization of a single criterion. Two distinct solutions for each objective to optimize are generated. Each solution is then subjected to three different improvement scheme called *Job-index insertion*, *Overall-seed sequence insertion* and *Job-index swapping*. For more details see [210].

Selection phase

In the first iteration all the sequences in the initial solution set (ISS) are selected for the improvement phase. In the remaining iterations, only a single solution is selected. Such different behavior avoids that, by selecting always a single sequence, a great improvement during the first iteration could generate a set of solutions which dominates all the sequences in ISS. This could compromise the search toward other promising regions of the objective space.
A modified version of the *Crowding Distance Assignment* procedure, originally presented in [49], has been developed in order to carry on the selection process. Such procedure assigns to each element of the solution set a value (*Crowding distance*) depending on the distance between it and the nearest solutions of the same dominance level. The main difference consists in the fact that the modified procedure considers the number of times each solution has been selected in the preceding iterations and uses this information calculating a modified crowding distance (MCD). The element with the highest value of MCD is selected as current solution and used as a starting point in the Pareto improving phase. The goal purpose of this procedure consists in selecting a candidate solution which at the same time belongs to a less crowded region of the Pareto front and possibly selected a lower number of times. The use of this measure should improve the Pareto front in terms of quality and spread of its solutions. The pseudocode of this procedure is presented in figure 5.2.

Pareto Improving phase

This is a complex phase that, for a better understanding, can be subdivided in two sub-phases that are respectively called *Destruction* and *Construction*.
During the *Destruction*, k positions of the sequence are selected and the respective jobs are removed. Jobs may be selected in a totally random way i.e. each position is randomly generated without repetition. A different case is when a block of k consecutive positions is selected starting from a random generated point of the sequence. The set of the removed items (RI) is then rearranged in

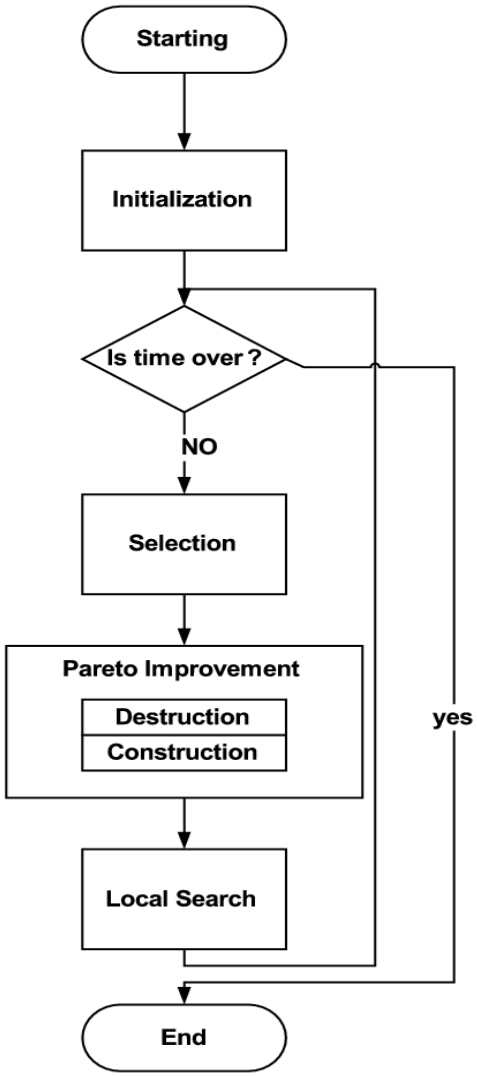

Figure 5.1: The figure represents a schematic flow chart of IPG

Modified-crowding-distance-assignment(ParetoSet)

$DimSet := |ParetoSet|;$

for all $i := 1 \ldots DimSet$

$ParetoSet[i]_{dist.} = 0;$

for all objective m

$ParetoSet := sort(ParetoSet, m);$

$ParetoSet[1]_{dist.} := -1;$

$ParetoSet[DimSet]_{dist.} := -1;$

for all $i := 2 \ldots DimSet - 1$

$ParetoSet[i]_{dist.} := ParetoSet[i]_{dist.} + \frac{(ParetoSet[i+1]_{Obj_i} - ParetoSet[i-1]_{Obj_i})}{(f_m^{max} - f_m^{min})};$

for all $i := 1 \ldots DimSet$

if $ParetoSet[i]_dist. = -1$ then

$ParetoSet[i]_{dist.} := \max_{i=1 \ldots DimSet} (ParetoSet[i]_{dist.});$

for all $i := 1 \ldots DimSet$

$ParetoSet[i]_{dist.} := ParetoSet[i]_{dist.} + \min_{i=1 \ldots DimSet}(ParetoSet[i]_{dist.} > 0);$

$ParetoSet[i]_{distance} := \frac{ParetoSet[i]_{dist.}}{ParetoSet[i]_{numEval}+1};$

Figure 5.2: Modified Crowding Distance Assignment Procedure(MCDA)

a new order. It can be totally random or using an heuristic rule which tries to minimize one objective. Hence two new sequences are created, the incomplete solution and the sequence of the removed elements.
For the *Construction* phase a variation of the NEH insertion scheme is used.

The main difference from the NEH heuristic is the use of Pareto dominance relationship to maintain not just one incomplete sequence at each iteration, but the whole set of nondominated sequences generated during the insertion process. More precisely, this procedure begins trying to insert the first job of sequence of the removed elements in each possible position of the incomplete current sequence. Let n be the initial sequence length and k the cardinality of the removed job set, we have $n-k+1$ possible insertion points and $n-k+1$ incomplete solutions of length $n-k+1$. Hence those sequences are evaluated and the dominated ones are removed, finally only $m_1 \leq n-k+1$ incomplete sequences are stored and used during the second iteration. At ith iteration a set of $m_i \leq (n-k+i-1)$ partial sequences are generated. This means that $(n-k+i-1)$ is an upper bound for the number of generated incomplete sequences at iteration i. Hence the total number of subsequences to evaluate in this phase is bounded by:

$$Bound_{eval} = \prod_{i=1}^{k} (n-k+i-1)$$

Anyway this bound is very far to be strict because at each iteration all the dominated incomplete sequences are removed. Anyway individuals with the same values of the objective functions but with different sequences of jobs are considered as distinct and kept in the current incomplete sequence set. According to our experience we observed that this Pareto greedy heuristic is very fast and effective.
The Improvement phase returns an archive of solutions which do not dominate each other. This archive is joined into the current solution set and the dominated solutions are removed. The fist time the improving phase is executed all the solutions in ISS are processed and the respective generated archives are joined before they are inserted into the current set of solutions. Afterwards, only one current solution is selected to be improved. This avoids that great improvements in the first iterations could dominate the seed solutions and compromise their promising search directions.

The local search phase

A simple and fast local search procedure has been demonstrated to be very useful to improve the quality of the solutions in single as well as in multi-objective framework. A swap or insertion move is used to implement the neighborhood for the local search. In the sequence of the current solution n_{sel} jobs are ran-

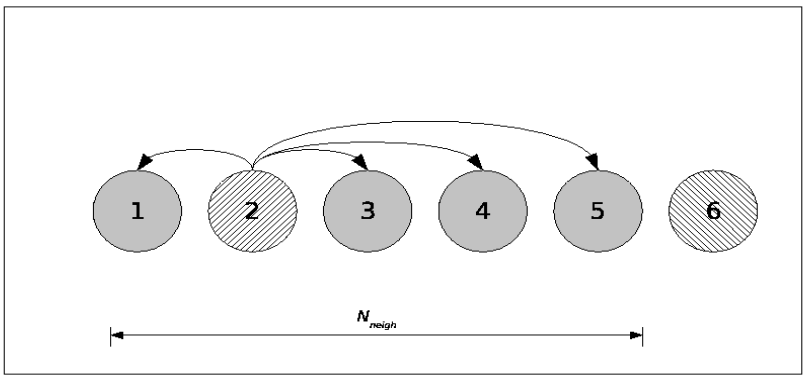

Figure 5.3: The figure represents an asymmetric neighbour for the LS.

domly selected and swapped with (or inserted in the positions of) n_{neigh} jobs chosen between its direct precursors and followers in the sequence. Depending on the position of the chosen job its neighborhood may or may not be symmetric. An example of an asymmetric neighborhood is depicted in figure 5.3. During the local search phase a small archive of $n_{sel} \times n_{neigh}$ solutions is generated, evaluated and joined at the working set. At each iteration the job with the best value of modified crowding is selected to be improved in the local search phase. To each nondominated individual is assigned a value of $n_{sel} \in [1 \ldots N/2]$ where N is the number of jobs of the current instance. Each time a sequence is selected for the local search procedure such a number is increased by one. Notice that when a new nondominated individual is found its value of n_{sel} is set to 1. The pseudocode of the local search prodecure is presented in figure 5.4

5.4 Experimental evaluation of the algorithm

Benchmark and performance measures

In all the experiments presented along this chapter we made use of two different sequence dependent instance sets based on the original instances of Taillard

---

LS(Seq,n_{sel},n_{neigh})

% Seq is the solution selected for the Local search.
% n_{neigh} is the maximum number of positions a job
% could be moved to the left or to the right.

$Pos = rand\_Vett(1, Length(Seq))$;
% A vector of different positions is randomly selected.

for all $i := 1 \ldots n_{sel}$
  for all $j := i \ldots n_{neigh\_left}$

    % Copying the current sequence.
    $Copy(OutSet[i][j], Sol)$;

    % Swapping the job in position $Pos[i]$ with that one in $Pos[i] - j$.
    $SWAP(Pos[i], Pos[i] - j, OutSet[i][j])$;

  for all $j := 1 \ldots n_{neigh\_right}$

    % Copying the current sequence.
    $Copy(OutSet[i][j], Sol)$;

    % Swapping the job in position $Pos[i]$ with that one in $Pos[i] - j$.
    $SWAP(Pos[i], Pos[i] + j, OutSet[i][j])$;

return $OutSet$

Figure 5.4: Local Search procedure (LS).

[193] and presented in [207] and used for the first time in [169]. Each set contains instances with several combinations of the number of jobs n and and number of machines m. The $n \times m$ combinations are: $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$. Setup times are selected to be respectively the 50% and the 125% of processing times (p_{ij}). So for example, if the p_{ij} in Taillard's instances are generated from a uniform distribution in the range $[0 - 99]$ in the first set called SSD50 setup times are uniformly distributed in range $[0 - 49]$ while in the second set (SSD125) their range is $[0 - 124]$. Such benchmark sets are

finally augmented by adding weights and due dates for each job. The weights are drawn from a uniform $U[1, 10]$ distribution. As regards the due dates for the tardiness criterion we use the same approach of [82]. In this work, a tight due date d_j is assigned to each job $j \in N$ following the expression: $d_j = P_j \times (1 + random \cdot 3)$ where $P_j = \sum_{i=1}^{m} p_{ij}$ is the sum of the processing times over all machines for job j and $random$ is a random number uniformly distributed in $[0, 1]$. This method of generating due dates results in very tight to relatively tight due dates depending on the actual value of $random$ for each job, i.e., if $random$ is close to 0, then the due date of the job is going to be really tight as it would be more or less the sum of its processing times. A total of 220 instances are used to carry out our experiments. These instances can be downloaded from `http://soa.upv.es`.
As regards the performance measures, all considerations made in the previous chapter remain valid. In particular measures so-called "Pareto-compliant " [224, 225] seem the most appropriate to be used. Among these we selected the hypervolume (I_H) and the multiplicative Epsilon (I_ε^1) indicators which represents the state-of-the-art as far as quality indicators are concerned. For more details see 4.3.

Adaptation of existing metaheuristics

In capther 3 we reviewed a large number of papers dealing with multi-objective flowshop. In chapter 4 the most performing among them have been reimplemented. We compared them with the best known general purpose multi-objective algorithms. In this way we could rank those algorithms identifying the best ones. To do this we needed a huge number of hours of computing due to the high number of algorithms to evaluate . In this chapter we follow a different strategy. We first reduced the number of algorithms by means of a preliminary test on a reduced set of 110 instances and, depending on the attained results, we selected the best ten algorithms. Note that such algorithms presented also the best results for permutation flowshop without setups. Seven of them are specifically designed to tackle flowshop problems while the remaining three are generic optimization algorithms. Such algorithm are summarized in table 5.1.

Acronym	Year	Author/s	Type
MOGA_Murata	1996	Murata et al.	Genetic algorithm. Specific
PESA	2000	Corne et al.	Genetic algorithm. General
PESAII	2001	Corne et al.	Genetic algorithm. General
CMOGA	2001	Murata et al.	Genetic algorithm. Specific
MOTS	2004	Armentano and Arroyo	Tabu search. Specific
$\varepsilon-$NSGAII	2005	Kollat and Reed	Genetic algorithm. General
MOGALS_Arroyo	2005	Arroyo and Armentano	Genetic algorithm. Specific
MOSA_Varad.	2005	Varadharajan and Rajendran	Simulated annealing. Specific
PGA_ALS	2006	Pasupathy et al.	Genetic algorithm. Specific
PILS	2007	Geiger	Iterated local search. Specific

Table 5.1: Re-implemented methods for the SDST multi-objective flowshop.

A short description of each algorithm is provided in section 4.2.
During the preliminary phase we noted that the algorithm proposed by Vadharajan and Rajendran [210] (MOSA_Varad.) achieved worst positions respect to those attained in the previous chapter regarding the problem without setups. The presence of sequence dependent setups makes surely the flowshop problem more difficult to solve, but we noted that the annealing process employed in such algorithm makes it stop before the time limit is reached. This is why we decided to implement an improved version of such procedure (we call it MOSA_Varad_M) able to entirely exploit the available time. In the original version such algorithm contains 4 nested loops. The inner one is a simulated annealing procedure and is repeated a fixed number of times hence, changing its initial *Temperature* (T), it is possible to control the algorithm completion time. Since for each instance we assign a certain amount of time depending on the number of machines an jobs, it is relatively simple modify T in such a way that the algorithm is executed exactly within the given time windows.
Implementation details including operators and parameter values are presented in tables 4.2 and 5.2.

Parameter	Values
k	5
Destruction policy	a block of k consecutive positions is selected starting from a random position
RI rearrangement	using a heuristic for each objective
LS type	Insert
n_{neig}	5
Selection of jobs for local search	Random

Table 5.2: Details and parameter of IPG algorithm.

Experimental results

The stopping criterion for all algorithms is given by a time limit depending on the size of the considered instance. The algorithms are stopped after a CPU running time of $n \cdot m/2 \cdot t$ milliseconds, where t is an input parameter. In this way we assign more time to larger instances that are obviously more difficult to solve.
Every algorithm is run 10 different independent times (replicates) on each instance with two different stopping criteria: $t = 150$ and 200 milliseconds. This

means that for the largest instances of 200×20 a maximum of 400 seconds of real CPU time are allowed. For every instance, stopping time and replicate we use the same random seed as a common variance reduction technique.
We run every algorithm on a cluster of 12 identical computers with Intel Core 2 Duo E6600 processors running at 2.4 GHz with 1 Gbyte of RAM. For the tests, each algorithm and instance replicate is randomly assigned to a single computer and the results are collected at the end.
A total of 52,800 data points are collected if we consider the 12 algorithms, 220 instances, 10 replicates per instance and two different stopping time criteria. In reality, each data point is an approximated Pareto front containing a set of vectors with the objective values.
From the $12 \cdot 10 = 120$ available Pareto front approximations for each instance, the best non-dominated Pareto front is found and stored. Additionally, a set of best Pareto fronts is stored for each one of the two employed stopping times. These last Pareto fronts are also used for obtaining the reference points for the hypervolume (I_H) indicator and are fixed to 1.2 times the worst known value for each objective. Also, these best Pareto fronts are also used as the reference set in the multiplicative epsilon indicator (I_ε^1).
Tables 5.3 and 5.4 contain average values but although each value is attained by means of a very large number of data points, it is still necessary to carry out a comprehensive statistical experiment to assess if the observed differences in the average values are statistically significant. A total of 16 different experiments are carried out. We do parametric ANOVA analyses as well as non-parametric Friedman rank-based tests on both quality indicators and for the two different stopping criteria. The utility of showing both parametric as well as non-parametric tests consists in improving the soundness of our conclusions. For more information about parametric and non-parametric tests the reader is refereed to [41] and [133].
We carry out eight multi-factor ANOVAS where the type of instance is a controlled factor with 11 levels (instances from 20×5 to 200×20). The algorithm is another controlled factor with 12 levels. The response variable on each experiment is either the hypervolume or the epsilon indicator. Lastly, there is one set of experiments for each stopping time. Considering that each experiment contains 13,200 data points, the three main hypotheses of ANOVA: normality, homocedasticity and independence of the residuals are easily satisfied.
To compare results, a second set of eight experiments are performed. In this case, non-parametric Friedman rank-based tests are carried out. Since there are 12 algorithms and 10 different replicates, the results for each instance are ranked between 1 and 120. A rank of one represents the best result for hy-

pervolume or epsilon indicator. We are performing four different statistical tests on each set of results (for example, for 150ms CPU time we are testing ANOVA and Friedman on both quality indicators). Therefore, a correction on the confidence levels must be carried out since the same set of data is being used to make more than one inference. We take the most conservative approach, which is the Bonferroni adjustment, and we set the adjusted significance level α_s to $\frac{\alpha}{4} = \frac{0.05}{4} \simeq 0.01$. This means that all the tests are carried out at a 0.01 adjusted confidence level for a real confidence level of 0.05.

First instance set: SSD50

For the parametric tests we use Tukey's Honest Significant Difference (HSD) intervals which counteract the bias in multiple pairwise comparisons. Similar Honest Significant Difference (HSD) intervals are used for the non-parametric tests.
Figures 5.5 and 5.9 show the means plot for the ranks of the epsilon indicator and the means plot for the factor algorithm in the ANOVA for the epsilon indicator response variable, respectively. Both figures refer to the 150 CPU time stopping criterion. As can be seen, the non-parametric test is less powerful. Not only are the intervals much wider (recall that overlapping intervals indicate a non-statistically significant difference) but ranking neglects the differences in the response variables. The considered images show a very similar result, i.e. that IPG algorithm is far better than any other. Moreover the remaining algorithm maintain the same relative positions in both figure with the exception of MOGALS_Arroyo, PESAII and PILS. MOGALS_Arroyo and PESAII in fact, have inverted positions but however they don't present a significant statistical difference while PILS is shown to be far better in rank than in ANOVA. The reason behind this behavior is that PILS is better for many small instances with a small difference in epsilon indicator. However, it is much worse for some other larger instances. When one transforms this to ranks, PILS obtains a better rank more times and hence it appears to be better, when in reality it is marginally better more times but significantly worse many times as well. Almost identical results are presented in figures 5.7 and 5.11 where are presented the means plot for ranks and ANOVA for the epsilon indicator response variable but with the 200 CPU time stopping criterion.
In figures 5.6, 5.8, 5.10 and 5.12 the hypervolume indicator is considered. They refer to the 150 and 200 CPU time stopping criteria. Again IPG is far better than any other and PILS shows to be better in rank than in ANOVA. Notice that PESA and PESAII present not observable statistical differences while

MOSA_Varad_M has a better position in ANOVA test respect its rank position. This is due to the fact that it shows to be good only for the subset made of larger instances. Finally notice that either Friedman and ANOVA return the same ranking for both stopping criteria.

Second instance set: SSD125

Figures 5.13, 5.15, 5.17 and 5.19 show the means plot for the ranks of the epsilon indicator and the means plot for the factor algorithm in the ANOVA for the epsilon indicator response variable, respectively. Figures refer to the 150 and to the 200 CPU time stopping criteria. As it is easy to see the best algorithm result to be the IPG with high statistical difference from MOGALS_Arroyo algorithm that is the second in the ranking. For set SSD125 we can make the same considerations made for set SSD50, the only differences consist in the higher positions MOTS, MOSA_Varad and MOSA_Varad_M that seems to suggest the presence in them of an embedded strategy able to effectively handle large setup times. Finally figures 5.14, 5.16, 5.18 and 5.20 show the means plot for the ranks of the hypervolume indicator and the means plot for the factor algorithm in the ANOVA for the hypervolume indicator response variable, respectively. Figures refer to the 150 and to the 200 CPU time stopping criteria. Here again IPG results to be the most performing and in second position we find MOGAL_Arroyo algorithm. A group of algorithms with incomparable performances is made of PESA, PESAII and MOTS, while MOSA_Varad_M holds the third positions in ANOVA tests and the sixth in the Friedman one. Again this is due to the fact that such algorithm achieves very good results in terms of hypervolume only for larger instances where other algorithms stuck because of their extensive local searches. Notice that hypervolume and epsilon indicator express different properties of a Pareto front. This is the reason of little differences in the ranking positions for some algorithms.

5.5 Conclusions and future research

In this chapter we have presented an algorithm of new conception which, compared with the most performing algorithms presented in literature, demonstrated to be the new state-of-the-art in the field of bicriteria permutation flowshop with sequence dependent setup times.
The comparative evaluation not only includes flowshop-specific algorithms but also adaptations of other general methods proposed in the multi-objective

optimization literature. A set of benchmark instances, based on the well known benchmark of [193] has been employed and a comprehensive statistical analysis of the results has been conducted with both parametric as well as non-parametric techniques. Overall, our Iterated Pareto greedy can be regarded as the best performer under our experimental settings. Another consistent performer is the genetic local search method MOGALS_Arroyo of [14]. Our adapted versions of PESA and PESAII from [45] and [44], respectively, have shown a very good performance over many other flowshop-specific algorithms. The recent PILS method from [67] has shown a promising performance for small instances while the multi-objective simulated algorithm of [210], MOSA_Varad. has shown good performance for the larger ones.

Tables and figures

Results for the first group of instances where setup times are on average 50% the length of the processing times

Table results

	Time	150			200	
#	Method	I_H	I_ε^1	Method	I_H	I_ε^1
1	IPG	1.197	1.119	IPG	1.190	1.124
2	MOGALS_Arroyo	1.102	1.185	MOGALS_Arroyo	1.095	1.185
3	PESAII	1.071	1.187	PESAII	1.064	1.191
4	PESA	1.066	1.198	PESA	1.057	1.203
5	MOSA_Varad_M	1.019	1.308	MOSA_Varad_M	1.008	1.313
6	MOTS	1.013	**1.255**	MOTS	1.000	**1.263**
7	PGA_ALS	0.994	**1.230**	PGA_ALS	0.975	**1.238**
8	MOSA_Varad	0.937	1.361	MOSA_Varad	0.910	1.377
9	MOGA_Murata	0.879	**1.339**	MOGA_Murata	0.872	**1.344**
10	PILS	0.840	1.390	PILS	0.868	1.375
11	$\varepsilon-$NSGAII	0.818	**1.349**	CMOGA	0.814	**1.357**
12	CMOGA	0.814	1.355	$\varepsilon-$NSGAII	0.814	**1.349**

Table 5.3: Results for the makespan and total weighted tardiness criteria. Average quality indicator values for the 12 algorithms tested under the two different termination criteria. Instance group where setup times length is 50% that of the processing times. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H.

Non-paremetric rank results

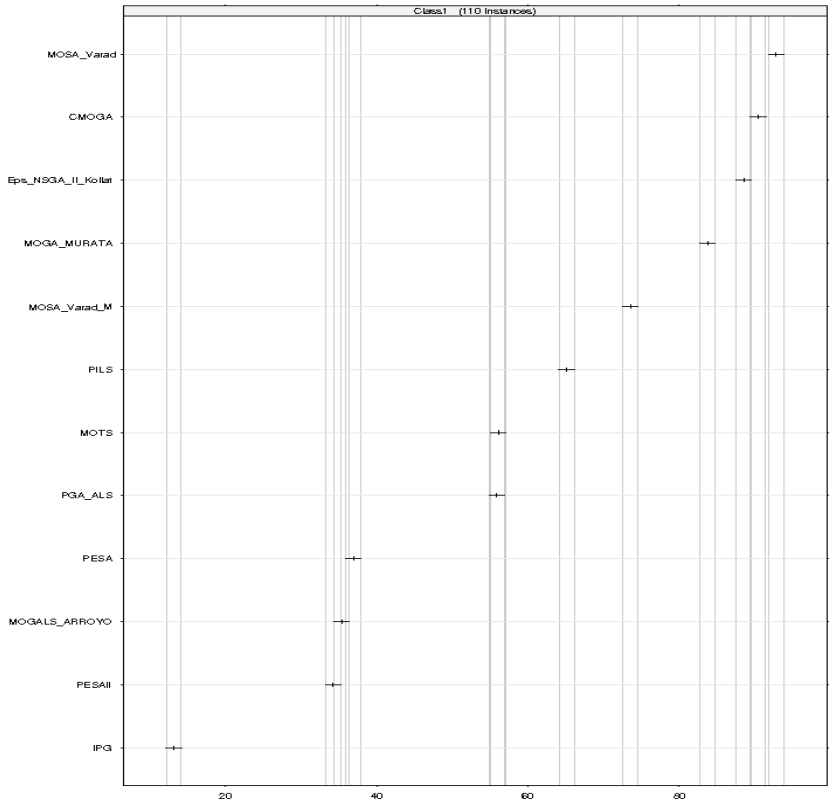

Figure 5.5: First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

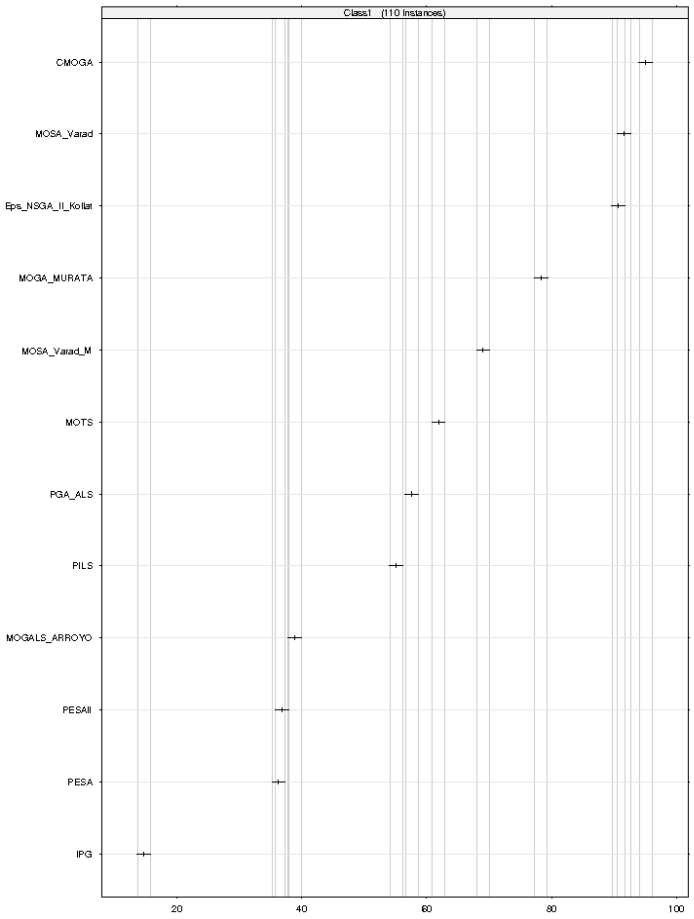

Figure 5.6: First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

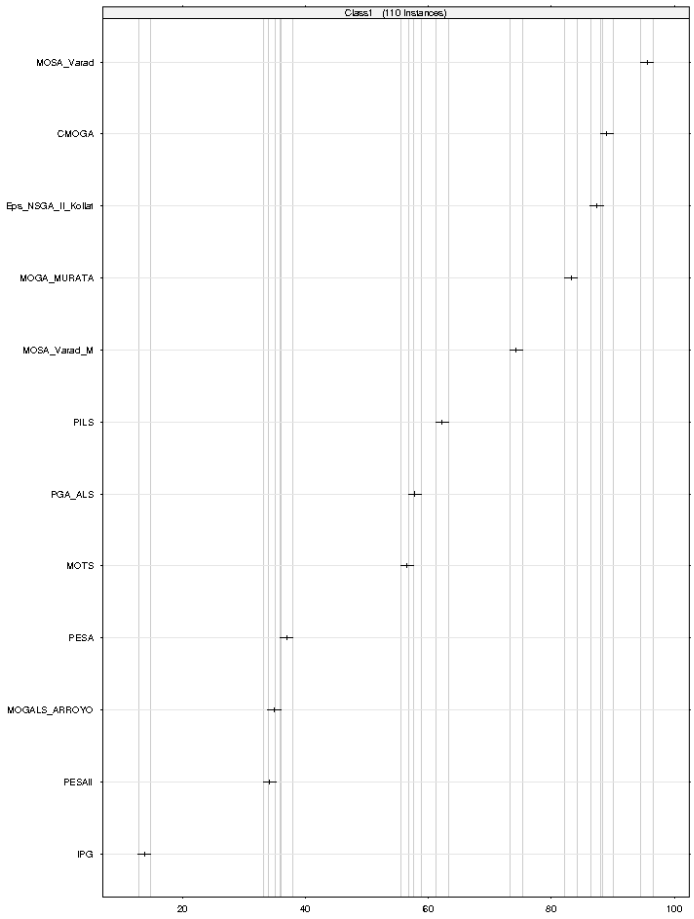

Figure 5.7: First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

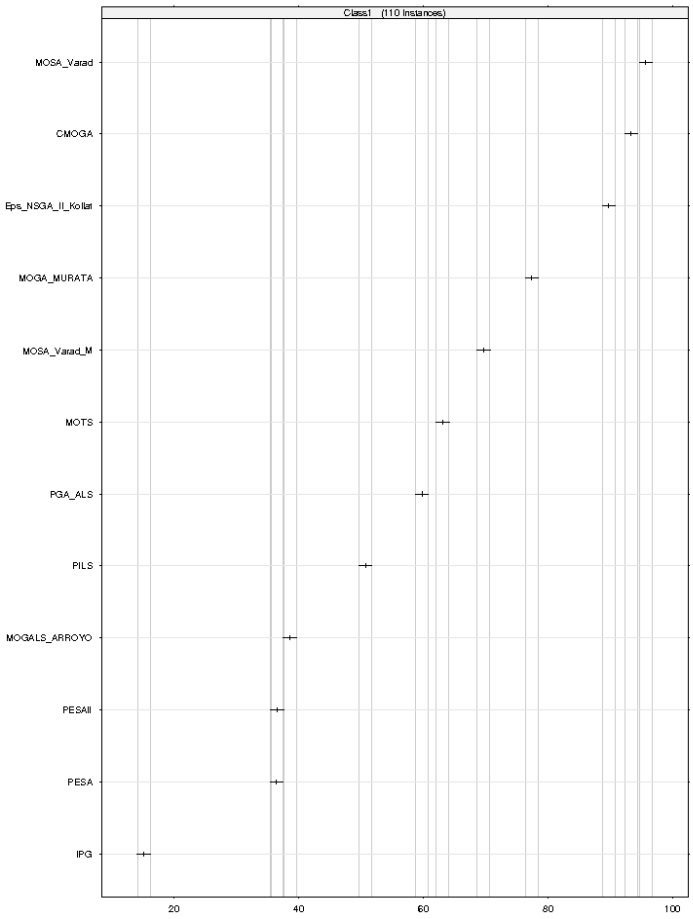

Figure 5.8: First instance set where setup times length is 50% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

Parametric ANOVA results

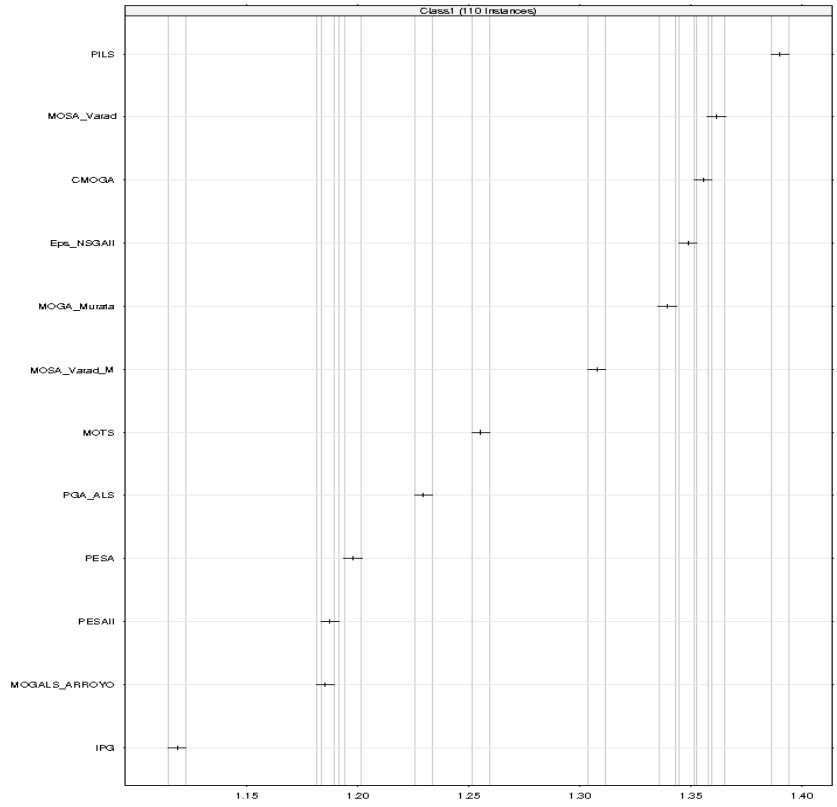

Figure 5.9: First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

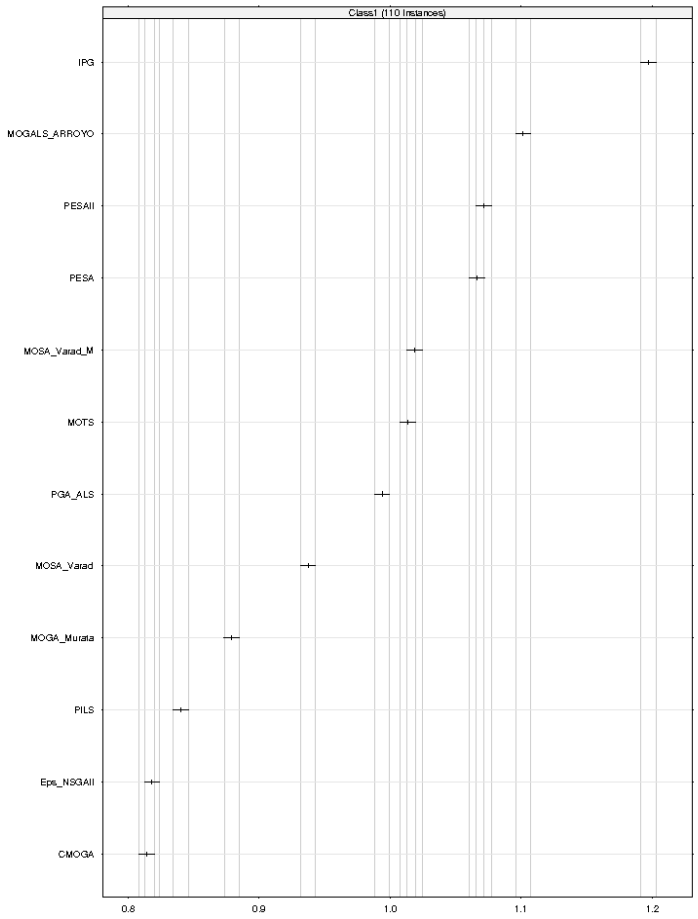

Figure 5.10: First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

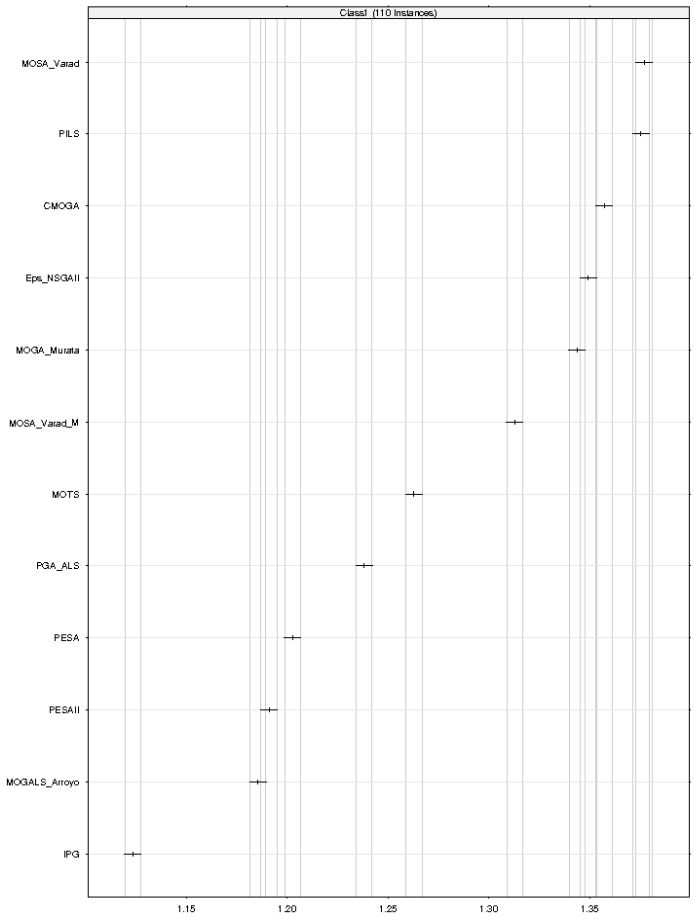

Figure 5.11: First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

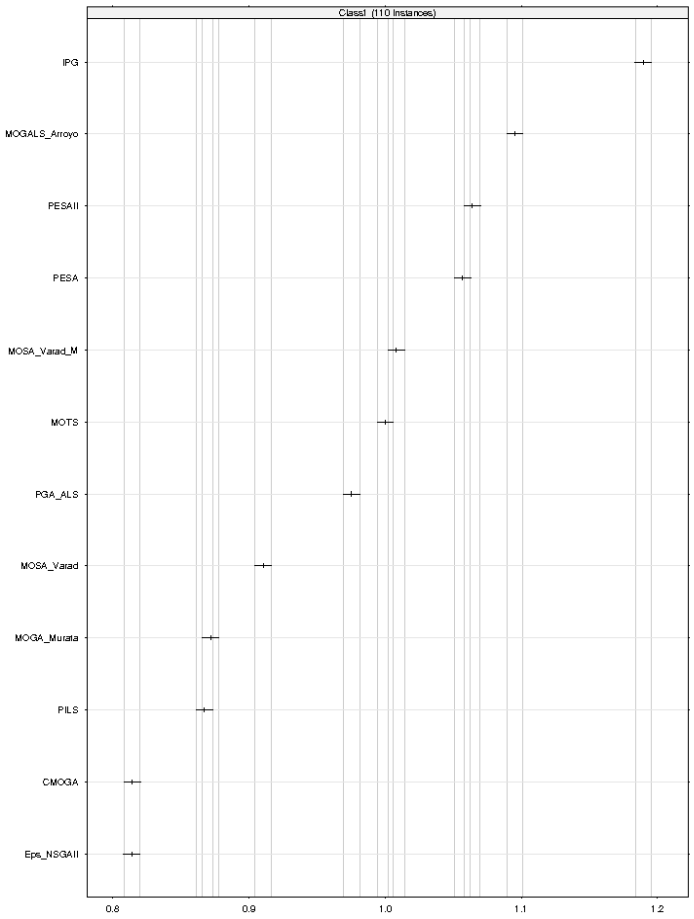

Figure 5.12: First instance set where setup times length is 50% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

Results for the second group of instances where setup times are on average 125% the length of the processing times

Table results

	Time	150			200	
#	Method	I_H	I_ε^1	Method	I_H	I_ε^1
1	IPG	1.215	1.115	IPG	1.211	1.116
2	MOGALS_Arroyo	1.102	1.177	MOGALS_Arroyo	1.101	1.174
3	MOSA_Varad_M	1.048	1.266	MOSA_Varad_M	1.041	1.269
4	MOTS	1.036	**1.232**	PESAII	1.030	**1.209**
5	PESAII	1.031	**1.211**	MOTS	1.028	1.234
6	PESA	1.030	1.220	PESA	1.026	**1.220**
7	MOSA_Varad	0.953	1.326	PGA_ALS	0.932	1.266
8	PGA_ALS	0.945	**1.261**	MOSA_Varad	0.930	1.338
9	PILS	0.831	1.394	PILS	0.861	1.373
10	MOGA_Murata	0.817	**1.377**	MOGA_Murata	0.817	1.376
11	ε−NSGAII	0.745	1.390	ε−NSGAII	0.746	1.388
12	CMOGA	0.735	1.406	CMOGA	0.744	1.401

Table 5.4: Results for the makespan and total weighted tardiness criteria. Average quality indicator values for the 12 algorithms tested under the two different termination criteria. Instance group where setup times length is 125% that of the processing times. Each value is averaged across 110 instances and 10 replicates per instance (1,100 values). For each termination criteria level, the methods are sorted according to I_H.

Non-parametric rank results

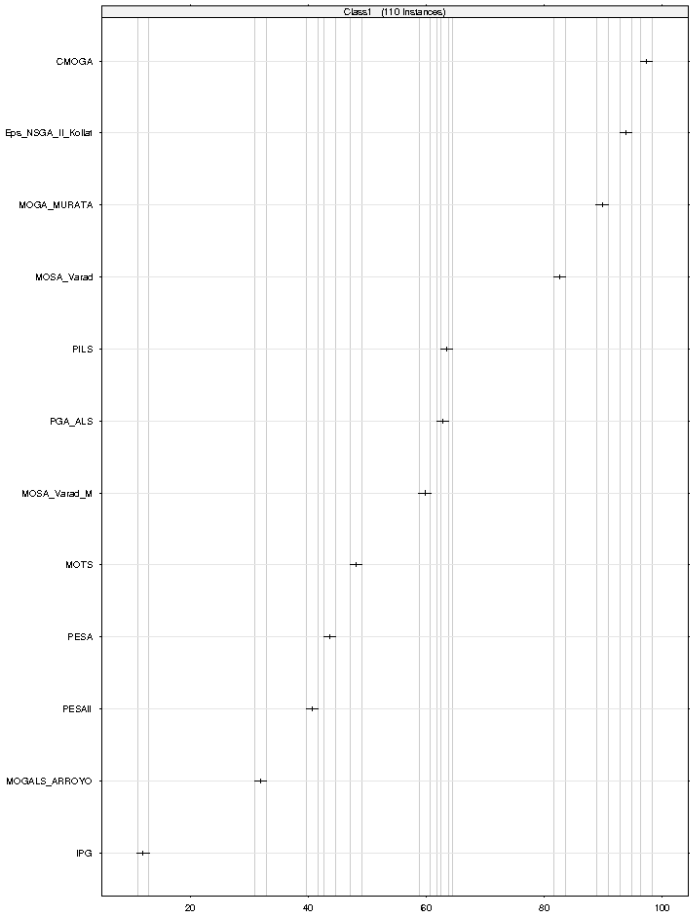

Figure 5.13: Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

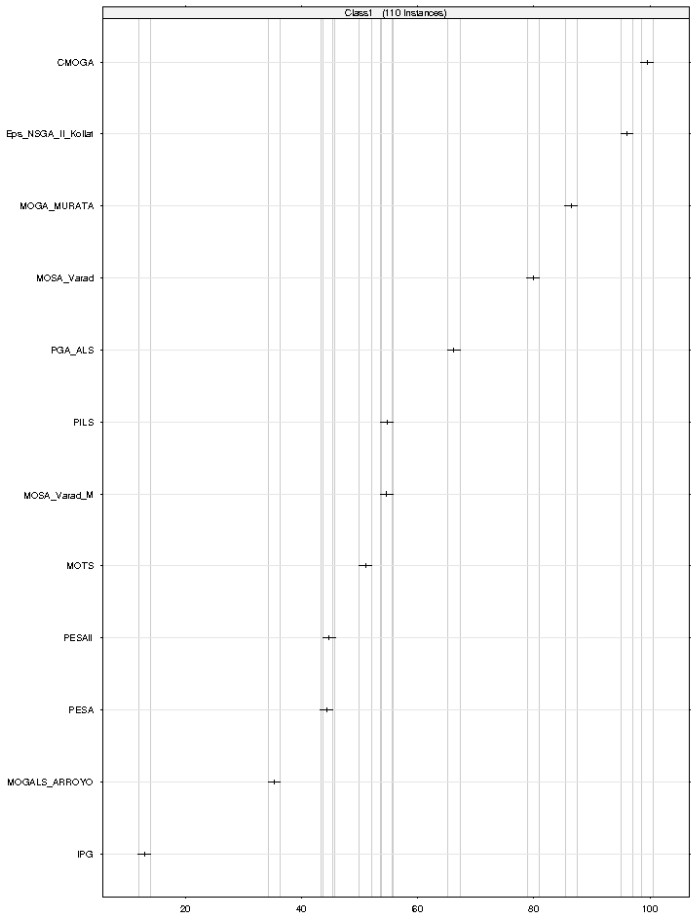

Figure 5.14: Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

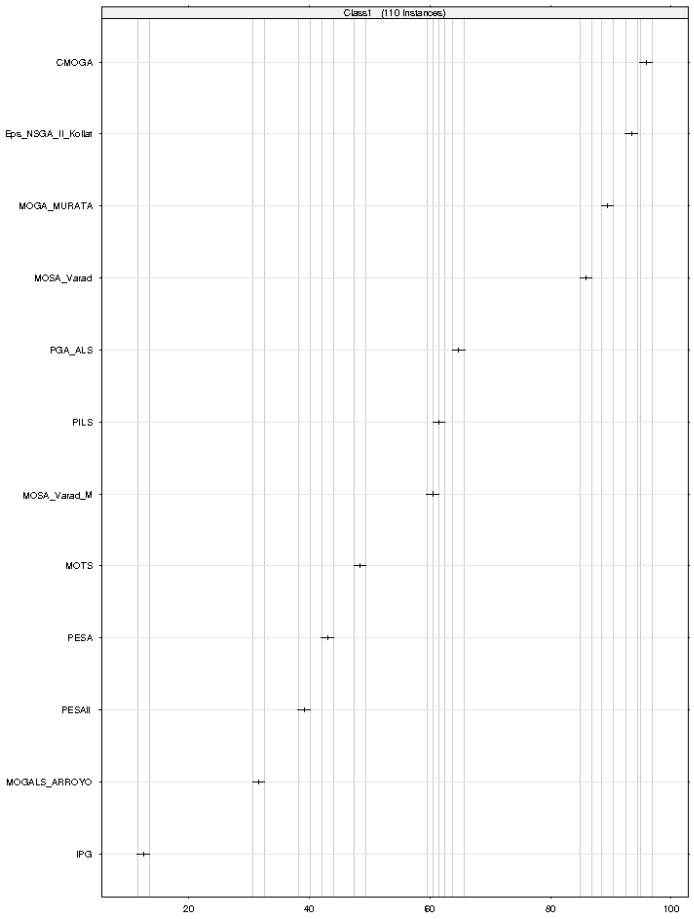

Figure 5.15: Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

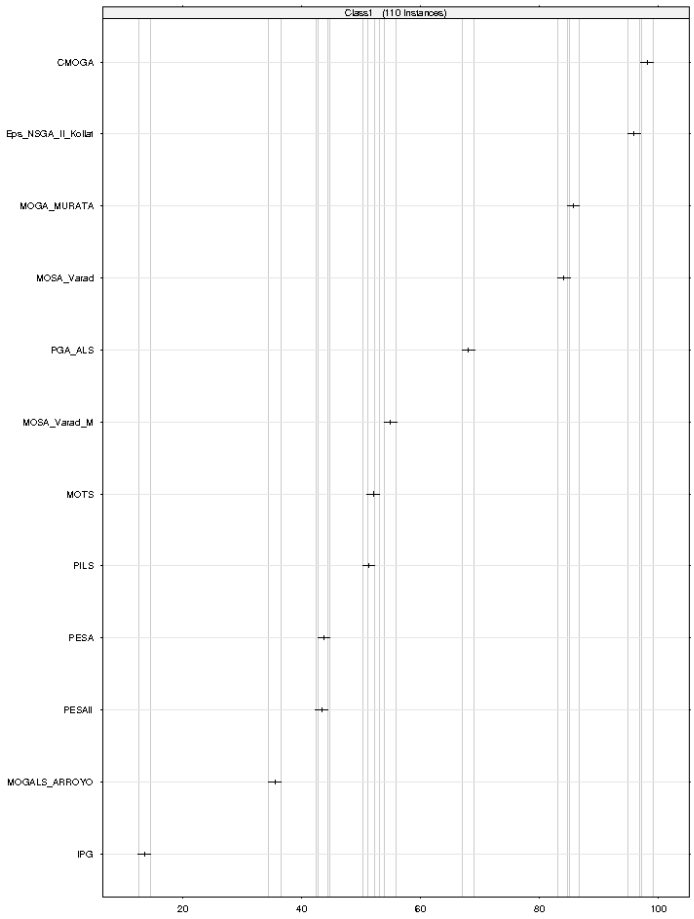

Figure 5.16: Second instance set where setup times length is 125% the length of processing times. Means plot and MSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the Friedman Rank-based test. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

Parametric ANOVA results

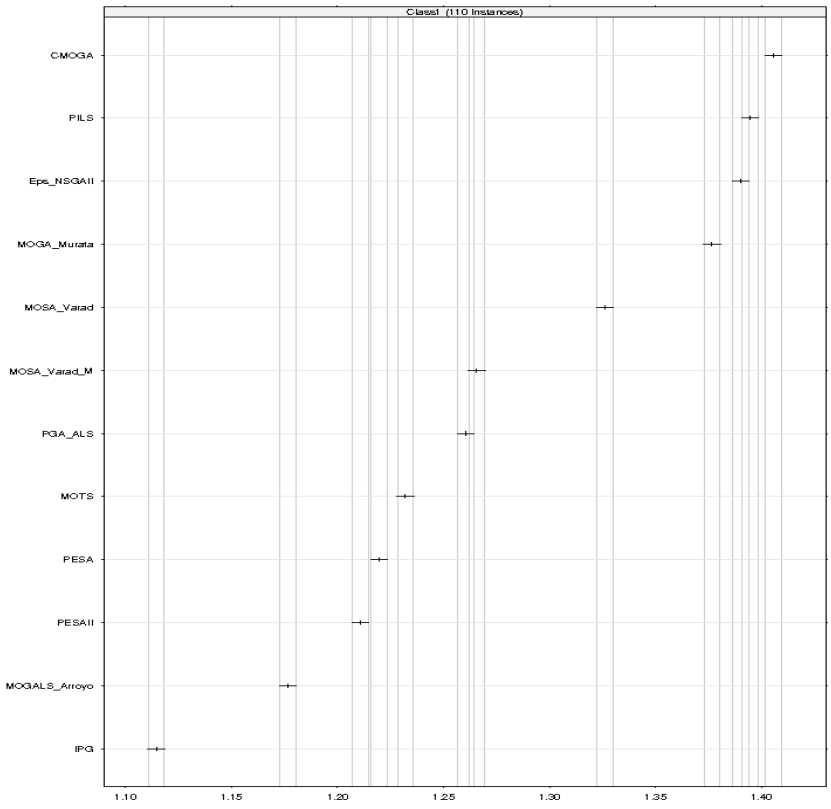

Figure 5.17: Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

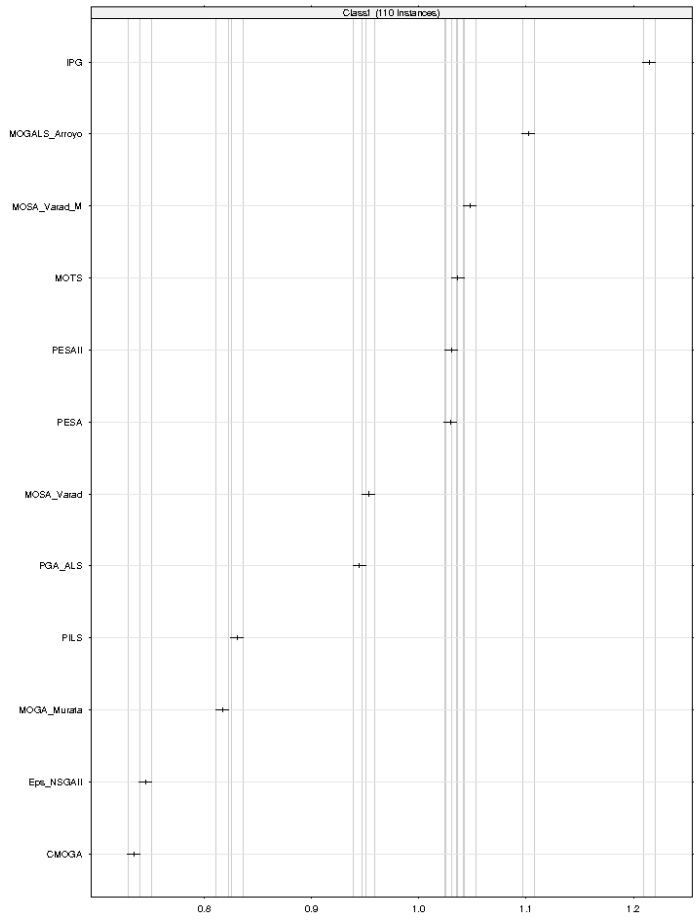

Figure 5.18: Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 150 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

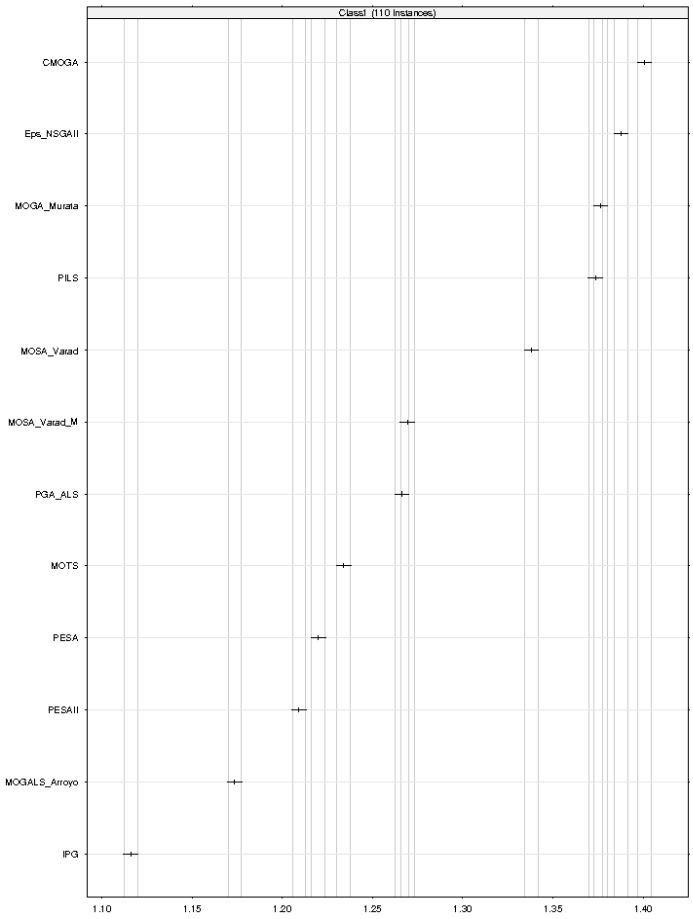

Figure 5.19: Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Epsilon indicator response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

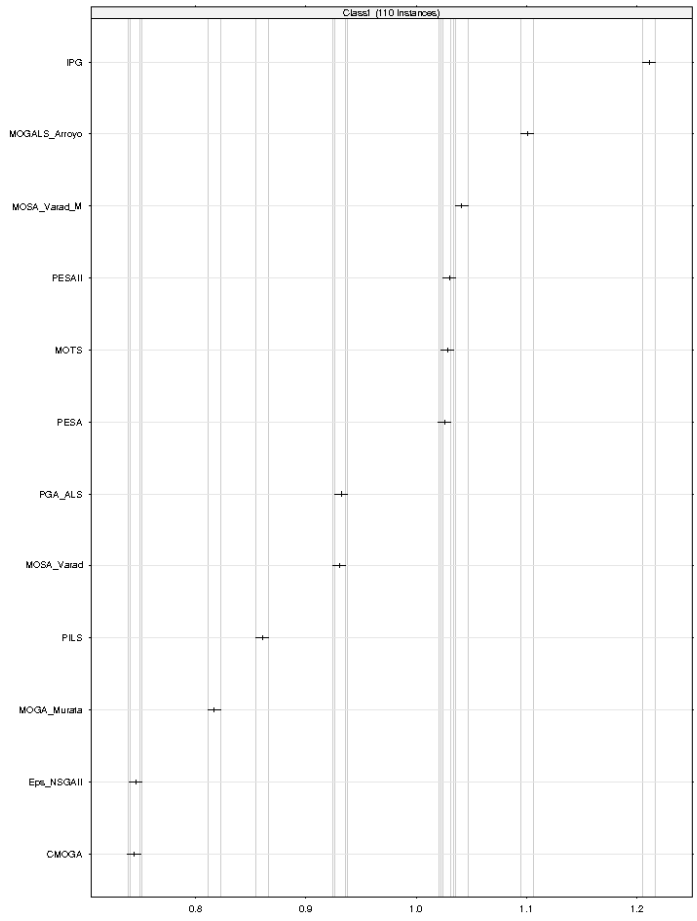

Figure 5.20: Second instance set where setup times length is 125% the length of processing times. Means plot and Tukey HSD confidence intervals ($\alpha_s = 0.01$, $\alpha = 0.05$) for the algorithm factor in the ANOVA experiment. Hypervolume response variable and 200 CPU time stopping criterion. Makespan and total weighted tardiness criteria.

Chapter 6

A bi-objective coordination setup problem in a two-stage production system

This chapter addresses another real life scheduling problem arising in the coordination between two consecutive departments of a production system, where parts are processed in batches, and each batch is characterized by two distinct attributes. Due to the lack of interstage buffering between the two stages, these departments have to follow the same batch sequence. In the first department, a setup occurs every time the first attribute of a new batch is different from the one of the previous batch. In the downstream department, there is a setup when the second attribute changes in two consecutive batches. The problem consists in finding a batch sequence optimizing the number of setups paid by each department. This case results in a particular bi-objective combinatorial optimization problem. We present a geometrical characterization for the feasible solution set of the problem, and we propose three effective metaheuristics, as shown by an extensive experimental campaign. The proposed approach can be also used to solve a class of single-objective problems, in which setup costs in the two departments are general increasing functions of the number of setups.

6.1 Introduction

This work deals with a problem of real-life manufacturing interest: The coordination of two consecutive production departments. Each department consists of a flexible machine, the first one deals with shaping items of raw wooden panels; the second concerns the painting of the just shaped items. To avoid unnecessary costs, each department works with batches of jobs, in which every job has the same shape and color. Due to limited interstage buffering between the two stages, these departments have to follow the same batch sequence. Hence, this problem belongs to the class of permutation sequencing problems. When two consecutive jobs with different features have to be processed, at least one department must pay a setup, in order to reconfigure its own machine, i.e., changing tools. The effort needed to accomplish a setup (in terms of manpower, machine reconfiguration, etc.) is almost the same in each department. This implies that the same cost must be paid for a setup in the first and in the second department. Hence, each department have to organize its own operations in order to minimize the number of setups, i.e, the reconfigurations of cutting and painting tools, respectively. This combinatorial problem is inherently bi-objective [57, 197] because each department has to minimize its setup costs, while from a global point of view, the minimization of the total setup cost must be pursued.
Note that, the considered problem can be modeled as a *tool switching* problem on a single machine with two classes of tools (cutting and painting tools), in which a given set of jobs (each of them requiring exactly one tool of each class) must be sequenced on the machine minimizing the number of tool switching.
In this chapter, a graph based model, first introduced in [4], is used to obtain a lower bound and an upper bound of the number of setups for each department. At an operational level, the goal is to find a, possible large, set of sequences of batches solving a trade-off between different objectives (i.e., the number of setups paid by each department). We tackled this multi-objective problem using a metaheuristic approach. The basic idea of this approach is to obtain a good estimation of the Pareto optimal front for the bi-objective problem. At this aim, first the total setup number of the production system is minimized, then a different procedure is employed to spread the setups over departments keeping constant the total number of setups. Different procedures to guide the algorithms toward the discovering of a greater part of the Pareto optimal front are implemented. The proposed approach returns a set of non-dominated points achieving a high probability of covering almost all the Pareto optimal front in an acceptable computation time.

The chapter is organized as follows. In Section 6.2, literature results and applications are discussed. In Section 6.3, the industrial context is described, a formal description of the problem and a geometrical characterization of the feasible solution set are given. Moreover, we also show that these geometrical properties are also useful to solve a class of single objective problems, in which the setup costs in the two departments are general increasing functions of the number of setups. In Section 6.4, three metaheuristic algorithms are presented, and in Section 6.5 a large sample of experiments are reported, showing the effectiveness of the proposed approaches. Finally, in Section 6.6, conclusions are drawn.

6.2 Literature and applications

Minimizing the impact of setups (i.e., changeovers) has been widely described as a main component of modern production management strategies [192]. Pursuing high changeover performances is a way to enable agile and responsive manufacturing processes by improving line productivity and reducing downtime losses [127]. This aspect of the production management, involving both organizational and economic points of view, has received an increasing attention also in fields as applied mathematics and operations research. In particular, over the past few decades, there has been a significant effort associated with reducing the time required to perform setups and developing suitable changeover modeling processes. This process can be quite complicated, but yields important benefits in planning and scheduling a production system [192] improving both the production capacity and the system manageability. Important surveys on changeovers and setups are proposed in [10, 155] where classifications of setup types are also given. While in [166] a review of heuristics for setup problems in serial systems is presented.
The problem addressed in this chapter refers to the furniture production case first presented in [4]. The importance of this kind of problems is documented by the large amount of literature dealing with the minimization of changeover or setup costs (see, for example, [183, 6, 204], for real world industrial applications). It is easy to see, that the problem we address can be modeled as a *tool switching* problem on a single machine with two classes of tools (cutting and painting tools), in which a given set of jobs (each of them requiring exactly one tool of each class) must be sequenced on the machine minimizing the number of tool switching. Tang and Denardo [194] address a similar problem arising in the metal working industry. In that context, they propose heuristic algo-

rithms. For the same problem, Laporte et al.[107] proposed exact approaches able to solve instances up to 25 jobs. Recently, Agnetis et al. [3] considered the problem of grouping parts into batches and scheduling them in a single machine minimizing the total number of tool switches, where at most k parts may be accommodated in the machine at the same time.

6.3 Problem description and formulation

This chapter addresses a problem arising in the coordination between two consecutive production departments of an industrial system [4], in which a large number of different slabs of wood are cut, painted and assembled to build kitchen furniture. Due to the lack of interstage buffering, departments must process the batches in the same order. The two departments are the cutting and the painting departments and batches are characterized only by shapes and colors. In the cutting department, a setup occurs when a batch has a different shape from the preceding one (cutting tools and machinery must be reconfigured). Similarly, in the painting station a setup occurs when a new color is used (the equipment and the pallets must be thoroughly cleaned in order to eliminate the residuals of the previous color). Since each item to be produced has its own shape and color, all the items sharing the same shape and color form a single batch. In fact, there is no convenience, on either side, in splitting such batches, and the actual cardinality of each batch is of no interest in this context. Hence, in processing two consecutive batches, at least a department has to pay a changeover. Therefore this kind of setups are considered as sequence dependent [10, 155]. Each given sequence of batches results in a number of setups to be paid by each department. Since the cost of a setup, in terms of manpower, machine reconfiguration operations, etc., is almost the same in the two departments, the problem is to find a collection of batch sequences minimizing the costs related to changeovers (i.e., setups). For this reason, the number of setups is a meaningful index of performance. However, minimizing the changeover cost for a department often implies the increasing of the setup costs for the other one. Therefore this problem is inherently multi-objective. More specifically, we analyze a bi-objective version of the described problem, where the two objectives we consider are the minimization of the setup number paid by each department. Even if the single-objective problem requiring to minimize the setup of a single department is clearly an easy task, this bi-objective problem is NP-hard [4] and it calls for a heuristic solution approach. Hence, our aim is to find a good approximation of the Pareto optimal front

in a reasonable computation time. Thus, in general, a solution of the problem is a set of trade-off solutions, i.e., non-dominated solutions. In this way, it is possible to offer to the Decision Maker a wide range of sequences holding the quality unchanged. In this chapter, we propose three metaheuristic approaches for the problem which are able to attain, even for large-size instances, a good compromise between solutions quality and computational effort.
The problem we consider can be more formally formulated as follows. Let B be a set of batches to be produced. The batches must be processed by two departments of the plant, called D_S and D_C, in the same order. Each batch is characterized by two attributes, say *shape* and *color*. Let S and C denote the sets of all possible shapes and colors respectively. We denote the shapes as s_i, $i = 1, \ldots, |S|$, and the colors as c_j, $j = 1, \ldots, |C|$. Each batch is therefore defined by a pair (s_i,c_j). If batch (s_i,c_j) is processed immediately after batch (s_h,c_k), a setup is paid in department D_S if $s_h \neq s_i$, and a setup is paid in department D_C if $c_k \neq c_j$. We can represent the input as a bipartite graph, $G = (S, C, B)$, where nodes in S correspond to shapes, nodes in C to colors, and each edge of B corresponds to a batch to be produced. The problem is to sequence the batches in a profitable way from the viewpoint of the number of setups. This means that we must find some particular ordering σ of the edges of G. If two consecutive edges (i, j) and (h, k) in σ have no nodes in common, then both departments have to pay a setup when switching from batch (i, j) to batch (h, k). We refer to this as a *global changeover*. On the other hand, if $i = h$ ($j = k$), only department D_C (D_S) pays a setup. This is called *local changeover*. For a given sequence σ, we can therefore easily compute the number of setups incurred by each department, and we call them $NS(\sigma)$ and $NC(\sigma)$ respectively. In fact, let δ_{ih} be equal to 1 if $i \neq h$ and 0 otherwise, and let $s(\sigma(q))$ denote the shape of the q-th batch in the sequence σ, and let $c(\sigma(q))$ denote its color. Hence,

$$NS(\sigma) = 1 + \sum_{q=1}^{|B|-1} \delta_{s(\sigma(q)),s(\sigma(q+1))} \tag{6.1}$$

$$NC(\sigma) = 1 + \sum_{q=1}^{|B|-1} \delta_{c(\sigma(q)),c(\sigma(q+1))} \tag{6.2}$$

The two objectives of the problem addressed in this chapter are exactly $NS(\sigma)$ and $NC(\sigma)$ (for sake of simplicity, we use NS and NC when the context does not require to remark the dependence of σ). The problem we address, denoted

as NC-NS problem, consists in finding a set of non-dominated batch sequences with respect to the two objectives NS and NC to be minimized.

Geometrical properties of the feasible solution set of the NC-NS problem

In this section, a geometrical characterization of the feasible solution set of the NC-NS problem is presented. Such characterization is useful to determine a good estimation of the Pareto optimal front.
In Detti et al. [53], a heuristic approach for another bi-objective optimization problem with similar combinatorial structure, but with a different pair of objective functions has been presented. In particular, in Detti et al. [53] the minimization of the following criteria is considered: $SUM = NS + NC$ and $MAX = \max\{NS, NC\}$. In the following, we refer to this bi-objective problem as SUM-MAX problem. The above criteria strongly depend on those considered in the problem under study in this chapter (i.e., NS and NC). The solution set of the SUM-MAX problem turns out to be strictly related to the solution set of the NC-NS problem. The computational experiments presented in Detti et al. [53] carried out on a wide set of instances highlight that the Pareto optimal front always contains few (often only one) non-dominated points. This is due to the strong correlation between the objectives SUM and MAX. Note that, in general, the same point in the space of criteria may correspond to several distinct batch sequences σ. In the following, we give a geometric description of the solution set of the NC-NS problem, using also the characteristics of the solution set of the SUM-MAX problem.
In Detti et al. [52] and Detti et al. [53], the authors proposed a pair of useful lower (upper) bounds LB_{SUM} (UB_{SUM}) and LB_{MAX} (UB_{MAX}) for the SUM and MAX objective, respectively. The lower bounds define an ideal point $IP = (LB_{SUM}, LB_{MAX})$ for the SUM-MAX problem, whereas the upper bounds are used to define a nadir point (UB_{SUM}, UB_{MAX}).
The solution space of the SUM-MAX problem has the following geometrical properties. Given a batch sequence σ_p, let $p = (SUM_p, MAX_p)$ be the corresponding point in the solution space of the SUM-MAX problem, where $SUM_p = NS(\sigma_p) + NC(\sigma_p)$ and $MAX_p = \max\{NS(\sigma_p), NC(\sigma_p)\}$. It can be noted that the following relations hold:

$$SUM_p \leq 2 * MAX_p \tag{6.3}$$

$$MAX_p + \eta \leq SUM_p \tag{6.4}$$

where $\eta = \min_\sigma\{NS(\sigma), NC(\sigma)\}$ is a value depending on the instance that can be easily computed. In fact, the minimum setup cost for a department can be easily attained ordering the batches in the sequence by colors or shapes.
Obviously, inequality (6.3) holds for all points p representing a feasible solution for the SUM-MAX problem. In fact, for each solution point, the corresponding value of the objective function SUM is less than twice of the value assumed by the second objective MAX. Relation (6.4) holds for all the feasible points of the SUM-MAX problem, too. Hence, inequalities (6.3), (6.4) and the lower and upper bounds previously referred describe a region containing the whole solution set of the SUM-MAX problem (see Figure 6.1for an example). It is important to note that while coordinates SUM_p and MAX_p depend on the particular σ_p, associated to the point p, η is a value which depends only on the problem's instance.
In the Figure 6.1, $N = (UB_{SUM}, UB_{MAX})$ is a nadir point, IP is the ideal point determined by LB_{SUM} and LB_{MAX}; while the two lines associated to inequalities (6.3) and (6.4) delimit the solution set, contained in the dark area of Figure 6.1.
These observations give us some useful information about the solution set of the NC-NS problem, object of this chapter. Given a batch sequence σ_p and its representative point $p = (SUM_p, MAX_p)$ in the solution space of SUM-MAX problem, the image of p in the solution space of NC-NS problem is given by the point $\pi = (NC(\sigma_p), NS(\sigma_p))$, where: $SUM_p = NS(\sigma_p) + NC(\sigma_p)$, and $MAX_p = \max\{NS(\sigma_p), NC(\sigma_p)\}$. Thus, several characteristics of one space can easily be mapped in the other one.
In Figure 6.2, the region containing the solution set of the NC-NS problem is depicted and, in Figure 6.3, the ideal Pareto front of this problem is shown. In particular, the case with $\min_\sigma\{NC(\sigma)\} \leq \min_\sigma\{NS(\sigma)\}$ (i.e., $\eta = NC(\sigma)\}$) is considered.
It can be observed that, the lower bound on the SUM objective, LB_{SUM}, is quite useful also in the solution space of the NC-NS problem giving an estimation of the Pareto optimal front. In fact, in this space the relation $SUM \geq LB_{SUM}$ can be re-viewed as $NS + NC \geq LB_{SUM}$ and hence it gives a lower bound for the Pareto optimal front of the problem. The dark area of Figure 6.2 contains the solution set of the NC-NS problem. In particular, every feasible point $\pi = (NC_\pi, NS_\pi)$ for the NC-NS problem belongs to a region bounded by the following five inequalities, namely

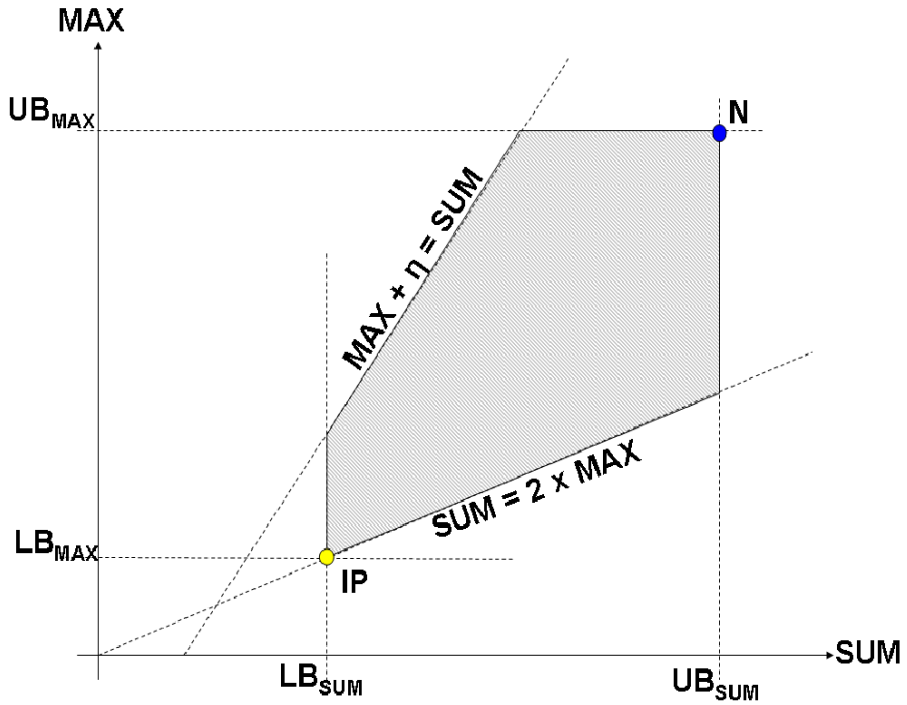

Figure 6.1: Geometric aspects of the feasible region of the SUM-MAX problem.

1. $NS_\pi \leq \max_\sigma\{NS(\sigma)\}$;
2. $NC_\pi \leq \max_\sigma\{NC(\sigma)\}$;
3. $NC_\pi + NS_p \geq LB_{SUM}$;
4. $NC_\pi \geq \min_\sigma\{NC(\sigma)\}$;
5. $NS_\pi \geq \min_\sigma\{NS(\sigma)\}$.

The nadir point

$$Nadir = (\max_\sigma\{NC(\sigma)\}, \max_\sigma\{NS(\sigma)\})$$

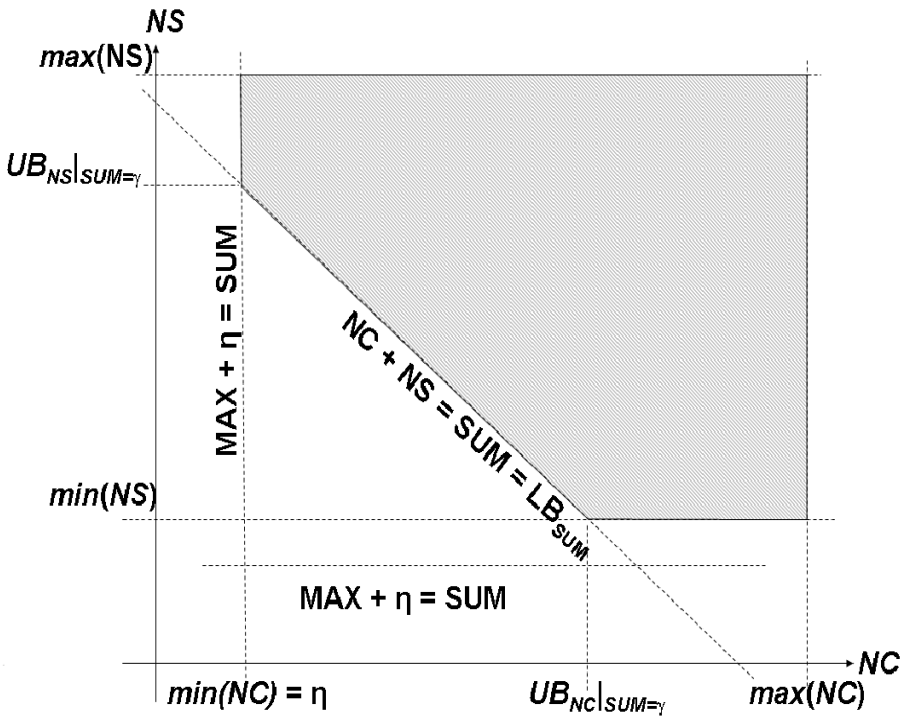

Figure 6.2: Geometric aspects of the solution space of the NC-NS problem.

and the ideal point

$$IP = (\min_{\sigma}\{NC(\sigma)\}, \min_{\sigma}\{NS(\sigma)\})$$

delimit the feasible solution set of the NC-NS problem. The estimation of the Pareto optimal front is given by integer points on a segment contained in the line $NC + NS = LB_{SUM}$.
More information about the solution set of the NC-NS problem can be obtained mapping other geometric objects from the SUM-MAX solution set, as described in the following and shown in Figure 6.2. Given an estimated Pareto front defined by $NS + NC = SUM = LB_{SUM}$, in the following expressions, for sake of simplicity, we indicate with γ the value of SUM. For an instance

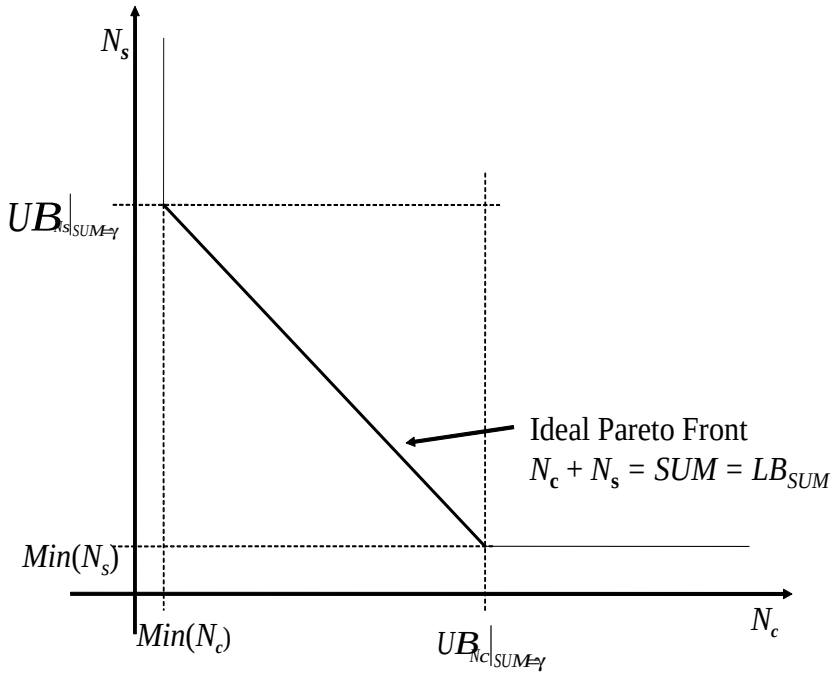

Figure 6.3: Bounds for the Pareto front for the NC-NS problem.

of the NC-NS problem, let $UB_{NS}|_{SUM=\gamma}$ and $UB_{NC}|_{SUM=\gamma}$ be the upper bounds on the values of NS and NC, respectively, when SUM is fixed to the value γ. We have the following relations delimiting the estimated Pareto front:

$$UB_{NS}|_{SUM=\gamma} = \gamma - \min_{\sigma}\{NC(\sigma)\} \tag{6.5}$$

$$UB_{NC}|_{SUM=\gamma} = \gamma - \min_{\sigma}\{NS(\sigma)\} \tag{6.6}$$

Moreover, using the information given by these geometric observations and the values of the bounds previously introduced, it is possible to calculate, in closed form, the cardinality $|PF_\gamma|$ of the set of all points laying on the estimated Pareto front PF_γ described by $NS + NC = SUM = \gamma$. In fact, due to the

geometric properties of the upper bounds (6.5) and (6.6) on the solution space, the number of integer points contained in the estimated Pareto front clearly is:

$$|PF_\gamma| = UB_{NC}|_{SUM=\gamma} - \min_\sigma\{NC(\sigma)\} + 1, \tag{6.7}$$

or, equivalently,

$$|PF_\gamma| = UB_{NS}|_{SUM=\gamma} - \min_\sigma\{NS(\sigma)\} + 1. \tag{6.8}$$

and the position of each point belonging to it can be easily computed. We will use this geometric characterization in the design and development of the solution algorithms. In fact, the number and the positions of the integer points on the estimated Pareto front will be used to guide the algorithms' behavior.

Solving single-objective problems with general setup costs

We next discuss the case in which setup costs of each department are different, and we show that the single-objective problem of minimizing the sum of the setup costs is closely related to the *NC-NS* problem. Let suppose that, the total costs incurred by departments D_C and D_S are expressed by increasing functions of the number of setups $c_C(NC)$ and $c_S(NS)$, respectively, and that the global cost to minimize is $c_S(NS) + c_C(NC)$.
Let $\bar{x}$ be a solution belonging to the Pareto optimal front of the *NC-NS* problem, and let x be a dominated solution. Clearly $c(x) \geq c(\bar{x})$ holds. Let $x^* = (NS^*, NC^*)$ be a solution on the Pareto optimal front for the *NC-NS* problem, with the minimum value of the global cost $c_S(NS) + c_C(NC)$. Then x^* is a solution minimizing the single-objective $c_S(NS) + c_C(NC)$. Hence, the following proposition holds.

Proposition 6.3.1 *The Pareto optimal front of the NC-NS problem contains a solution minimizing the objective* $c_S(NS) + c_C(NC)$.

In conclusion, solving the bi-objective *NC-NS* problem allows also to solve the class of single-objective problems in which $c_S(NS) + c_C(NC)$ must be minimized.

6.4 Algorithms

In this section we describe three metaheuristic algorithms developed to tackle the problem. A metaheuristic is an iterative solution procedure, combining

subordinate heuristic tools into a more sophisticated framework [72, 89]. All the developed algorithms are based on the same basic subordinate procedures, namely a constructive heuristic, some improving procedures and an update routine.
These subordinate procedures are used by some *Master procedure* to build the *solution front*, which contains all the non-dominated points generated by the algorithms. In the following subsections, first we describe in details the tools that constitute our framework, and next we introduce two Master procedures employing the described subordinate tools. We conclude this section illustrating the metaheuristic algorithms developed for the problem under study.

Constructive heuristic

The constructive heuristic is an iterative greedy procedure used to generate starting solutions, i.e., batch sequences. The heuristic starts from the empty sequence $\sigma = \emptyset$ and at each step it selects one attribute from a department. Then it adds to the partial sequence σ all the batches sharing the selected attribute. This process is repeated until all the batches are sequenced.
More in details, the behavior of the heuristic is guided by a parameter ρ, determining which department is selected at each iteration. In particular, if $\rho > 0$ ($\rho < 0$) then attributes from department D_S (department D_C) are chosen in the next $|\rho|$ iterations of the heuristic. At each of these $|\rho|$ iterations, the attribute (of the selected department) with the smaller number of occurrences among the unsequenced batches is chosen.
In terms of the bipartite graph $G = (S, C, B)$, the heuristic selects either the node subset S or C according to parameter ρ. At each of the $|\rho|$ iterations, the node of the selected node set having the smaller degree is chosen. Then, all edges incident to that node are removed from the graph, and consecutively added to the sequence under construction. In this way, the graph always represents the unsequenced batches.

Solution improving procedures

As solution improving tools we consider two different procedures which try to minimize the objectives SUM and MAX, respectively. The two single objective problems minimize the overall number of setups and balance of the setup among the two departments, respectively.
Observe that, minimizing the SUM objective, i.e., minimizing $NS + NC$, corresponds to a move toward the Pareto optimal front of the NC-NS problem.

In fact, given a solution $\pi = (NS(\sigma_p), NC(\sigma_p))$, the reduction of the SUM objective by one generates a new solution π' which is either $\pi' = (NS(\sigma_p) - 1, NC(\sigma_p))$ or $\pi' = (NS(\sigma_p), NC(\sigma_p) - 1)$. On the other hand, since the MAX objective represents the balance among the setups paid by the departments, the variation of the MAX objective, while maintaining constant the value of SUM, allows the exploration of new solutions having the same SUM quality. Note that, in general, the reduction of the MAX objective generates a new adjacent solution π' toward the center of the solution space NC-NS, while the increase of the MAX objective generates a new adjacent solution π' toward the extremity.
More in details, this is done by the following procedures:

- We use an Iterated Local Search (ILS) procedure [120] for minimizing the SUM objective and to obtain a solution belonging to the Pareto optimal front or close to it. This ILS algorithm has been introduced by Detti et al. [51], and given an initial solution it performs a Variable Neighborhood Descent (based on two complex neighborhoods) as local search step. The Perturbation phase applies random moves to the incumbent solution. The actual number of perturbing moves is randomly drawn in the $[1; 0.15|B|]$ interval. A solution is accepted as the new incumbent solution only if it strictly decreases the SUM objective. Each run of the ILS is stopped whenever the optimality of the solution is proved or when the computational bounds (3 seconds or 50 iterations) are reached. The parameters for the ILS algorithm are as in [51], whereas the lower bound used to prove the optimality and to stop the search process is described in details in [52].

- Once a new non-dominated solution is obtained, a diversification procedure optimizing the MAX objective, called COVER, is used. Given a solution π the COVER procedure attains all the non-dominated solutions that can be reached by a fast exploration of a simple neighborhood in the MAX objective [53] starting from π. In particular, given a starting solution $\pi = (NS(\sigma_p), NC(\sigma_p))$, the procedure basically consists in generating all the adjacent solutions that can be achieved starting from π either minimizing or maximizing the MAX objective. The COVER procedure is based on the observation that minimizing or maximizing the MAX objective produces a solution $\pi' = (NS(\sigma_p) - 1, NC(\sigma_p) + 1)$ or $\pi'' = (NS(\sigma_p) + 1, NC(\sigma_p) - 1)$ having the same SUM value and adjacent to π in the $NS - NC$ space. In other words, COVER applies a local search procedure starting from π minimizing and maxi-

mizing the MAX objective. Let $\pi_{\min} = (NS(\sigma_p) - i, NC(\sigma_p) + i)$ and $\pi_{\max} = (NS(\sigma_p) + j, NC(\sigma_p) - j)$ be the two local minima in the MAX neighborhood, obtained by applying i and j local search steps, respectively. Due to the particular neighborhood structure it is possible to generate all the solutions between $\pi_{\min}$ and $\pi_{\max}$. Once $\pi_{\min}$ and $\pi_{\max}$ are reached, a single iteration of the ILS is applied and, if a new solution on the solution front has been generated COVER is applied again to explore the solution front. This procedure is applied iteratively until no new solutions on the solution front are discovered. In Figure 6.4, a graphical representation of the behavior of the COVER procedure is reported.

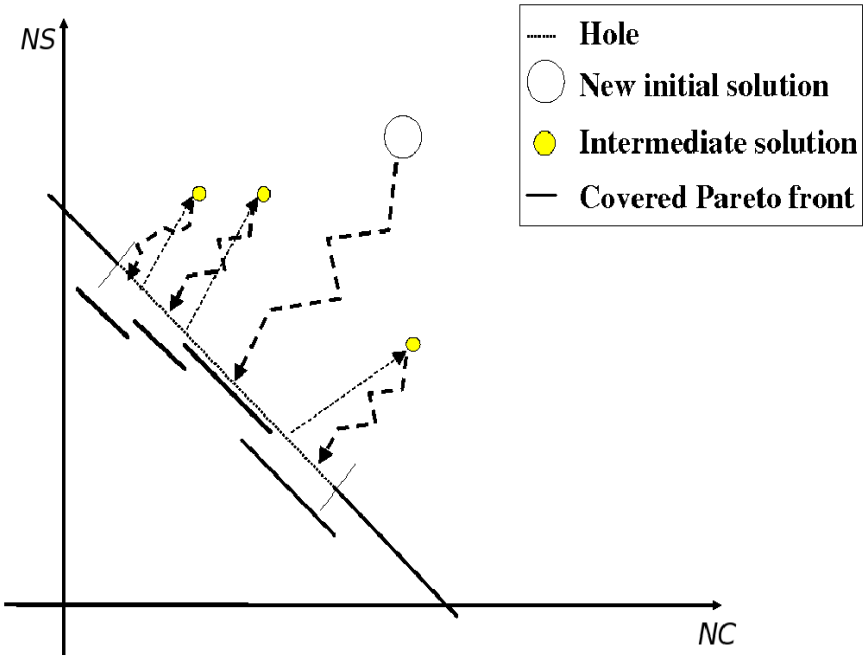

Figure 6.4: The COVER procedure.

Update routine

This function updates the solution front. Since COVER returns a set of non-dominated solutions that are reachable by the solution found by the ILS phase, the knowledge of the two extreme solutions is sufficient to calculate the exact amount of solution front covered in the iteration.

Master procedures

The subordinate procedures are used by *Master procedures* to guide the search process. Starting solutions are generated by the constructive greedy and they are improved by local search procedures previously described. In the following two Master procedures are introduced:

- The *Sweep* procedure (Figure 6.5) uses the constructive greedy to generate an *Equally Spaced Initial Solutions Set* (ESISS) which is used to control the search process. In particular, the ESISS contains a set of solutions (i.e., batch sequences) that are generated by the constructive heuristic, using different ρ values. Starting from each solution in ESISS the improving procedures (ILS and COVER) are applied in cascade aiming to reach a Pareto optimal point and consequently covering a portion of the Pareto optimal front. The number of solutions in the ESISS is a parameter for the algorithm. After a preliminary campaign of experiments we set ESISS as a function of the size and density of the input graph. This choice provides a way to take into account the different possible classes of inputs which are mainly characterized by the size and the density of the graph.

- The *Fill* procedure uses a different strategy. Starting from the knowledge of the coverage of the solution front, a *hole* is identified, i.e., a portion of the solution front not yet explored by the algorithm. Given a hole, Fill pilots the constructive heuristic to build an initial solution having the same balance of the hole (Figure 6.6). In this way it creates a new initial solution trying to drive the exploration to fill up the hole. The search process, in this case, is guided by the knowledge of the solution front found during the search process, and no additional parameters are required.

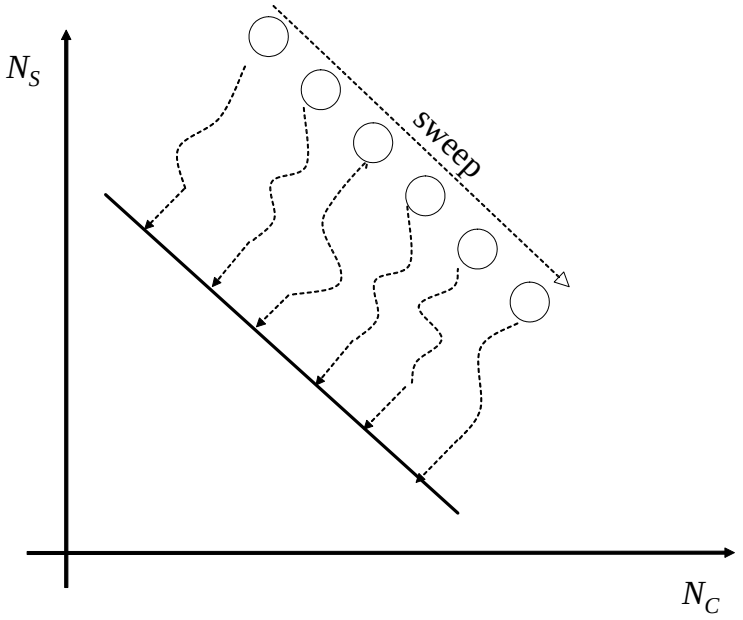

Figure 6.5: The Sweep procedure.

Metaheuristics

Using the two defined Master procedures we developed three different metaheuristic algorithms.

- The *Multi-Sweep* (MS) algorithm repeatedly applies the Sweep procedure. Once all the solutions of the ESISS are considered, it starts again from the same sequence of piloting starting solutions. This process is iterated until the time limit t_{max} is reached or the whole Pareto optimal front is covered.

- The *Sweep and Fill* (SF) algorithm uses a different strategy. Like Multi-

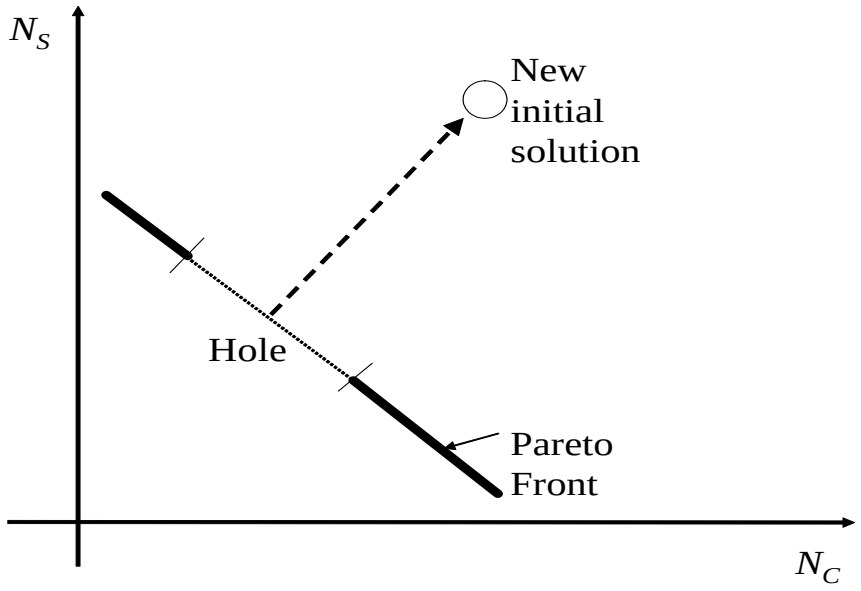

Figure 6.6: The Fill procedure.

Sweep it creates an Equally Spaced Initial Solution Set and it uses it as a pool of initial solutions. Once all the solutions in the ESISS are considered the algorithm applies the Fill procedure trying to fill the holes in the solution front until the time limit t_{max} is reached or the whole Pareto optimal front is covered.

- The *Front Fill* (FF) algorithm relies only on the Fill procedure. Clearly at the beginning no solution in the solution front has been yet identified, therefore it generates the initial solution corresponding to the central element of the solution front. The Fill procedure is repeated until the whole Pareto optimal front is identified or the time limit t_{max} is reached.

6.5 Computational experiments

In this section we describe the experiments carried out to evaluate the behavior of the three proposed algorithms.
The algorithms have been tested on one set of 32 real-life instances (GSET) and on 460 randomly generated problems ([4]). The 32 real-life instances consist of unbalanced bipartite graphs $G = (V_1, V_2, E)$ where $|V_1| = 32$ and $|V_2| = 14$. In these instances $|E|$ ranges from 150 to about 300. The other sets of randomly generated instances consist of connected balanced bipartite graphs $G = (V_1, V_2, E)$, with cardinality $n = |V_1| = |V_2|$ and graph density $d \in \{5\%, 10\%, 20\%, 30\%, 40\%\}$. In particular, five sets with varying n have been considered ($n \in \{10, 30, 60, 80, 100\}$), and each set is divided into subsets having different densities d. For each subset, 20 connected instances have been randomly generated.
In a preliminary test phase we tuned the proposed algorithms. On the basis of these tests, the cardinality of the ESISS (i.e., the number of initial solution considered in the Sweep routine) for MS and SF is set to $((|PF| * d)/2) + 1$, where $|PF|$ is the cardinality of the estimated Pareto front calculated from Equations 6.7 and 6.8. The knowledge of the cardinality of the ESISS allows to easily calculate the parameter ρ controlling the constructive heuristic. The algorithms stop when one of the following conditions holds:(i) the computation time exceeds t_{max} (in our tests t_{max} is set to 60 seconds); (ii) the whole Pareto optimal front is found. Since the algorithms are stochastic, for each instance 10 runs of each algorithm are executed, in order to obtain a meaningful representation of their behavior. All the experiments have been performed on a 3.2 GHz Pentium 4 laptop equipped with 512 MB RAM, and the algorithms are coded in standard C language.
In Table 6.1, we summarize the results for each algorithm on each test set. Columns 2 and 3 show the density d and the average cardinality of the estimated Pareto front ($|PF|$), respectively. The remaining columns show, for each algorithm, the computation time in seconds (time), the number of initial solutions generated (Iter.) and the Pareto optimal front covering percentage (%). Each row refers to the average results over 20 instances and 10 repetitions with the computation time limit set to 60 seconds.

Table 6.1: Results of the three algorithms (60 seconds runs).

Instance set	d	$\|PF\|$	MS			SF			FF		
			time	Iter.	%	time	Iter.	%	time	Iter.	%
GSET	-	211.313	0.281	217.066	100.000	4.896	1198.244	99.975	5.194	1357.291	99.975
RND10B	0.1	2.750	15.101	45959.010	86.667	12.923	37904.325	88.167	13.235	41272.645	88.167
RND10C	0.2	6.150	3.285	9623.530	95.000	3.196	8455.960	95.000	3.370	9983.985	95.000
RND10D	0.3	10.700	0.433	1317.100	100.000	0.996	2545.910	99.917	1.065	2797.770	99.833
RND10E	0.4	18.100	0.003	7.450	100.000	0.003	6.695	100.000	0.002	7.000	100.000
RND30B	0.1	39.000	4.252	1738.605	99.826	6.502	2156.575	99.707	6.294	2030.000	99.720
RND30C	0.2	118.100	0.067	45.215	100.000	0.112	52.870	100.000	0.134	87.030	100.000
RND30D	0.3	207.400	0.206	143.000	100.000	0.126	82.270	100.000	0.103	73.245	100.000
RND30E	0.4	296.950	0.331	199.775	100.000	0.237	144.955	100.000	0.192	124.000	100.000
RND60A	0.05	78.450	42.899	3718.530	97.314	47.502	4047.035	94.422	48.773	4221.920	93.292
RND60B	0.1	241.550	4.449	180.400	99.991	13.052	403.560	99.929	11.790	370.105	99.936
RND60C	0.2	594.750	22.372	583.585	99.995	15.245	385.870	99.998	12.884	337.165	100.000
RND60D	0.3	946.350	45.365	1009.990	99.957	35.705	775.660	99.976	34.883	768.360	99.982
RND60E	0.4	1299.050	54.885	932.375	99.927	53.584	839.340	99.930	53.337	847.165	99.856
RND80A	0.05	165.100	19.352	542.295	98.073	30.961	813.045	97.792	32.844	869.750	98.446
RND80B	0.1	482.400	15.965	167.030	99.996	16.922	152.815	99.945	20.029	181.830	99.950
RND80C	0.2	1113.600	59.714	284.185	99.856	58.760	280.940	99.869	56.640	293.635	99.886
RND80D	0.3	1741.050	60.337	209.490	99.319	60.410	204.845	99.575	60.105	197.375	99.310
RND80E	0.4	2370.150	60.556	145.295	98.707	60.662	148.735	98.867	60.561	134.715	95.815
RND100A	0.05	300.250	31.623	399.460	99.553	49.368	652.945	99.191	52.641	713.340	99.005
RND100B	0.1	801.500	55.063	173.745	99.886	50.563	165.185	99.892	47.331	152.895	99.912
RND100C	0.2	1791.600	60.637	80.450	50.091	60.599	85.465	51.194	60.589	74.750	96.934
RND100D	0.3	2782.800	61.050	57.590	43.930	61.024	60.650	44.701	61.087	48.455	82.714
RND100E	0.4	3764.200	61.200	61.440	41.841	61.204	59.280	40.186	61.111	46.412	77.005

Table 6.2: Results of the three algorithms (300 seconds runs).

Instance set	d	$\|PF\|$	MS			SF			FF		
			time	Iter.	%	time	Iter.	%	time	Iter.	%
RND100A	0.05	300.250	69.832	600.425	99.950	233.523	2597.770	99.603	242.974	3131.685	99.599
RND100B	0.1	801.500	115.034	415.885	99.993	76.933	270.765	100.000	75.202	275.725	100.000
RND100C	0.2	1791.600	299.348	423.855	99.808	291.676	431.520	99.834	284.430	430.495	99.802
RND100D	0.3	2782.800	301.026	302.110	97.609	300.131	288.775	97.949	301.081	272.855	95.843
RND100E	0.4	3764.200	301.067	328.250	98.507	300.749	317.000	97.654	301.219	297.065	93.815

It can be observed that, on average a large part of the Pareto optimal front is covered by all algorithms (%) in a small computation time. In real-life instances (GSET), which have a relatively low number of nodes and high density, all the proposed algorithms achieve a covering percentage close to 100% ($\geq 99\%$). Regarding the balanced and small sets (RND10), the average percentage of covering is quite high ($\geq 85\%$), even if a very high number of starting solutions is generated. After a closer examination of the results on small and sparse instances, it can be observed that the number of iterations to attain all the Pareto optimal front is, in general, small; but there are some exceptions, while the Pareto coverage is smaller than 100%. In fact, the estimated Pareto front calculated as described in Section 6.3 may even be unreachable, since LB_{SUM} is only a lower bound on the optimal value.
All the algorithms are able to attain high coverage of the Pareto optimal front (which results often $\geq 98\%$) for all the considered tests. For the largest instances we observe a substantial performance difference when comparing MS and SF algorithms with FF. In fact, the coverage of the Pareto optimal front on instances RND100C, RND100D and RND100E is quite high for FF (about 80% on average), while it is low for MS and SF (about 45% on average). Therefore, a second experimental campaign has been carried out on the largest instances (RND100) to evaluate the behaviour of the algorithms when more computation time is allowed. In particular, in this second campaign we tackled only the bipartite graphs with 100 nodes and we set to 300 seconds the computational time limit. The results are shown in Table 6.2. From this second campaign we observe that, when more computation time is allowed, all the algorithms are able to cover more than 90% of the front, and MS and FF algorithms are able to explore more than 95%.
We also compared the three algorithms on the basis of a statistical analysis. In particular, we performed two non-parametric tests, namely *Friedman T Test* and *the Kendall's W Test*, which are able to compare multiple tuples of related data and they neither require the distributions of such data are normal or symmetric. In all cases (with $\alpha = 0.05$), both tests lead to the same conclusions. In the following, only results related to the *Friedman T Test* are reported. The null hypothesis is that an algorithm significantly performs better than others across the test instances. We consider both the real-life and the randomly generated instances, six sets in total (recall that each random set contains instances with the same number of nodes and different density values).
In Table 6.3, the results related to algorithms' computational times and coverage are reported. Since the distribution of scores is expected to be neither normal nor symmetric, we use the median execution time and Pareto opti-

mal front coverage although considering average executing times and Pareto optimal front coverages leads to the same conclusions. In particular, Table 6.3 reports only on the instance sets for which significant statistical differences arise. For each instance, the algorithms are ranked and the ranks are then summed over all instances in the same set, i.e., with the same number of nodes. Thus, in Table 6.3 (a) a lower rank sum indicates that an algorithm tends to be faster; whereas in Table 6.3 (b) a higher rank sum indicates that an algorithm tends to achieve better Pareto optimal front coverage. Concerning the execution time of each metaheuristic (Table 6.3 (a)), we observe a statistical difference only when comparing the three algorithms on small instance sets (GSET, RND10–RND60), in which FF is faster than the other algorithms. No statistical differences arise on larger instances. This is not surprising because, on larger instances, the algorithms tend to use all the allowed computational time. On the other hand, for what concerns the percentage of Pareto optimal front covering (Table 6.3 (b)), significant statistical difference arises only on larger instances (set RND100) when considering both one and five minutes runs. On short runs, with the time limit set to 60 seconds, there is a statistical difference according to the Friedman test, and it can be noted that FF performs better (i.e., it exhibits higher sum rank than the others algorithms). On the other hand, when more computation time is allowed, a statistical difference arises again but, in this case, the best algorithm turns out to be MS. Such behaviors can be explained observing that FF generates as starting solution only promising candidates and, hence, it quickly covers the Pareto optimal front at the beginning. However, FF algorithm is not able to easily cover all the Pareto optimal front. On the other hand, MS uses sequentially a huge number of starting points in ESISS, and thus resulting in a better coverage of the Pareto optimal front.

Table 6.3: Rank sum of (a) median computation times and (b) Pareto optimal front coverage.

Instance set	MS	SF	FF
GSET	76	60	56
RND10	144	170	166
RND30	179	159	142
RND60	226	194	180

Instance set	MS	SF	FF
RND100 (60 sec.)	175	170	255
RND100 (300 sec.)	223	211	166

From the above discussion, it follows that, although all algorithms perform in an excellent way with results very similar in time and covering, FF has the

best performances when a small computation time is considered, while MS is slower, but more accurate and effective in covering every zone of the front.

6.6 Conclusions and future research

In this chapter a two objective setup coordination problem arising in a two stage serial manufacturing system is addressed. The problem consists in finding a common sequence of batches to be produced such that the number of setup paid on each department is minimized. In particular the case in which all the setups are identical has been considered. For this problem a geometrical characterization of the Pareto optimal front is given and it is used to develop effective algorithms. Three heuristic approaches have been proposed and they have been extensively tested on a wide set of instances. The proposed algorithms are able to cover large portions of the Pareto optimal front on average, within a reasonable computation time.
An extension of the proposed approach to a class of single-objective problems is also discussed. Future research directions include:
i) Improve the lower bound procedure and to try to establish a priori that some points of the estimated Pareto front do not correspond to feasible solutions. Such improvements could guarantee better performance of the algorithms both in terms of quality and computation time.
ii) From the decision maker point of view it could be also interesting to find different sequences corresponding to each Pareto point. In fact, our approach finds only a solution for each Pareto point.
iii) Adapt and test the algorithmic approach to deal with the case of different setup costs in each department.

Chapter 7

Scheduling dispensing and counting in secondary pharmaceutical manufacturing

The last case study presented in this thesis is a problem of operations scheduling in dispensing and counting departments of pharmaceutical manufacturing plants. The departments are modelled as a multi-objective parallel machines scheduling problem under a number of both standard and realistic constraints, such as release times, due dates and deadlines, particular sequence-dependent setup times, machine unavailabilities, and maximum campaign size. Main characteristics of the metaheuristic methodology, presented to solve this problem, are the modularity of the solution algorithms, the adaptability to different objectives and constraints to fulfill production requirements, the easiness of implementation, and the ability of incorporating human experience in the scheduling algorithms. Computational experience carried out on two case studies from a real pharmaceutical plant shows the effectiveness of this approach.

7.1 Introduction

The pharmaceutical production systems are called to improve their production planning and scheduling methods to strive for better utilization of resources and reduction of the response time. In fact, declining windows of product ex-

clusivity, competition from generics, and new market entrants from third world countries are increasing the competitive pressure on pharmaceutical companies [151], [186]. Pharmaceutical producers need to increase flexibility, and, at the same time, to obtain a noticeable reduction of production costs preserving the quality of products and processes.
Maintaining high growth rates requires to increase the number of new products or the internal efficiency of the organizations. In fact, there are great margins for costs reduction in facing time-to-market opportunity costs, research and development inefficiencies, under-developed processes for matching supply with demand, poor manufacturing cycle efficiency and discontinuous workflows [151], [178].
In this chapter we focus on one of such issues, namely the optimization methods for production scheduling. Common features of the production processes are the small size of the lots, strict requirements on product quality and timing delivery, and the large variety of constraints arising from the shop-floor characteristics or from various technological issues. Moreover, the quality of a schedule may involve several indices, such as the use of shop-floor resources, production costs in terms of personnel and energy consumption, the attainment of production targets, and so on.
Due to the inherent complexity and variety of pharmaceutical scheduling problems, a large extent of scheduling and related issues are still carried out by human schedulers, who are able to develop feasible schedules based on their past experience and intuition.
Developing and implementing effective computerized systems for addressing such operational problems require, therefore, focusing on a number of aspects that are rarely taken into account jointly in the scheduling theory. Among the others, we recall the need for general solution algorithms, able to deal with different objectives and constraints to fulfill production requirements, and automated scheduling systems able to easily embed observations and suggestions arising from the scheduling practice.
In pharmaceutical manufacturing, the production strongly depends on the customer demands in terms of products, quantities and delivery times. In this context, the automation of scheduling activities in non-critical areas allows the management focalizing on critical areas then propagating solutions and constraints to the other areas of the plant. Thus it allows to enlightening promptly possible infeasibilities under on overall view of the plant, since such areas often serve different departments or lines. This contributes to a plantwide increase of scheduling responsiveness.
In this chapter we move in this direction and focus on a general methodology to

address scheduling problems arising in two areas of pharmaceutical production systems. In particular, we apply the method for scheduling the production in a parallel machine environment, including lot production with setups, due dates, deadlines and other realistic constraints. Moreover, we report on two case studies detailing the implementation of the proposed methodology in a real industrial context. The rationale is developing algorithms easily adaptable to deal with different constraints and objectives, and suitable for incorporating human experience in the computerized logic. Such methods are not necessarily based on strong mathematical properties of the particular problem to solve. Rather, the search process is guided by several heuristic procedures, called pilot heuristics, so that it is possible to include new procedures suggested by human experience and/or to modify the behavior of the system by adding, removing or modifying the pilot heuristics. At this aim we apply a metaheuristic strategy known as *rollout* method [25] or *pilot* method [56]. These methodologies for the approximate solution of discrete optimization problems have been independently developed by Bertsekas *et al.* [25] and Duin and Voß [56]. The main idea of a rollout algorithm is to include one or more pilot heuristics in a larger framework with the purpose of improving their performance. We also considered general local search techniques for improving the quality of the schedules, and we focused in particular on variable neighborhood descent techniques [80, 81].
The chapter is organized as follows. Section 7.2 introduces the main issues of pharmaceutical manufacturing system. In section 7.3 the case studies are illustrated in detail, and in section 7.4 the proposed algorithmic approaches are described, while the relative results are reported and discussed in the successive section 7.5. Finally, conclusions are drawn in Section 7.6.

7.2 Pharmaceutical manufacturing systems

The pharmaceutical industrial production mainly consists of at least two manufacturing stages: *primary* and *secondary* manufacturing [178, 39]. The former is dedicated to the production of active ingredients and other basic components and production is typically a *push* process (driven by forecast demand) organized in long campaigns to reduce the impact of long cleaning and setup times that are necessary to ensure quality and to avoid cross-contamination. Primary manufacturing is therefore not very sensitive to short-term fluctuations of the customers demand, and the main issue is a careful lot sizing to avoid shortages of active ingredients in successive production steps [186] [178]. Secondary man-

ufacturing production is usually a *pull* process, driven by wholesalers' orders, in which active ingredients and other components are dispensed, processed and packed to compose the final products.
In this chapter, we focus on secondary manufacturing systems only, which consist of a set of multi-purpose production facilities that produce a variety of intermediate and finished products through multi-stage production processes. Facilities are linked by supplier-customer relations, and each of them may interact with other external (e.g. suppliers) or internal (e.g. warehouses) entities. The production processes are commonly devoted to produce solid, liquid, aerosol or powdered items according to a family of similar recipes [39]. They are typically organized with three or four main departments (Figure 7.1). These departments, to a certain extent, can operate independently from each other because intermediate products can be stored in sealed bins.
The preliminary activities of materials preparation are performed in the *dispensing* department [39]. These operations attain to weighing, dosage and preparation of each ingredient in a recipe and represent one of the most important steps in the pharmaceutical process. All recipes must be strictly observed and only approved materials can be used in the operation.
Weighing materials is a time consuming operation requiring workers to pay particular attention for: containment and separation of product from operators; possible cross-contamination; cleaning of booth and equipment and gowning and washing for the operators and separation from the other areas. The main function of the dispensing department is to test materials and release them to production lines. The essential requirement is that material released has been tested and approved by quality controllers for production purposes, and that only this material is used for manufacturing products. The dispensary handles several different classes of material some of which require particular care in the preparation due to their active or dangerous properties. In addition, material may also have to be passed back to the warehouse when only a partial quantity is required from a large container. Finally, the dispensing department provides records that enable a complete audit trail of all the materials dispensed.
The specific production process activities, such as binder preparation, bulk materials granulation and blending, tablet or capsule production, are performed in the *manufacturing* department, while activities of counting and packaging are in charge of the *packaging* department [211]. However, when counting activities are relevant, *counting* and packaging activities are performed in different departments. This situation frequently arises in Europe due to the many different national specific rules and languages requiring to handle a huge number of different packages in the same plant [151],[178], [39].

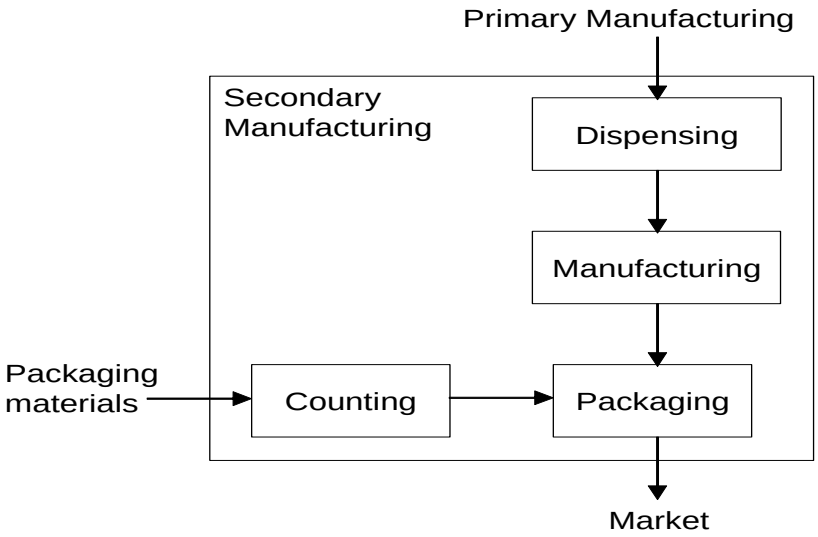

Figure 7.1: The main production phases in the secondary pharmaceutical manufacturing.

7.3 Problem description

In this chapter, we report our experience with a practical implementation of a scheduling system at a pharmaceutical production plant located in Italy. We consider the production scheduling of two departments: *dispensing* and *counting*. They are the simplest departments in secondary pharmaceutical manufacturing. The plant supplies different European countries and the production flow is organized with the four main phases of Figure 7.1. However, while there is only one dispensing and one counting department in the plant, manufacturing and packaging activities are organized with several departments. Late and urgent orders are managed as orders with strict deadlines, to be processed as soon as possible. Clearly, deadlines make difficult to organize long campaigns, and therefore tend to reduce the actual capacity of the departments. This reduction may cause, in turn, late deliveries at the end of the week and such

negative effects may propagate over several weeks. All these problems motivate the need for more coordination among the departments and with the planner.

Dispensing department

In the dispensing department the availability of all raw materials required by each recipe is checked. Raw materials, picked up from a warehouse, are weighed and prepared in sealed bins which are sent to buffers waiting for processing in the manufacturing department. The remainders, if any, are sent back to the warehouse unless the following job requires the same components. The weighing operations are performed in two independent rooms in parallel.
Cross-contamination issues require one product at a time being processed in a room and the room cleaned when switching from one product to another. *Minor cleaning* is sufficient when two consecutive products need the same raw materials. The production is organized in groups of products requiring the same raw materials to be scheduled consecutively, which are called *campaigns*. *Major cleaning* is however necessary after a maximum number of products of the same type, called the *size* of a campaign.
From the scheduling point of view each room acts as a single machine with sequence-dependent setup times and campaigns. For each production order there may be a release time and a due date or a deadline, when there is a risk of stock-out for customers. There may be planned temporary room unavailability, mainly due to lack of personnel, which must be taken into account when scheduling the production. Note that, while a setup operation can start before an unavailability and complete after the interruption, ordinary processing operations cannot be interrupted.
A schedule is considered feasible by the dispatchers when all the deadlines, if any, are respected. Hence, to attain feasibility, one must respect all the deadline constraints. Once the feasibility is reached we use as primary objective the minimization of the maximum lateness, i.e., the maximum over all products of the difference between the product completion time and its due date. As secondary objectives, in lexicographic order, we consider the makespan minimization and the minimization of the number of late jobs.

Counting department

The counting department prepares materials required for packaging. Packages, labels and information leaflets are taken from a warehouse, counted and prepared for the subsequent packaging operations. This department typically

deals with a much larger number of lots with respect to the dispensing department, due to the number of different packages that must be used in different countries and the presence of different packaging departments of the plant.
The counting department is composed of three independent rooms, although a room may be temporarily unavailable, and there is no significant setup between two consecutive operations. Also in the counting department there are release times (when the packages becomes available), deadlines and due dates (propagated backward from the packaging departments). As in the dispensing department the main goal is the respect of deadlines and the objectives are in lexicographic order minimizing maximum lateness, makespan and number of late jobs.

Scheduling models

The scheduling environment of interest in both departments is the parallel machines case, i.e., the problem is to decide when to start an operation and on which machine under a number of realistic constraints. In this context, industrial examples can be found in [142], while a survey on the parallel machine scheduling research is available in [36], while the parallel machine scheduling problem with multiple objectives is discussed by [197]. The version of the problem with sequence dependent setup times is addressed by [110] and [195] which considers also the case with minor and major setups. The latter case is particularly frequent in the pharmaceutical industry in which minor setups are related to successive processing of jobs belonging to the same family, while major setups occur when the families of two successive jobs are different and a more accurate and time/work consuming changeover operation is required.
A formal model for these scheduling problems can be formulated adopting the three fields $(\alpha/\beta/\gamma)$ classification scheme of Graham *et al.* [75], where α indicates the scheduling environment, β describes the job characteristics or restrictive requirements, and γ defines the objective function to be minimized. With this notation, the dispensing department can be classified as

$$P_2|r_i, d_i, D_i, s_{ij}, MCS, unavail|Lex(L_{max}, C_{max}, U),$$

in which, P_2 indicates identical parallel machines production environment with 2 machines; r_i, d_i, and D_i indicate that jobs have release times, due dates, and deadlines, respectively; s_{ij} indicates the presence of sequence-dependent setup times; MCS represents the constraints on the maximum campaign size; *unavail* represents the possible room unavailability constraint; the objectives

are (in lexicographic order): L_{max} minimization of maximum lateness, C_{max}, makespan minimization, U, minimization of the number of tardy jobs.
Similarly, the counting department can be described as:

$$P_3|r_i, d_i, D_i, unavail|Lex(L_{max}, C_{max}, U).$$

We notice that, although the dispensing and counting departments are considered simple departments from the operations practice point of view the resulting scheduling problems are considered quite difficult NP-hard problems in the scheduling literature [65],[152]. Moreover, some constraints as the maximum size of a campaign or the particular machine unavailability that allows to resume only cleaning operations are not frequently addressed in the scheduling literature [186].

7.4 Solution methods

In this section we describe a general approach to design scheduling algorithms that are easily adaptable, modular and suitable for incorporating human experience in the computerized methods. Simple heuristics are more easily accepted and trusted by the human schedulers, who can understand their principles and suggest modifications to improve performance over time. In these methods, the search process is performed by several heuristic procedures guided by a general optimization strategy. The method is applied to the cases under study, and the following subsections are devoted to describe in details the characteristics of the considered algorithms.

Constructive algorithms for dispensing and counting

We focus on greedy algorithms to produce a solution. Greedy algorithms typically sort the operations according to a given *criterion* and then, starting from the empty solution, build a complete schedule by adding to the partial schedule one operation at a time, according to the order induced by the adopted criterion.
We introduce two greedy algorithms developed for scheduling the production in the dispensing and counting departments. We developed a modified version of the Jackson Schedule [92] algorithm and we refer to it as *MJS*. Details on the MJS algorithm are illustrated in Figure 7.2. The heuristic criterion sorts all the jobs that are available to be scheduled first according to their priority (i.e.,

the presence of a deadline) and second by the smallest due date value. The dispatching rule assigns the selected job to the machine which is able to complete it first. Clearly, the completion time of a job takes in account the presence of sequence-dependent setup times, campaigns and machine availabilities. We observed that, the schedules produced by hand in the counting department were quite similar to that of the MJS algorithm.

Modified Jackson Schedule (MJS)
Input: a set P of production orders;
$t = \min\{r_i : i \in P\}$, $S = \emptyset$
repeat
  $R = \{i \in P : r_i \leq t\}$
  if there is a job in R with a deadline **then**
    select a job $j \in R$ with the smallest deadline
    in case of tie, select a job with the smallest setup
  else **if** there is a job in R with a due date **then**
    select a job $j \in R$ with the smallest due date
    in case of tie, select a job with the smallest setup
  else $t = \min\{r_i : i \in P\}$, $R = \{i \in P : r_i \leq t\}$
  if a job has been selected **then**
    assign j to the machine able to complete it first, taking into account (if any)
    setups, campaigns and machine availability, $S = S \cup \{j\}$, $P = P \setminus \{j\}$
    update t to the smallest completion time of all available machines
until $P = \emptyset$

Figure 7.2: Algorithmic scheme of the Modified Jackson Schedule.

Algorithm *Delta* extends the MJS algorithm by dividing processing horizon into intervals of length Δ, and by replacing the due dates (and deadlines) in the interval $[k\Delta, (k+1)\Delta]$ with $k\Delta$. Then, it schedules jobs according to the MJS algorithm. This means that in the preprocessing step the production orders having similar due dates are grouped together. Hence, a larger value of Δ favors the formation of larger campaigns, whereas setting $\Delta = 0$ corresponds to the MJS algorithm. In Figure 7.3 we show the details of the algorithm Delta. Algorithm Delta surrogates the schedules produced by hand in the dispensing department. In fact, in the dispensing department the schedulers strive to obtain large campaigns other than respecting the due dates.
The results of MJS and Delta heuristics were considered feasible, although not particularly performing, by the schedulers. We used the rollout method

Algorithm Delta (Δ)
Input: a set P of production orders;
Forall jobs $i \in P$ compute modified due dates and deadlines: $\bar{d}_i = \lfloor \frac{d_i}{\Delta} \rfloor \Delta$; $\bar{D}_i = \lfloor \frac{D_i}{\Delta} \rfloor \Delta$
$t = \min\{r_i : i \in P\}$, $S = \emptyset$
repeat
  $R = \{i \in P : r_i \leq t\}$
  if there is a job in R with a deadline then
    select a job $j \in R$ with the smallest modified deadline,
    in case of tie, select a job with the smallest setup
  else **if** there is a job in R with a due date then
    select a job $j \in R$ with the smallest modified due date,
    in case of tie, select a job with smallest the setup
  else $t = \min\{r_i : i \in P\}$, $R = \{i \in P : r_i \leq t\}$
  if a job has been selected then
    assign j to the machine able to complete it first, taking into account setups,
    campaigns and machine availability, $S = S \cup \{j\}$, $P = P \setminus \{j\}$
    update t to the smallest completion time of all available machines
until $P = \emptyset$

Figure 7.3: The scheme of the Algorithm Delta (Δ).

of Figure 7.4 to improve the performance of these basic heuristics. It is well known that greedy heuristics may exhibit an erratic behavior, possibly because (*i*) locally promising decisions may not lead to good global solutions, and (*ii*) wrong choices can not be changed anymore in the solution process. The main idea of the *Rollout* algorithm [25].[24] or *Pilot* method [56],[212] is to overcome or, at least, mitigate these limitations by means of a look-ahead strategy. More specifically, the partial solution is iteratively enlarged by using one or more greedy algorithms as a look-ahead strategy. This approach can be viewed also as an approximated dynamic programming method [23],[37]. These algorithms have been applied to different problems, such as stochastic vehicle routing problems [176], test sequencing for fault diagnosis applications [205], general versions of job shop scheduling problems [129].
We next illustrate the rollout/pilot method for a parallel machine scheduling problem. A solution S is a schedule of m machines and n jobs. Starting from the empty solution (with no fixed components), the k-th iteration consists in evaluating a *scoring function* $p(j)$ for each job j which can be added to the

```
Algorithm Pilot/Rollout
begin
for k = 1, ..., |J|  do
    begin
   p(j_0) = +∞
    for all j ∈ J  do
        if (p(j) = H(S_{k-1}, j)) < p(j_0)  then j_0 = j
   S_k = {S_{k-1} ∪ j_0}, J = J \ {j_0}
    end
end
```

Figure 7.4: Algorithmic scheme of Pilot/Rollout

current partial solution S_{k-1} and choosing the one having the smallest scoring function. The iteration is repeated, for $k = 1$ to n, when a complete solution is found. Given a partial solution S_{k-1}, and let j be a possible job to be added to the schedule, a pilot heuristic $H(\cdot)$ is a constructive algorithm that, starting from the partial solution S_k in which job j has been added according to the heuristic dispatching rule, produces a complete solution with objective function value $p(j) = H(S_{k-1}, j)$. At the end of the k-th iteration the job j_0 associated to the smallest $H(S_{k-1}, j)$ is permanently added to the partial solution. Figure 7.4 shows a sketch of the rollout algorithm using a single pilot heuristic applied to the problem of sequencing a set J of jobs and m machines.

Local search procedures

Another well known technique to improve a given solution is based on the *local search* principle. Local search algorithms are based on a neighborhood structure N. In particular, we considered three neighborhood structures associated with the following moves, aiming at improving the single machine schedules independently from each other.

- SWAP: Two adjacent jobs assigned on the same machine M_x, say j_k and j_{k+1}, are swapped. The rest of the schedule remains unchanged.
- INSERT: A job j_h is removed from its current position in the schedule of M_x and it is inserted on the same machine, before or after another operation j_k.

- EXCHANGE: Two (non-adjacent) jobs assigned on the same machine M_x, say j_h and j_k, are swapped, i.e., j_h replaces j_k and vice versa. The swap move is therefore a particular exchange move.

Figure 7.5 provides an illustration of the proposed moves.

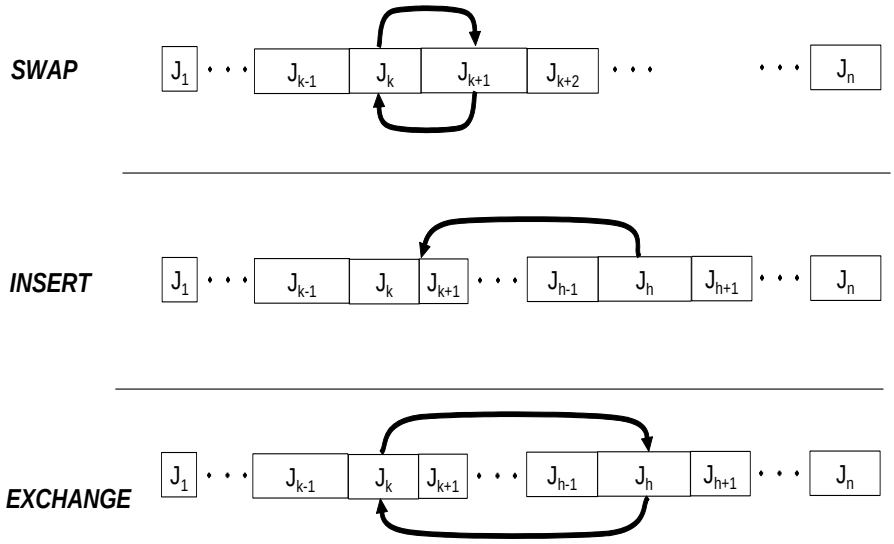

Figure 7.5: An illustration of the three moves.

Local search algorithms converge to a local minimum, which may be of poor quality. To contrast this drawback, a variety of strategies have been proposed to escape from local minima. The *Variable Neighborhood Descent* (VND) is a local search technique focused on systematic neighborhood changes [80],[81]. It is based on the observation that a local minimum for a given neighborhood is not necessarily a local minimum when other neighborhoods are considered. More specifically, let $\{N_1 \dots N_k\}$ be a set of neighborhood structures. The VND starts the search process from an initial solution in the first neighborhood $i = 1$, i.e., N_1. In the generic iteration, the algorithm scans the i-neighborhood looking for an improving move; once an improving move is detected it is per-

formed and the VND starts searching again in the first neighborhood (N_1). If no such improving move exists, the algorithm starts searching in the $(i+1)$-neighborhood. The search process terminates when no improving moves are available in all the considered neighborhoods. The sketch of the algorithm is given in Figure 7.6.
In our implementation we ordered the three neighborhoods are ordered according to their size, i.e., SWAP, INSERT and EXCHANGE.

Variable Neighborhood Descent (VND)

Input: a set of neighborhoods: $\{N_1 \dots N_k\}$, and an initial solution;
$i := 1$
repeat
  search for a profitable move in N_i
  if an improving move is found then
    apply the move
    $i := 1$
  else $i := i + 1$
until $i \geq k$

Figure 7.6: Algorithmic scheme of Variable Neighborhood Descent (VND).

7.5 Computational results

In what follows we study four different algorithms obtained by combining the three building blocks previously described:

- Algorithm H consists in the application of a simple greedy heuristic, either MJS or Delta.
- Algorithm RH is the rollout algorithm using H as pilot heuristic.
- Algorithm H+LS corresponds to improving the solution of the greedy algorithm by applying the VND procedure.
- Finally, we call RH+LS the rollout algorithm in which the final solution is used as initial solution by the VND algorithm.

These algorithms have been tested on two sets of 15 randomly generated realistic instances for each department, each instance representing two weeks of planned production. The instances have been generated respecting a typical production mix of the plant. Each test instance in the dispensing department has a number of jobs ranging from 40 to 60. The number of jobs in the dispensing department, for each instance, ranges from 400 to 480. These sizes represent the typical number of jobs produced in the two departments in a two-week production horizon. In the first set there are some jobs with higher priority, i.e., having a deadline, whereas in the second set there are no urgent jobs. In the instances belonging to the first set (i.e., having a deadline) 20% of the jobs has a deadline instead of a due date. The time in which a given job must be completed, i.e., the difference between due date (deadline) and release time $d_i - r_i$, ranges from one to three production time shifts (each production time shift is set to 16 consecutive hours). Besides a few random machine unavailability have been added. All the algorithms are written in C and run on a AMD X2 processor with 2.2 GHz with Linux operating system.
In Tables 7.1–7.6 (7.3–7.8) we present the results of the experimental campaign on the dispensing (counting) department, respectively. Each row of the tables represents the results obtained by applying a different greedy heuristic. Namely, we report on the results of MJS (which corresponds to setting $\Delta = 0$ in the Delta algorithm), and two versions of the Delta algorithm obtained by setting Δ equals to 8 hours (i.e., half time shift) and to the average processing time of all the jobs in the instance. We refer to them as Δ_{8h} and Δ_{avg}, respectively. For each algorithm we present the computation time (Time) in seconds, the objective functions values (maximum lateness L_{max}, makespan C_{max} and number of tardy jobs U) and the percentage of feasible instances (%feas), i.e., instances in which the deadline constraints are not violated. All the values in each row of the tables are the average over 15 instances.
In Tables 7.1–7.2 we show the results of the proposed algorithms on the test instances with deadlines. The computation time never exceeds one second when a rollout algorithm is applied, and it is negligible when H and H+LS algorithms are applied. As far as the solution quality is concerned we can see that MJS heuristic performs worse than Delta variants, although tends to find more feasible solutions. We observe that, when the Delta heuristic is used it is able to obtain better results than MJS and the two variants (Δ_{8h} and Δ_{avg}) yield exactly to the same solutions. However, when studying the effects of the different algorithmic schemes the performance indicators obtained by RH+LS are greatly improved: The maximum lateness and the makespan are reduced of about 1000 seconds, the number of late jobs is more than halved and

the solutions obtained are always feasible, i.e., all the deadlines are respected. From results reported in Table 7.2 it is clear that the main contribution to these improvements is due to the rollout algorithm.

Heuristic	H					H+LS				
	Time	L_{max}	C_{max}	U	%feas	Time	L_{max}	C_{max}	U	%feas
MJS	<0.01	4313.87	20813.53	14.47	1.00	<0.01	4134.47	20725.40	12.13	1.00
Δ_{8h}	<0.01	3965.47	20794.87	11.60	0.67	<0.01	3799.87	20643.93	8.47	0.73
Δ_{avg}	<0.01	3965.47	20794.87	11.60	0.67	<0.01	3799.87	20643.93	8.47	0.73

Table 7.1: Dispensing department: Instances with deadlines (H and H+LS)

Heuristic	RH					RH+LS				
	Time	L_{max}	C_{max}	U	%feas	Time	L_{max}	C_{max}	U	%feas
MJS	0.53	3192.53	19878.40	7.20	1.00	0.53	3140.13	19878.40	6.73	1.00
Δ_{8h}	0.54	2972.13	19879.73	4.73	1.00	0.54	2972.13	19879.73	4.67	1.00
Δ_{avg}	0.53	2972.13	19879.73	4.73	1.00	0.53	2972.13	19879.73	4.67	1.00

Table 7.2: Dispensing department: Instances with deadlines (RH and RH+LS)

Heuristic	H					H+LS				
	Time	L_{max}	C_{max}	U	%feas	Time	L_{max}	C_{max}	U	%feas
MJS	0.04	527.93	17312.73	3.00	1.00	0.07	509.80	17312.67	3.00	1.00
Δ_{8h}	0.04	526.27	17310.47	4.07	1.00	0.06	526.27	17310.47	4.07	1.00
Δ_{avg}	0.04	447.00	17319.13	3.27	1.00	0.06	446.93	17319.07	3.00	1.00

Table 7.3: Counting department: Instances with deadlines (H and H+LS)

Heuristic	RH					RH+LS				
	Time	L_{max}	C_{max}	U	%feas	Time	L_{max}	C_{max}	U	%feas
MJS	2148.28	459.13	17257.53	1.80	1.00	2148.31	457.87	17257.53	1.80	1.00
Δ_{8h}	2208.72	453.33	17256.07	2.67	1.00	2208.74	453.33	17256.07	2.60	1.00
Δ_{avg}	2230.41	358.13	17280.60	1.40	1.00	2230.43	358.13	17280.60	1.40	1.00

Table 7.4: Counting department: Instances with deadlines (RH and RH+LS)

Heuristic	H				H+LS			
	Time	L_{max}	C_{max}	U	Time	L_{max}	C_{max}	U
MJS	<0.01	3742.13	20672.47	10.47	<0.01	3664.87	20595.20	9.67
Δ_{8h}	<0.01	3762.60	20692.93	10.67	<0.01	3685.33	20615.67	9.80
Δ_{avg}	<0.01	3762.60	20692.93	10.67	<0.01	3685.33	20615.67	9.80

Table 7.5: Dispensing department: Instances without deadlines (H and H+LS)

Heuristic	RH				RH+LS			
	Time	L_{max}	C_{max}	U	Time	L_{max}	C_{max}	U
MJS	0.53	2625.27	19568.07	5.27	0.53	2625.27	19568.07	5.27
Δ_{8h}	0.54	2708.93	19651.73	6.67	0.54	2708.93	19651.73	6.07
Δ_{avg}	0.54	2708.93	19651.73	6.67	0.54	2708.93	19651.73	6.07

Table 7.6: Dispensing department: Instances without deadlines (RH and RH+LS)

When tackling instances without deadlines (see Table 7.5 and 7.6) we observe that the MJS algorithm is able to produce slightly better results. Clearly in these cases the column %feas is not reported.
In Tables 7.3 and 7.4 we show the results obtained for the counting department when a deadline is present, whereas in Tables 7.7 and 7.8 we report on the results for instances without deadline.
When used as stand-alone heuristic (see Table 7.3) Δ_{avg} is the heuristic able to produce better results. However when tackling instances without deadlines (Table 7.7) MJS results to be slightly better. In the counting departments all the algorithms are able to find always a solution respecting the deadline constraints. Regarding the computation times we note that more than half an hour is needed to terminate the rollout procedure. This is clearly due to the larger number of jobs in these instances. However such computation time does not affect the applicability of this approach since the amount of time in which a solution should be produced by the automated scheduling system is usually larger. Moreover we observe that the LS phase requires always an almost negligible time. On the other hand, the solution quality of the rollout algorithms is significant better than the H+LS. We also observe that no significative difference arises when comparing RH and RH+LS, or when comparing the effects of different pilot heuristics.
Finally, regarding the results when no deadline is imposed, the larger influence on the solution quality is again due to the application of the rollout algorithm. On the other hand, also in this case there is a clear trade-off between the computational effort and the solution quality.

Heuristic	H				H+LS			
	Time	L_{max}	C_{max}	U	Time	L_{max}	C_{max}	U
MJS	0.04	924.47	17779.67	5.40	0.04	924.07	17779.27	5.40
Δ_{8h}	0.04	1014.13	17786.20	6.20	0.04	1010.80	17782.87	6.07
Δ_{avg}	0.04	924.73	17779.93	5.27	0.04	924.33	17779.53	5.13

Table 7.7: Counting department: Instances without deadlines (H and H+LS)

Heuristic	RH				RH+LS			
	Time	L_{max}	C_{max}	U	Time	L_{max}	C_{max}	U
MJS	2214.80	605.13	17571.40	2.40	2214.80	605.13	17571.40	2.40
Δ_{8h}	2176.31	716.73	17510.80	3.80	2176.31	716.73	17510.80	3.80
Δ_{avg}	2163.06	605.13	17571.40	2.40	2163.07	605.13	17571.40	2.40

Table 7.8: Counting department: Instances without deadlines (RH and RH+LS)

In Figure 7.7 and Figure 7.8 we plot the improvements over the basic standalone heuristics due to the proposed improvement schemes for the dispensing and counting department respectively. The percentage improvement is obtained as follows: $(1 - x_v^i)/x_H^i$, where $i \in \{L_{max}, C_{max}, U\}$, $v = \{H, H + LS, RH, RH + LS\}$ and x_v^i is the solution quality according to the objective function i obtained by algorithm v. Hence, percentage improvement on L_{max} possibly gets values $> 100\%$ when, starting from $L_{max} \geq 0$, a new solution with $L_{max} < 0$ is obtained. In the dispensing department the main contribution is given by the application of the rollout algorithm, although the use of a VND allows to slightly improve the solution quality without increasing significantly the computation time. For both departments the algorithms are able to obtain relevant improvements in L_{max} and U objectives. In particular, L_{max} is improved up to 27% on average, and U is improved of about 50%. A more detailed analysis of the results shows that in about 50% of the instances no late jobs are present. The main difference between the two departments is that, in the counting department the algorithms are able to improve the C_{max} objective by only 1%, while the reduction in the dispensing department is up to 5%. This is due to the presence of sequence dependent setups in the dispensing which allow to improve the makespan objective, while in the counting department the makespan reduction is indeed marginal.

In a first step of the algorithms development phase, we implemented several simple greedy procedures. Some algorithms were simple list schedules such as the Earliest Due Date, Shortest Processing Time and similar. Other greedy heuristics were insertion heuristics or more sophisticated algorithms. The pur-

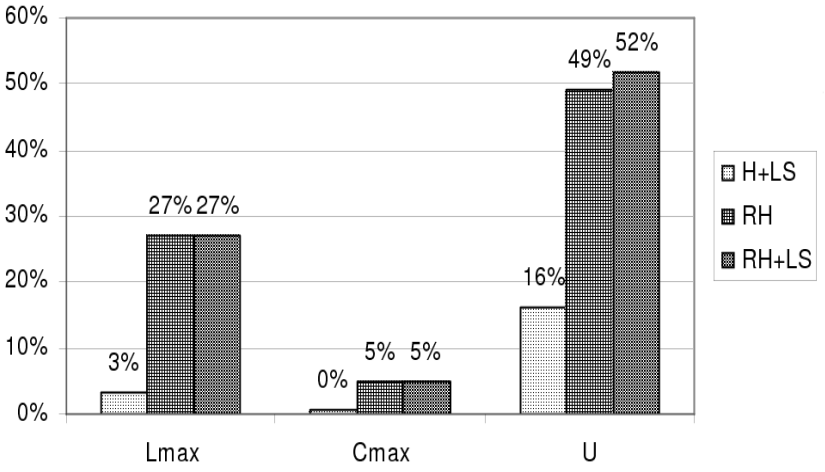

Figure 7.7: Average improvements in the dispensing department

pose of the first phase was that of verifying the correctness of the models with the users, and looking for useful properties of the problem instances to exploit in the algorithms. Discussion with the users led to validation of the models described in the previous section and to a description of the behavior of human scheduler when building a feasible schedule. In fact, the human schedulers in the plant did not follow any formal procedure to schedule production orders and even did not use any formal definition of quality of a schedule. The schedules were simply the result of schedulers intuition and past experience. However, the schedules produced by hand in the counting department were quite similar to that of the MJS algorithm illustrated in Figure 7.2, a modified version of the Jackson Schedule [92]. At the dispensing department, the schedulers strived to obtain large campaigns other than respecting the due dates. Algorithm Delta in Figure 7.3 is a surrogate of their behavior.
The lack of detailed information on the performance of the previous scheduling process does not allow a direct and complete comparison with the new system.

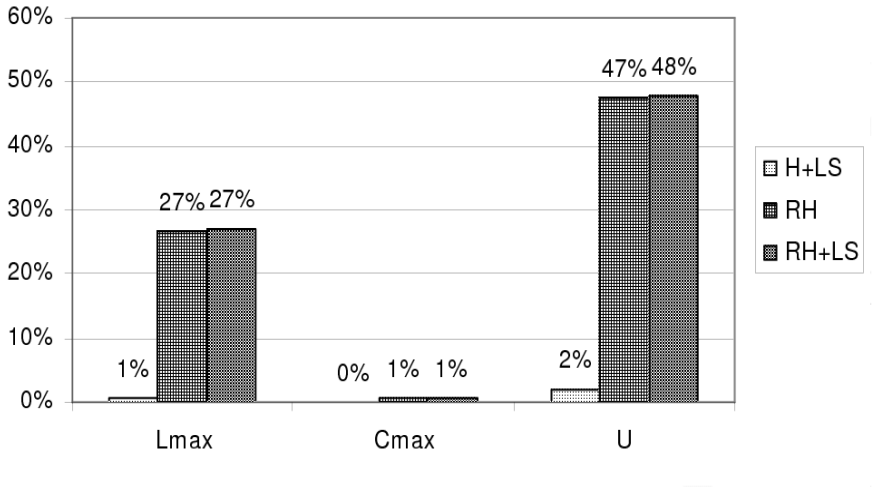

Figure 7.8: Average improvements in the counting department

As far as the dispensing and the counting departments are concerned, the assessment of the new system was based on the feedback from the department managers. Specifically, the solutions provided by the stand-alone greedy algorithms were considered not satisfactory, since it was simple for the users to improve the solutions with frequent and tedious local changes in the schedules. On the other hand, the schedules obtained after the rollout and VND phases were rarely improved by users. The increase of capacity for the dispensing department has been estimated up to one hour in a day by the department manager, which approximately corresponds up to the 2% of productivity increase. Similar results were obtained for the counting department.

7.6 Conclusions

In this chapter we described a production scheduling problem arising in secondary pharmaceutical manufacturing. More specifically, we addressed the automation of the scheduling process for the dispensing and counting departments. It results in a complex scheduling problem involving both standard and

complex uncommon constraints. Any improvement in these departments is directly translated into a relaxation of the timing constraints for more critical departments. We proposed an algorithmic technique that can be effectively used to rapidly automate the production scheduling process and we presented a practical implementation in a pharmaceutical manufacturing plant. Results are encouraging and show a substantial improvement over simple heuristic techniques surrogating the schedules produced by hand by the plant schedulers.

Chapter 8

Conclusions

Production optimization methods are widely used in manufacturing environments. Very often just one criterion is taken into account to be optimized while nowadays a single criterion is deemed as insufficient for real and practical applications. In order to preserve their competitiveness and market share, companies must keep the production costs low and maximize the quality and costumer's satisfaction. These factors are contrasting and this means that decisions to take are not straightforward. On the other side, costumers become more and more exigent and the need to ship orders on time often contrasts with the need of balancing workload in production lines. Hence it is very important to have an optimization system which could take into account more than one objective at a time, providing a wider view over the problem considering both costs and priorities.
Multi-objective optimization is without a doubt a very important research topic not only because of the multi-objective nature of most real-world problems, but also because there are still many open questions in this area. Over the last decade, multi-objective optimization has received a big impulse in Operations Research. Some new techniques have been developed in order to deal with functions and real-world problems that have multiple objectives, and many approaches have been proposed.
The easiest way of dealing with a multi-objective problem is the so called "a priori" approach where two or more objectives are weighted and combined into a single measure, usually linear.

A more desirable approach is the "a posteriori" method. In this case, the aim is to obtain many solutions with as many associated values as objectives. In such cases, the traditional concept of "optimum" solution does not apply. In this context, a set of solutions is obtained where their objective values form what is referred to as the Pareto front. In the Pareto front all solutions are equally good, since there is no way of telling which one is better or worse. In other words, all solutions belonging to a Pareto front are the "best" solutions for the problem in a multi-objective sense.
The majority of real life problems arising from classical production environments (as jobshop, flowshop, parallel machines) belong to the class of $\mathcal{NP}$-hard problems and indeed it is improbable that could exist an algorithm for solving them in polynomial time. Hence every exact approach proposed in literature works well only for small size instances while real-world cases seldom can be solved using this methods. These problems call for a metaheuristic approach that allows to tackle large instances. Although not granting the "optimum", metaheuristic algorithms are able to find in relatively short time good quality solutions.
In chapter 2 the most important metaheuristic algorithms have been presented along with a description of the elements that are characteristic of this class of methods.
In this thesis several production environments belonging to the classes of permutation flowshop (PFSP) and parallel machines have been considered. Such environments are important both for their practical and theoretical reasons. Real cases of ceramic and pharmaceutical productions have been reported. Unfortunately the majority of the literature for the PFSP is centred on a single optimization criterion and there are no comprehensive reviews in the literature aboutflowshops with several objectives. Hence, we carried out this work and presented it in chapter 3. In the past years a number of interesting algorithms have been proposed. However, new proposals are often not validated against the existing methods and when done, the quality indicators used are not appropriate. Additionally, the multi-objective literature is rich on advanced methods that have not been applied to the PFSP before. Therefore we identified the most promising methods for the most common criteria combination. We evaluate a total of 23 methods, from local search metaheuristics such as tabu search or simulated annealing to evolutionary approaches like genetic algorithms. Furthermore, we use the latest Pareto-compliant quality measures for assessing the effectiveness of the tested methods. Careful and comprehensive statistical testing is employed to ensure the accuracy of the conclusions. As a result, we have identified the best performing methods from the literature which constitutes a

reference work for further research. Successively we extended the model presented in the chapter 4 considering the presence of sequence dependent setup times. In literature many papers have tackled the flowshop scheduling problem with setups but, although it is definitely a relevant topic, according to our knowledge nothing has been published dealing with both the optimization of more than one objective and sequence dependent setup times Then we presented an algorithm (IPG) of new conception and we compared it with the most performing approaches. Again, statistical techniques are employed to prove it widely outperforms all other approaches for the bicriteria problem of $C_{max} - \sum_i w_i T_i$ Pareto optimization.
The comparative evaluation not only includes flowshop-specific algorithms but also adaptations of other general methods proposed in the multi-objective optimization literature. A set of benchmark instances, based on the well known benchmark of [193] has been employed and a comprehensive statistical analysis of the results has been conducted with both parametric as well as non-parametric techniques. Overall, our Iterated Pareto greedy can be regarded as the best performer under our experimental settings.
The second case study tackled in this thesis arises in the coordination between two consecutive departments of a production system, where parts are processed in batches, and each batch is characterized by two distinct attributes. Due to the lack of interstage buffering between the two stages, these departments have to follow the same batch sequence. In the first department, a setup occurs every time the first attribute of a new batch is different from the one of the previous batch. In the downstream department, there is a setup when the second attribute changes in two consecutive batches. The problem consists in finding a batch sequence optimizing the number of setups paid by each department. This case results in a particular bi-objective combinatorial optimization problem. I presented a geometrical characterization for the feasible solution set of the problem, and three effective metaheuristics have been proposed, as well as an extensive experimental campaign. Such approach can be also used to solve a class of single-objective problems, in which setup costs in the two departments are general increasing functions of the number of setups. In the last chapter we have described a general methodology for operations scheduling in dispensing and counting departments of pharmaceutical manufacturing plants. The departments are modelled as a multi-objective parallel machines scheduling problem under a number of both standard and realistic constraints, such as release times, due dates and deadlines, particular sequence-dependent setup times, machine unavailabilities, and maximum campaign size. Main characteristics of this methodology (Rollout/Pilot method) are the modularity of the

solution algorithms, the adaptability to different objectives and constraints to fulfill production requirements, the easiness of implementation, and the ability of incorporating human experience in the scheduling algorithms. Computational experience carried out on two case studies from a real pharmaceutical plant shows the effectiveness of this metaheuristic approach.
The aim of this Ph.D thesis is to demonstrate the effectiveness, adaptability and modularity of different metaheuristics applied on real life difficult scheduling problems. Computational campaigns of test have been performed and all results are been evaluated by means of statistical tools.
However nowadays not all the basic mechanisms to create high performing metaheuristics are known. A possible extension of this work consists in deepening the knowledge of such mechanisms in order to identify those elements that confer effectiveness and those that, instead, disturb the solution process. Then using such know-how acquired with this study it will be possible develop more efficient algorithms able to fast solve larger size problems. A second possible extension of this work is to apply the existing metaheuristics to tackle complex real life problems with several unusual constraints. As example in many case the available resources are limited and must be shared among machines. Such complex cases call for algorithms able to adapt them-self employing a learning process to select those procedures that are more effective for a particular instance.
Finally a future evolution of the methodologies presented in this thesis is the possibility to manage an on line environment. All the algorithms implemented here, in fact, work optimizing the production of short time intervals (often no more than two weeks). In real life, new orders can arrive during such intervals and it may be convenient to reschedule the production. Different criteria could be considered in this case as, for example, the minimization of the total number of changes in schedules due to rescheduling or the workload balancement.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1990.

[2] G. Adamopoulos and C. Pappis. Scheduling under a common due date on parallel unrelated machines. *European Journal of Operational Research*, 105:494–501, 1998.

[3] A. Agnetis, A. Alfieri, and G. Nicosia. A heuristic approach to batching and scheduling a single machine to minimize setup costs. *Computers & Industrial Engineering*, 46:793–802, 2004.

[4] A. Agnetis, P. Detti, C. Meloni, and D. Pacciarelli. Setup coordination between two stages of a supply chain. *Annals of Operations Research*, 107:15–32, 2001.

[5] R. Akkiraju, P. Keskinocak, S. Marthy, and F. Wu. An agent-based approach for scheduling multiple machines. *Applied Intelligence*, 14:135–144, 2001.

[6] M. H. Al-Haboubi and S. Z. Selim. A sequencing problem in the weaving industry. *European Journal of Operational Research*, 66:65–71, 1993.

[7] B. Alidaee and A. Ahmadian. Two parallel machine sequencing problems involving controllable job processing times. *European Journal of Operational Research*, 70:335–341, 1993.

[8] A. Allahverdi. The two- and m-machine flowshop scheduling problems with bicriteria of makespan and mean flowtime. *European Journal of Operational Research*, 147(2):373–396, 2003.

[9] A. Allahverdi. A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. *Computers & Operations Research*, 31(2):157–180, 2004.

[10] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27:219–239, 1999.

[11] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[12] V. A. Armentano and J. E. C. Arroyo. An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10(5):463–481, 2004.

[13] J. E. C. Arroyo and V. A. Armentano. A partial enumeration heuristic for multi-objective flowshop scheduling problems. *Journal of the Operational Research Society*, 55(9):1000–1007, 2004.

[14] J. E. C. Arroyo and V. A. Armentano. Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, 167(3):717–738, 2005.

[15] T. P. Bagchi. *Multiobjective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[16] T. P. Bagchi. Pareto-optimal solutions for multi-objective production scheduling problems. In E. Zitzler, K. Deb, L. Thiele, Carlos A. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, pages 458–471. Springer, 2001.

[17] N. Balakrishnan, J.J Kanet, and S. V. Sridharan. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, 26:127–141, 1999.

[18] J. Bank and F. Werner. Heuristic algorithms for unrelated parallel-machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modeling*, 33:363.383, 2001.

[19] M. Basseur. *Conception d'algorithmes coopératifs pour l'optimisation multi-objectif: Application aux problèmes d'ordonnancement de type Flow-shop.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille. Lille, France., 2005. In French.

[20] R. Battiti and M. Protasi. Reactive search, a history-based heuristic for max-sat. *ACM Journal of Experimental Algorithmics*, 2, 1997.

[21] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Technical report, Caltech, Pasadena, CA,, manuscript, 1986.

[22] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[23] D. P. Bertsekas. Dynamic programming and suboptimal control: A survey from adp to mpc. *European Journal on Control*, 11(4–5):310–334, 2005.

[24] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[25] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3:245–262, 1997.

[26] P. Brucker. *Scheduling Algorithms.* Springer-Verlag, Berlin, Germany, 4 edition, 2004.

[27] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems. part i. *Discrete Appl. Math.*, 65(1-3):97–122, 1996.

[28] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems. part ii. *Discrete Appl. Math.*, 72(1-2):47–69, 1997.

[29] P. Calégari, G. Coray, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5(2):145–158, 1999.

[30] S. A. Canuto, M. G. C. Resende, and C. C. Ribeiro. Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks*, 38:50–58, 2001.

[31] S. Cavalieri and P. Gaiardelli. Hybrid genetic algorithms for a multiple-objective scheduling problem. *Journal of Intelligent Manufacturing*, 9(4):361–367, 1998.

[32] V. Černý. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[33] K. Chakravarthy and C. Rajendran. A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning and Control*, 10(7):707–714, 1999.

[34] P. C. Chang, J. C. Hsieh, and S. G. Lin. The development of gradual-priority weighting approach for the multi-objective flowshop scheduling problem. *International Journal of Production Economics*, 79(3):171–183, 2002.

[35] T. Cheng and Z.-L. Chen. Parallel-machine scheduling problems with earliness and tardiness penalties. *Journal of the Operational Research Society*, 45:685–695, 1994.

[36] T. C. E. Cheng and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1990.

[37] J. Choi, M. J. Realff, and J. H. Lee. Approximate dynamic programming: Application to process supply chain management. *AIChE Journal*, 52(7):2473–2485, 2006.

[38] F. D. Chou and C. E. Lee. Two-machine flowshop scheduling with bicriteria problem. *Computers & Industrial Engineering*, 36(3):549–564, 1999.

[39] G. C. Cole. *Pharmaceutical production facilities. Design and applications (2nd edition).* CRC Press, 1998.

[40] R. K. Congram, C. N. Potts, and S. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[41] W. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, third edition, 1999.

[42] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[43] S. T. Mc Cormick and M. Pinedo. Scheduling n independant jobs m uniform machines with both flowtime and makespan objectives: a parametric analysis. *ORSA Journal on Computing*, 7(1):63–77, 1995.

[44] D. W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, Max H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 283–290, San Francisco, California, USA, 2001. Morgan Kaufmann.

[45] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto envelope-based selection algorithm for multiobjective optimization. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, Juan J. Merelo Guervós, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France, September 18-20, 2000, Proceedings*, volume 1917 of *Lecture Notes in Computer Science*, pages 839–848. Springer, 2000.

[46] R. L. Daniels and R. J. Chambers. Multiobjective flow-shop scheduling. *Naval research logistics*, 37(6):981–995, 1990.

[47] D. de Werra and A. Hertz. Tabu-search techniques a tutorial and an application to neural networks. *Operations Research Spektrum*, 11:131–141, 1989.

[48] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, West Sussex, England, 2001.

[49] K. Deb. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[50] K. Deb, M. Mohan, and S. Mishra. A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions. Technical Report 2003002, Kanpur Genetic Algorithms Laboratory (KanGAL), Kampur, India, February 2002.

[51] P. Detti, C. Meloni, and M. Pranzo. Local search algorithms for the minimum cardinality dominating trail set of a graph. *Technical report RT-DIA-84-2003 Dipartimento di Informatica e Automazione, Università Roma Tre, Roma, Italy*, 2003.

[52] P. Detti, C. Meloni, and M. Pranzo. Simple bounds for the minimum cardinality dominating trail set problem. *Technical report RT-DIA-87-2004 ipartimento di Informatica e Automazione, Università Roma Tre, Roma, Italy*, 2004.

[53] P. Detti, C. Meloni, and M. Pranzo. Minimizing and balancing setups in a serial production system. *to appear on International Journal of Production Research*, 2006.

[54] B. L. Dietrich. A two-phase heuristic for scheduling on unrelated parallel machines with setup. Technical Report RC 14330, IBM TJ Watson Research Center, York Town Heights, NY, 1989.

[55] J. Du and J. Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.

[56] C. Duin and S. Voß. The pilot method: a strategy for heuristic repetition with application to the steiner problem in graphs. *Networks*, 34:181–191, 1999.

[57] M. Ehrgott. Multicriteria optimization. In Springer-Verlag, editor, *Lecture Notes in Economics and Mathematical Systems*, volume 491, 2000.

[58] A. El-Bouri, S. Balakrishnan, and N. Popplewell. A neural network to enhance local search in the permutation flowshop. *Computers & Industrial Engineering*, 49(1):182–196, 2005.

[59] H. Emmons. Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics*, 34:803–810, 1987.

[60] G. Feng and H. C. Lau. Efficient algorithm for machine scheduling problems with earliness and tardiness penalties. *Proceedings of the* 2^{nd} *Multidisciplinary International Conference of Scheduling: Theory and Applications, New York, USA, July 18-21*, pages 196–211, 2005.

[61] J M. Framinan, J. N. D. Gupta, and R. Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan

objective. *Journal of the Operational Research Society*, 55(12):1243–1255, 2004.

[62] J. M. Framinan and R. Leisten. A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics*, 99(1-2):28–40, 2006.

[63] J. M. Framinan, R. Leisten, and R. Ruiz-Usano. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141(3):559–569, 2002.

[64] R. Gangadharan and C. Rajendran. A simulated annealing heuristic for scheduling in a flowshop with bicriteria. *Computers & Industrial Engineering*, 27(1-4):473–476, 1994.

[65] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP completeness*. Freeman, 1979.

[66] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[67] M. J. Geiger. On operators and search space topology in multi-objective flow shop scheduling. *European Journal of Operational Research*, 2007. In press.

[68] A. M. Geoffrion and G. W. Graves. Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/lp approach. *Operations Research*, 24:595–610, 1976.

[69] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[70] F. Glover. Tabu search– part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[71] F. Glover. Tabu search– part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[72] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[73] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

[74] T. Gonzalez and S. Sahni. Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26(1):36–52, 1978.

[75] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[76] J. N. D. Gupta, K. Hennig, and F. Werner. Local search heuristics for two-stage flow shop problems with secondary criterion. *Computers & Operations Research*, 29(2):123–149, 2002.

[77] J. N. D. Gupta, V. R. Neppalli, and F. Werner. Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *International Journal of Production Economics*, 69(3):323–338, 2001.

[78] J. N. D. Gupta, N. Palanimuthu, and C. L. Chen. Designing a tabu search algorithm for the two-stage flow shop problem with secondary criterion. *Production Planning & Control*, 10(3):251–265, 1999.

[79] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. *Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy*, 1986.

[80] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[81] P. Hansen and N. Mladenović. A tutorial on variable neighborhood search. *European Les Cahiers du GERAD G–2003*, 46, 2003.

[82] S. Hasija and C. Rajendran. Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42(11):2289–2301, 2004.

[83] R. B. Heady and Z. Zhu. Minimizing the sum of job earliness and tardiness in a multimachine system. *International Journal of Production Research*, 36, 1619-1632.

[84] S. R. Hejazi and S. Saghafian. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43(14):2895–2929, 2005.

[85] A. Hertz and D. kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.

[86] A. Hertz and M. Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151(2):247–252, 2003.

[87] J. C. Ho and Y.-L. Chang. A new heuristic for the n-job, M-machine flow-shop problem. *European Journal of Operational Research*, 52:194–202, 1991.

[88] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592–623, 2005.

[89] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, 2004.

[90] H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems Man and Cybernetics*, 28(3):392–403, 1998.

[91] H. Ishibuchi, T. Yoshida, and T. Murata. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223, 2003.

[92] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report 43, Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

[93] L. W. Jacobs and M. J. Brusco. A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42(7):1129–1140, 1995.

[94] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel-machines. *ACM Symposium on Theory of Computing*, pages 408–417, 1999.

[95] D. S. Johnson. Local optimization and the travelling salesman problem. volume 443 of LNCS, pages 446–461. Springer Verlag, 1990.

[96] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing : an Experimental evaluation ; Part I, Graph Partitioning,. *Operations Research*, 37:365–892, 1989.

[97] D. S. Johnson and L. A. McGeoch. *The travelling salesman problem: A case study in local optimization*, pages 215–310. John Wiley & Sons, Chichester, England, 1997.

[98] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68, 1954.

[99] D. F. Jones, S. K. Mirrazavi, and M. Tamiz. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137(1):1–9, 2002.

[100] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, New York, 1972.

[101] K. Katayama and N. Narihisa. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proceedings of GECCO99*, volume 1, pages 321–328. Morgan Kaufmann, 1999.

[102] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[103] J. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[104] J. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, February 2006. revised version.

[105] J. B. Kollat and P. M. Reed. The value of online adaptive search: A performance comparison of nsgaii, $\varepsilon-$nsgaii and $\varepsilon-$moea. In Carlos A. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005, Proceedings*, volume 3410 of *Lecture Notes in Computer Science*, pages 386–398. Springer, 2005.

[106] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.

[107] G. Laporte, J. J. Salazar, and F. Semet. Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 35(1-?9), 2003.

[108] C. E. Lee and F. D. Chou. A two-machine flowshop scheduling heuristic with bicriteria objective. *International Journal of Industrial Engineering*, 5(2):128–139, 1998.

[109] W. C. Lee and C. C. Wu. Minimizing the total flow time and the tardiness in a two-machine flow shop. *International Journal of Systems Science*, 32(3):365–373, 2001.

[110] Y. H. Lee and M. Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100:464–474, 1997.

[111] J. Lemesre, C. Dhaenens, and E. G. Talbi. An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research*, 177(3):1641–1655, 2007.

[112] J. Y.-T. Leung and G. H Young. Minimizing schedule length subject to minimum flow time. *SIAM Journal of Computing*, 18(2):314–326, 1989.

[113] C.-L Li and T. Cheng. The parallel machine min-max weighted absolute lateness scheduling problem. *Naval Research Logistics*, 41:33–46, 1994.

[114] C.-J. Liao, W.-C. Yu, and C.-B. Joe. Bicriterion scheduling in the two-machine flowshop. *Journal of the Operational Research Society*, 48(9):929–935, 1997.

[115] B. M. T. Lin and J. M. Wu. Bicriteria scheduling in a two-machine permutation flowshop. *International Journal of Production Research*, 44(12):2299–2312, 2006.

[116] T. Loukil, J. Teghem, and P. Fortemps. Solving multi-objective production scheduling problems with tabu search. *Control and Cybernetics*, 29(3):819–828, 2000.

[117] T. Loukil, J. Teghem, and D. Tuyttens. Solving multi-objective production scheduling problems using metaheuristics. *European Journal of Operational Research*, 161(1):42–61, 2005.

[118] H. R. Lourenço. Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[119] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. Economics Working Papers 513, Department of Economics and Business, Universitat Pompeu Fabra, November 2000. available at http://ideas.repec.org/p/upf/upfgen/513.html.

[120] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353, Boston, 2003. Kluwer Academic Publishers.

[121] H. R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem, 1996.

[122] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34:111–124, 1986.

[123] E. Marchiori and A. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In S. Cagnoni et al., editors, *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, volume 1803 of *Lecture Notes in Computer Science*, pages 367–381. Springer Verlag, Berlin, Germany, 2000.

[124] O. Martin and S. W. Otto. Partitoning of unstructured meshes for load balancing. . *Concurrency: Practice and Experience*, 7:303–314, 1995.

[125] O. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[126] O. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.

[127] R. I. McIntosh, S. J. Culley, A. R. Mileham, and G. W. Owen. Changeover improvement: A maintenance perspective. *International Journal of Production Economics*, 73:153–163, 2001.

[128] N. Melab, M. Mezmaz, and E. G. Talbi. Parallel cooperative metaheuristics on the computational grid. a case study: the bi-objective flowshop problem. *Parallel Computing*, 32(9):643–659, 2006.

[129] C. Meloni, D. Pacciarelli, and M. Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131:215–235, 2004.

[130] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[131] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1994.

[132] S. Mohri, T. Masuda, and H. Ishii. Bi-criteria scheduling problem on three identical parallel machines. *International Journal of Production*, 60-61:529–536, 1999.

[133] D.C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, sixth edition, 2004.

[134] T. Murata, H. Ishibuchi, and M. Gen. Specification of genetic search directions in cellular multi-objective genetic algorithms. In E. Zitzler, K. Deb, L. Thiele, Carlos A. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2001.

[135] T. Murata, H. Ishibuchi, and H. Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & Industrial Engineering*, 30(4):957–968, 1996.

[136] A. Nagar, S. S. Heragu, and J. Haddock. A branch-and-bound approach for a two-machine flowshop scheduling problem. *Journal of the Operational Research Society*, 46(6):721–734, 1995.

[137] A. Nagar, S. S. Heragu, and J. Haddock. Mutiple and bicriteria scheduling: A lierature survey. *European Journal of Operational Research*, 81(1):88–104, 1995.

[138] A. Nagar, S. S. Heragu, and J. Haddock. A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem. *Annals of Operations Research*, 63(3):397–414, 1996.

[139] M. Nawaz, E. E. Enscore Jr, and I. Ham. A heuristic algorithm for the m machine, n job flowshop sequencing problem. *Omega-International Journal of Management Science*, 11(1):91–95, 1983.

[140] V. R. Neppalli, C.-L. Chen, and J. N. D. Gupta. Genetic algorithms for the two-stage bicriteria flowshop problem. *European Journal of Operational Research*, 95(2):356–373, 1996.

[141] P. F. Ostwald. Cost estimating. *Handbook of Industrial Engineering*, 2nd ed.:1263–1288, 1992.

[142] IM. Ovacik and R. Uzsoy. *Decomposition methods for complex factory scheduling problems*. Kluwer, 1997.

[143] S. Panwalker, R. Dudek, and M. Smith. Sequencing research anth the industrial scheduling problem. *Symposium on the Theory of Scheduling and its Applications*, pages 29–38, 1973.

[144] C. H. Papadimitriou. *Combinatorial Complexity*. Addison-Wsley, Amsterdam, The Netherlands, The Netherlands, 1994.

[145] L. F. Paquete. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Method and Analysis*. PhD thesis, Computer Science Department. Darmstadt University of Technology. Darmstadt, Germany, 2005.

[146] S. Parthasarathy and C. Rajendran. An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International Journal of Production Economics*, 49(3):255–263, 1997.

[147] S. Parthasarathy and C. Rajendran. A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs - a case study. *Production Planning & Control*, 8(5):475–483, 1997.

[148] T. Pasupathy, C. Rajendran, and R. K. Suresh. A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *The International Journal of Advanced Manufacturing Technology*, 27(7-8):804–815, 2006.

[149] M. E. Pfund, J. W. Fowler, and J. N. D. Gupta. A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of Chinese Institute of Industrial Engineers*, 21(3):230–241, 2004.

[150] M. E. Pfund, L. Yu, W. M. Carlyle, and J. W. Fowler. The effects of processing time variability and equipment downtimes on various scheduling approaches for a printed wiring board assembly operation. *Journal of Electronics Manufacturing*, 11:19–31, 2002.

[151] B. Piachaud. *Outsourcing of R&D in the Pharmaceutical Industry: From Conceptualization to Implementation of the Strategic Sourcing Process.* Palgrave Macmillan, 2005.

[152] M. Pinedo. *Theory, algorithms, and systems.* Prentice-Hall, 1995.

[153] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Upper Saddle, N.J, second edition, 2002.

[154] S. G. Ponnambalam, H. Jagannathan, M. Kataria, and A. Gadicherla. A TSP-GA multi-objective algorithm for flow-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 23(11-12):909–915, 2004.

[155] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[156] S. Radhakrishnan and J. A. Ventura. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38:2233–2252, 2000.

[157] A. R. Rahimi-Vahed and S. M. Mirghorbani. A multi-objective particle swarm for a flow shop scheduling problem. *Journal of Combinatorial Optimization*, 13(1):79–102, 2007.

[158] C. Rajendran. Two-stage flowshop scheduling problem with bicriteria. *Journal of the Operational Research Society*, 43(9):871–884, 1992.

[159] C. Rajendran. A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multicriteria. *International Journal of Production Research*, 32(11):2541–2558, 1994.

[160] C. Rajendran. Heuristics for scheduling in flowshop with multiple objectives. *European Journal of Operational Research*, 82(3):540–555, 1995.

[161] C. Rajendran and H. Ziegler. A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. *Computers & Industrial Engineering*, 33(1-2):281–284, 1997.

[162] C. Rajendran and H. Ziegler. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research*, 149(3):513–522, 2003.

[163] C. Rajendran and H. Ziegler. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Computers & Industrial Engineering*, 48(4):789–797, 2005.

[164] D. Ravindran, A. N. Haq, S. J. Selvakuar, and R. Sivaraman. Flow shop scheduling with multiple objective of minimizing makespan and total flow time. *The International Journal of Advanced Manufacturing Technology*, 25(9-10):1007–1012, 2005.

[165] R. Z. Ríos-Mercado and J. F. Bard. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25(5):351–366, 1998.

[166] R. Z. Ríos-Mercado and J. F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76–98, 1998.

[167] R. Z. Ríos-Mercado and J. F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5(1):53–70, 1999.

[168] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

[169] R. Ruiz, C. Maroto, and J. Alcaraz. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1):34–54, 2005.

[170] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[171] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[172] A. J. Ruiz-Torres, E. E. Enscore, and R. R. Barton. Simulated annealing heuristics for the average flow-time and the number of tardy jobs bi-criteria identical parallel machine problem. *Computers & Industrial Engineering*, 33(1-2):257–271, 1997.

[173] S. C. Sarin and R. Hariharan. A two machine bicriteria scheduling problem. *International Journal of Production Economics*, 65:125–139, 2000.

[174] S. Sayın and S. Karabatı. A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research*, 113(2):435–449, 1999.

[175] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[176] N. Secomandi. Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics*, 9:321–352, 2003.

[177] W. J. Selen and D. D. Hott. A mixed-integer goal-programming formulation of the standard flowshop scheduling problem. *Journal of the Operational Research Society*, 37(12):1121–1128, 1986.

[178] N. Shah. Pharmaceutical supply chains: key issues and strategies for optimisation. *Computers and Chemical Engineering*, 28:929–941, 2004.

[179] D. B. Shmoys and E. Tardos. Scheduling unrelated machines with costs. *Proceedings of the 4th Annual ACM-SIAM Symposiu, Austin, Texas*, pages 448–454, 25-27 January 1993.

[180] J. V. Simons, Jr. Heuristics in flow shop scheduling with sequence dependent setup times. *Omega-International Journal of Management Science*, 20(2):215–225, 1992.

[181] F. Sivrikaya-Şerifoğlu and G. Ulusoy. A bicriteria two-machine permutation flowshop problem. *European Journal of Operational Research*, 107(2):414–430, 1998.

[182] F. Sivrikaya-Şerifoğlu and G. Ulusoy. Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, 26:773–787, 1999.

[183] R. Spina, L. M. Galantucci, and M. Dassisti. A hybrid approach to the single line scheduling problem with multiple products and sequence-dependent time. *Computers & Industrial Engineering*, 45:573–583, 2003.

[184] J. Sridhar and C. Rajendran. Scheduling in flowshop and cellular manufacturing systems with multiple objectives: A genetic algorithmic approach. *Production Planning & Control*, 7(4):374–382, 1996.

[185] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[186] H. Stefansson, N. Shah, and P. Jensson. Multiscale planning and scheduling in the secondary pharmaceutical industry. *AIChE Journal*, 52(12):4133–4149, 2006.

[187] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, FB Informatik, TU Darmstadt, August 1998.

[188] T. Stützle. *Local search algorithms for combinatorial problems—Analysis, improvements, and new applications*. PhD thesis, Darmstadt University of Technology, 1999.

[189] P. Sundararaghavan and M. Ahmed. Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. *Naval Research Logistics Quarterly*, 31(2):325–333, 1984.

[190] R. K. Suresh and K. M. Mohanasundaram. Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives.

In *IEEE Conference on Cybernetics and Intelligent Systems (CIS), Singapore, December 1-3, 2004, Proceedings*, volume 2, pages 712–717, 2004.

[191] V. Suresh and D. Chaudhuri. Bicriteria scheduling problem for unrelated parallel-machines. *Computers & Industrial Engineering*, 30:77–82, 1996.

[192] S.Voß and D. L. Woodruff. *Introduction to computational optimization models for production planning in a supply chain*. Springer–Verlag, Berlin, 2003.

[193] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[194] C. S. Tang and E. V. Denardo. Models arising from a flexible manufacturing machine, part i: Minimization of the number of tool switches. *Operations Research*, 36:767–777, 1988.

[195] C.S. Tang. Scheduling batches on parallel machines with major and minor set-ups. *European Journal of Operational Research*, 46:28–37, 1990.

[196] V. T'kindt and J.-C. Billaut. Multicriteria schduling problems: A survey. *RAIRO Recherche opérationnelle - Operations Research*, 35(2):143–163, 2001.

[197] V. T'kindt and J.-C. Billaut. *Multicriteria scheduling: Theory, models and algorithms*. Springer, Berlin, 2002.

[198] V. T'kindt, J.-C. Billaut, and H. Houngbossa. A multi-criteria heuristic to solve a 2-stage hybrid flowshop scheduling problem. *European Journal of Automation (JESA)*, 34:1187–1200, 2000.

[199] V. T'kindt, J.-C. Billaut, S. Laurin, and O. Meslet. Un algorithme optimal polynomial pour résoundre un problème d'ordonnancement bricritère à machines parallèles. *Conference on Automation-Computers Engineering-Image-Signal (AGIS'97)*, pages 179–184, 1997.

[200] V. T'kindt, J.-C. Billaut, and C. Proust. An interactive algorithm to solve bicriteria scheduling problems on unrelated parallel machines. *European Journal of Operational Research*, 135(1):42–49, 2001.

[201] V. T'kindt, J. N. D. Gupta, and J.-C. Billaut. Two-machine flowshop scheduling with a secondary criterion. *Computers & Operations Research*, 30(4):505–526, 2003.

[202] V. T'kindt, N. Monmarche, F. Tercinet, and D. Laugt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257, 2002.

[203] B. Toktaş, M. Azizoğlu, and S. Kondakcı Köksalan. Two-machine flow shop scheduling with two criteria: Maximum earliness and makespan. *European Journal of Operational Research*, 157(2):286–295, 2004.

[204] Y. Tsujimura and M. Gen. Parts loading scheduling in a flexible forging machine using an advanced genetic algorithm. *Journal of Intelligent Manufacturing*, 12(3):413–420, 1999.

[205] F. Tu and K. R. Pattipati. Rollout strategies for sequential fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans.*, 33(1):86–99, 2003.

[206] A. Tuzikov, M. Makhaniok, and R. Manner. Bicriterion scheduling of identical processing time jobs by uniform processors. *Computers & Operations Research*, 25(1):31–35, 1998.

[207] E. Vallada, R. Ruiz, and C. Maroto. Synthetic and real benchmark for complex flow-shop problems. Technical report, Universidad Politecnica de Valencia, Spain, 2003.

[208] E. Vallada, R. Ruiz, and G. Minella. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373, 2008.

[209] P. J. M. van Laarhoven and E. H. L Aaris. *Simulated annealing: theory and application*. D. Reidel Publishing Company, Dordrech, 1987.

[210] T.K. Varadharajan and C. Rajendran. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research*, 167(3):772–795, 2005.

[211] L. Venditti, C. Meloni, and D. Pacciarelli. A tabu search algorithm for scheduling pharmaceutical packaging operations. *Proceedings of ORP^3 2007 conference, Guimarães, Portugal*, pages 107–118, 2007.

[212] S. Voß and C. Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136:285–302, 2005.

[213] J. M. Wilson. Alternative formulations of a flow-shop scheduling problem. *Journal of the Operational Research Society*, 40(4):395–399, 1989.

[214] D. Wortman. Managing capacity: getting the most from your firm's assets. *Industrial Engineering*, 24:47–49, 1992.

[215] Y. Yang, S. Kreipl, and M. Pinedo. Heuristics for minimizing total weighted tardiness in flexible flow shops. *Journal of Scheduling*, 3(2):89–108, 2000.

[216] W.-C. Yeh. A new branch-and-bound approach for the $n/2/flowshop/\alpha F + \beta C_{max}$. *Computers & Operations Research*, 26(13):1293–1310, 1999.

[217] W.-C. Yeh. An efficient branch-and-bound algorithm for the two-machine bicriteria flowshop scheduling problem. *Journal of Manufacturing Systems*, 20(2):113–123, 2001.

[218] W.-C. Yeh. A memetic algorithm for the $n/2/flowshop/\alpha F + \beta C_{max}$ scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 20(6):464–473, 2002.

[219] Y. Yi and D. W. Wang. Soft computing for scheduling with batch setup times and earliness-tardiness penalties on parallel machines. *Journal of Intelligent Manufacturing*, 14:311–322, 2003.

[220] L. Yu, H. Shih, M. E. Pfund, W. M. Carlyle, and J. W. Fowler. Scheduling of unrelated parallel-machines: An application to pwb manufacturing. *IIE Transactions on Scheduling and Logistics*, 34:921–931, 2002.

[221] Z. Zhu and R. B. Heady. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers & Industrial Engineering*, 38:297–305, 2000.

[222] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag, Berlin, Germany.

[223] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, May 2001.

[224] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[225] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE, Transactions on Evolutionary Computation*, 7(2):117–132, 2003.