UNIVERSITY OF MODENA AND REGGIO EMILIA
Department of Sciences and Methods for Engineering
Doctorate School in Industrial Innovation Engineering
XXXV Cycle

Optimization methods for knapsack and tool switching problems

DOCTORAL THESIS

Author:
Alberto LOCATELLI

Supervisor:
Prof. Manuel IORI
Co-Supervisor:
Prof. Valentina CACCHIANI

Academic year 2021/2022

UNIVERSITY OF MODENA AND REGGIO EMILIA

Abstract

Doctorate School in Industrial Innovation Engineering
Department of Sciences and Methods for Engineering

Doctor of Philosophy

Optimization methods for knapsack and tool switching problems

by Alberto LOCATELLI

This Ph.D. thesis deals with two different classes of optimization problems: knapsack problems and tool switching problems.
The first purpose of this thesis is to provide a comprehensive survey on the knapsack problems to cover the developments appeared in this field after the publication of the latter volume of the two seminal books by Martello and Toth (1990) and Kellerer, Pferschy, and Pisinger (2004). We review over 450 different papers, mostly appeared after 2004 and before Summer 2021.
In the subsequent part of the thesis, motivated by a real-world application in the colour printing industry, we deal with different variants of the well-known Tool Switching Problem (ToSP). Firstly, we introduce four different variants of ToSP. For each variant, we discuss its complexity and propose a mathematical formulation. The third and fourth variants introduce a novel requirement into ToSP: the tool order constraint. We show that the new problem variants are NP-hard even when the job sequence is given as part of the input and the setup times are binary. We solve them by using dedicated arc flow models, whose effectiveness is evaluated on several instances that are generated with the aim of covering different scenarios of interest. The thesis continues by addressing a challenging real-world industrial problem arising in a food packaging company located in the city of Reggio Emilia (Italy). The real-world problem generalizes the fourth introduced variant of ToSP to a great extent (as it includes parallel heterogeneous machines, due dates, and several additional complicating features). To solve it, we developed a greedy randomized adaptive search procedure equipped with several local search procedures. The excellent performance of the algorithm is proved by extensive computational experiments on real-world instances, for which it produced good-quality solutions within a limited computing time. As in the real problem the setup times are subject to significant uncertainties, forecasting these times is a very difficult task and significantly influences the quality and efficiency of job scheduling. Thus, we explore the use of machine learning regression algorithms for setup time prediction using a real-world industrial dataset. The experimental results demonstrate these approaches outperform by a large margin the evaluation methods available in the literature, proving their effectiveness in modeling the application.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Knapsack problems are a highly active research area in combinatorial optimization. Although, the knapsack problem has been known since over a century, see Mathews [368], the first algorithmic studies were published in the Fifties (by Dantzig [141] and Bellman [48]) while an intense research activity started in the Sixties. It produced, over the next fifty years, an impressive number of scientific results, making this field a very relevant area of combinatorial optimization. The success of this topic in the subsequent decades is also shown by the fact that a study by Skiena [455] ranked it as the 18th most popular algorithmic problem, and the second among the $\mathcal{NP}$-hard problems (after traveling salesman). Two specific monographs have been devoted to knapsack problems. The first book was published in 1990 by Martello and Toth ([361]). It focuses on algorithms and computer implementations for knapsack problems, and offers a detailed description and analysis of approximately 200 results that were published in the previous thirty years. The second book, published in 2004 by Kellerer, Pferschy, and Pisinger ([297]), is specifically dedicated to this area. The first purpose of this thesis is to cover the developments appeared in this field after the publication of the latter volume, until Summer 2021.

Chapter 2 is devoted to problems whose goal is to optimally assign items to a single knapsack. Besides the classical knapsack problems (binary, subset sum, bounded, unbounded, change-making), we review problems with special constraints (setups, multiple-choice, conflicts, precedence, sharing, compartments) as well as relatively recent fields of investigation, like robust and bilevel problems.

Chapter 3 covers multiple, multidimensional (vector and geometric), and quadratic knapsack problems, as well as other relevant variants, such as, e.g., multiobjective and online versions. Chapters 2 and 3 list over 450 different papers, mostly appeared after 2004, the publication year of the latter of the two classical books specifically dedicated to these topics.

In the subsequent part of the thesis, motivated by a real-world application in the colour printing industry, we deal with different variants of the well-known Tool Switching Problem (ToSP). The ToSP consists in optimally sequencing jobs and in assigning tools to a capacitated magazine in order to minimize the number of tool switches. This is equivalent to considering unit-time setups. To the best of our knowledge, most of the previous research on ToSPs concerned uniform switching time (see, e.g, Calmels [73]) and the only works that take into consideration non-uniform setup times are those by Privault and Finke [409], Windras Mara et al. [482], and Iori et al. [278].

In the ToSP, the position of the tools in the magazine is irrelevant (see Laporte, Salazar-González, and Semet [329]) and this information can thus be neglected. On the other hand, this information is crucial in the ToSP with non-uniform setup times,

since the setup time depends on the tools switched at the same magazine slot. Another problem variant of the ToSP with unit-time setups, but in which the information about the position of the tools in a magazine cannot be neglected, arises in the case in which tools may require more than one slot. To tackle this problem, Tzur and Altman [467] proposed an *Integer Linear Programming* (ILP) formulation and developed a heuristic procedure. Later, Van Hop [469] proposed a construction heuristic, while Crama et al. [126] proved that the problem is NP-hard even when the order in which the jobs are processed is fixed.

In Chapter 4, we address four different variants of ToSP of increasing difficulty: CUF-ToSP, GUF-ToSP, GOF-ToSP, and GOV-ToSP. For each problem, we discuss its complexity and propose a mathematical programming model. The third and fourth variants introduce a novel requirement into ToSP: the tool order constraint. Under this requirement, during the processing of each job, the selected tools must be sorted along the slot sequence in the machine, and the machine will use them for processing the job applying the tools in that order. We show that the new problem variants are NP-hard even when the job sequence is given as part of the input and the setup times are binary. We solve them by using dedicated arc flow models. We evaluate the effectiveness of the models on several instances that are generated with the aim of covering different scenarios of interest.

In Chapter 5, we tackle an interesting real-world industrial problem arising in a food packaging company located in the city of Reggio Emilia (Italy). The problem faced by the company generalizes the last variant of ToSP, tackled in the fourth chapter, to a great extent (as it includes parallel heterogeneous machines, release and due dates, and several additional complicating features). In particular, we address a scheduling problem that consists in assigning printing jobs to a heterogeneous set of parallel flexographic printer machines, with the aim of minimizing a weighted sum of total weighted tardiness and total setup time. To solve it, we developed a greedy randomized adaptive search procedure equipped with several local search procedures. The excellent performance of the algorithm is proved by extensive computational experiments on real-world instances, for which it produced good-quality solutions within a limited computing time.

As in the real scheduling problem the setup times are subject to significant uncertainties, forecasting these times is a very difficult task and significantly influences the quality and efficiency of scheduling. Although shortcomings in the setup times prediction may lead to develop inefficient schedules and represent a relevant source of gap between scheduling theory and practice, setup times estimation complexities received limited attention in the literature. With the aim of filling this gap, in the last chapter, we explore the use of machine learning regression algorithms for setup time prediction and we apply them to the real-world scheduling application faced in Chapter 5. Using a real-world industrial dataset, we train three different machine learning models: linear regression, random forests, and gradient boosting machines. We take into account a wide set of features and several possible interdependencies among them, and then, using a feature selection method based on feature importance scores generated by random forests, we identify a parsimonious but comprehensive subset of these features. The experimental results demonstrate that the gradient boosting machine approach obtains the best performance overall, immediately followed by random forests. For both models, the mean squared error on the predicted setup times is less than half of that of the heuristic evaluation method available in the literature. The accuracy of the obtained predictions shows the effectiveness of the proposed approach. Moreover, the versatility of the machine

learning models allows an easy application to similar evaluation tasks in a multitude of scheduling scenarios, making the obtained results particularly significant and valuable.

Chapter 2

Knapsack problems - An Overview of Recent Advances. Part I: Single Knapsack Problems*

After the seminal books by Martello and Toth (1990) and Kellerer, Pferschy, and Pisinger (2004), knapsack problems became a classical and rich research area in combinatorial optimization. The purpose of the first two chapters of this thesis is to cover the developments that appeared in this field after the publication of the latter volume. This chapter is devoted to problems whose goal is to optimally assign items to a single knapsack. Besides the classical knapsack problems (binary, subset sum, bounded, unbounded, change-making), we review problems with special constraints (setups, multiple-choice, conflicts, precedences, sharing, compartments) as well as relatively recent fields of investigation, like robust and bilevel problems. Chapter 3 covers multiple, multidimensional, and quadratic knapsack problems, and includes a succinct treatment of online and multiobjective knapsack problems.

2.1 Introduction

The area of knapsack problems is one of the most active research areas of combinatorial optimization. Two specific monographs have been dedicated to this field. In 1990, Martello and Toth published the first book ([361], now available online at `http://www.or.deis.unibo.it/knapsack.html`) explicitly dedicated to algorithms and computer implementations for knapsack problems, in which about 200 results that appeared in the previous thirty years were thoroughly described and commented. In 2004, Kellerer, Pferschy, and Pisinger published the second book ([297]) specifically dedicated to this area. Quoting their introduction, *"Thirteen years have passed since the seminal book on knapsack problems by Martello and Toth appeared. On this occasion a former colleague exclaimed back in 1990: "How can you write 250 pages on the knapsack problem?"... However, in the last decade a large number of research publications contributed new results for the knapsack problem in all areas of interest such as exact algorithms, heuristics and approximation schemes."* Indeed, this new volume included about 500 bibliographic references, two thirds of which appeared after 1990. Seventeen more years have passed, during which the intense research activity on these topics has continued. The purpose of the first two chapters of this thesis is thus to report the many results that appeared in this period. As all the basic algorithmic approaches have been fully described in the two monographs, they will not be repeated

*The results of this chapter appears in: V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. "Knapsack problems - an overview of recent advances. Part I: Single knapsack problems". In: *Computers & Operations Research* 143 (2022), 105692.

here. We will concentrate instead on a succinct description of the main recent results appeared after 2003 (and until Summer 2021), with the objective of completing the overview of this topic.

The knapsack problem has been known since over a century, see Mathews [368], and, according to folklore, the name was suggested by Tobias Dantzig (1884-1956), father of George Dantzig. (According to some authors, the suggestion was included in Dantzig [142], which is wrong: the term 'knapsack' does not appear in this book.) While the first algorithmic studies were published in the Fifties (by Dantzig [141] and Bellman [48]), an intense research activity started in the Sixties. The success of this topic in the subsequent decades is also shown by the fact that a study by Skiena [455] ranked it as the 18th most popular algorithmic problem, and the second among the $\mathcal{NP}$-hard problems (after traveling salesman).

The ancestor problem is known as the *0-1 Knapsack Problem* (KP01). Using Dantzig's 1957 words, *"In this problem a person is planning a hike and has decided not to carry more than 70 lb of different items, such as bed roll, geiger counters (these days), cans of food ...."*. Formally, we are given a *capacity* c and a set of n items, each with a *weight* w_j and a *profit* p_j. We want to determine a subset of items such that its total weight does not exceed the capacity and its total profit is a maximum. The problem can then be formulated as the *Integer Linear Programming* (ILP) model:

$$\max \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq c \tag{2.2}$$

$$x_j \in \{0,1\} \qquad (j = 1,\ldots,n), \tag{2.3}$$

where x_j takes the value 1 if and only if item j is selected. The first study on the KP01, see [141], concerned the *Linear Programming* (LP) relaxation of this model, in which (2.3) is replaced by

$$0 \leq x_j \leq 1 \qquad (j = 1,\ldots,n).$$

In the following, we will generally assume, without loss of generality, that all input values are positive, that $w_j \leq c$ $(j = 1,\ldots,n)$, and that $\sum_{j=1}^{n} w_j > c$.

The number of knapsack problem variants addressed in recent years is huge. In this chapter, we mostly concentrate on those classical problems that were addressed in the main chapters of Kellerer, Pferschy, and Pisinger [297] (which also correspond to chapters in Martello and Toth [361]) and on variants that received significant attention in the recent literature.

Most results on knapsack problems were developed by the combinatorial optimization community, to which the authors of monographs [361] and [297] dedicated to these problems belong. This survey is mainly aimed at this community, and hence our choice of subjects privileges problems with a clear combinatorial aspect. The stochastic knapsack problems are not included, while we address in a succinct way knapsack problems with a clear continuous, non-linear flavor. Problems closer to the computer science community (*online* knapsack problems) or belonging to the area of multiple criteria decision aiding (*multi-objective* knapsack problems) are discussed in Section 3.6.2. Polyhedral aspects of knapsack problems are briefly mentioned in the text but not covered in a dedicated section: in Section 2.2 we provide pointers to a recent survey and to successive updates.

The decision version of a special case of the KP01 (similar to the *subset sum problem* treated in Section 2.4) is one of the famous Karp's 21 $\mathcal{NP}$-complete problems (see Karp [291]). In general, all optimization problems considered in this survey are $\mathcal{NP}$-hard. The "simplest" single knapsack problems (basically those reviewed in Sections 2.3-2.8) are $\mathcal{NP}$-hard in the weak sense, i.e., they may be solved in pseudo-polynomial time through *Dynamic Programming* (DP). Most variants and generalizations considered in the subsequent sections, as well as most problems treated in Chapter 3, are instead $\mathcal{NP}$-hard in the strong sense (i.e., they cannot be solved by pseudo-polynomial time algorithms unless $\mathcal{P} = \mathcal{NP}$), as specifically stated in the corresponding sections. $\mathcal{NP}$-completeness considerations and several detailed complexity proofs for knapsack problems can be found in Chapter 1 of [361] and in Appendix A of [297].

Monograph [297] is updated to the end of June 2003. Apart from a handful of earlier publications (mostly in the present and in the next section), all references in this survey are from 2003 onwards. With very few exceptions, we restricted our review to contributions appeared in peer reviewed journals. The survey is subdivided into two parts. This chapter covers the classical single knapsack problems, while Chapter 3 is devoted to the multiple, multidimensional, and quadratic cases, and to a succinct treatment of online and multiobjective knapsack problems. As we review several variants and generalizations, the thesis includes many acronyms: we provide the list of the used abbreviations in Appendix 6.6.

2.2 Books and surveys

We already mentioned the two monographs dedicated to knapsack problems. Besides the KP01, the book by Martello and Toth [361] includes specific chapters on the following basic variants: *bounded* (and *unbounded*) *knapsack*, *subset-sum*, *change-making*, *multiple knapsack*. In addition, there are two chapters on two companion problems: generalized assignment and bin packing. The book by Kellerer, Pferschy, and Pisinger [297] includes chapters on the same basic variants (but change-making), plus chapters on *multidimensional*, *multiple-choice*, and *quadratic knapsack problems*.

Monograph [297] only briefly mentions bin packing and generalized assignment, which are instead extensively treated in monograph [361]. In the *Bin Packing Problem* (BPP), one is given n items with weights w_j $(j = 1, \dots, n)$ and an unlimited number of identical knapsacks (*bins*) of capacity c, and the objective is to pack *all* the items into the minimum number of bins without exceeding their capacities. In the *Generalized Assignment Problem* (GAP), one is given n items and m knapsacks of capacity c_i $(i = 1, \dots, m)$: inserting item j into knapsack i $(i = 1, \dots, m; j = 1, \dots, n)$ produces a profit p_{ij} and assigns a weight w_{ij} to knapsack i. The objective is to pack *each* item into exactly one knapsack so as to maximize the overall profit without assigning to any knapsack a total weight greater than its capacity. In the last decades, these two topics where subject to intense investigation, which made them autonomous areas of research. For this reason, they will not be treated in the present survey. We refer the reader to some studies specifically devoted to them, namely:

- for the BPP, a review of exact approaches has recently been presented by Delorme, Iori, and Martello [160], while an exhaustive treatment of approximation algorithms can be found in Coffman et al. [119];
- for the GAP, three surveys have been published in the last fifteen years, by Morales [380], Öncan [389], and Wu, Mutsunori, and Toshihide [484].

Coming to knapsack problems, a number of surveys devoted to specific methodologies or problem variants appeared in the last two decades:

- Pisinger [404] presented an overview of exact solution approaches for the KP01 using classical and new benchmark tests. Also see the recent article by Smith-Miles, Christiansen, and Muñoz [456] for a revisiting of this study;
- Bretthauer and Shetty [66] presented an exhaustive review of algorithms for various classes of *nonlinear knapsack problems* (see Section 3.5.1): continuous, integer, convex, nonconvex, separable, and nonseparable. In addition, they discussed some interesting applications arising in production planning, health care, and computer networks. An updated survey was later presented by Li and Sun [337]. We also refer the reader to Ibaraki and Katoh [272] and Lin [343] for previous surveys;
- Fréville [184] provided a survey on the *multidimensional knapsack problem* (see Section 3.3) reviewing exact, heuristic, approximation, and metaheuristic algorithms, as well as commercial software products; the work was later extended in Fréville and Hanafi [185];
- Pisinger [405] reviewed the literature on the *quadratic knapsack problem* (see Section 3.5), with special emphasis on methods for computing upper bounds. The study includes an extensive experimental analysis, comparing tightness and computational effort of the various bounds;
- Wilbaut, Hanafi, and Salhi [480] discussed heuristic algorithms for a number of knapsack problem variants;
- Hu, Landa, and Shing [267] reviewed exact and approximation approaches to the *unbounded knapsack problem* (see Section 2.5.2). Recently, Becker and Buriol [42] reported the results of extensive computational experiments on several exact algorithms from the literature for this problem;
- Lust and Teghem [349] considered the *multiobjective* version of single and multidimensional knapsack problems, reviewing exact, approximation, heuristic and metaheuristic algorithms (see Section 3.6.2);
- Kellerer and Strusevich [299] reviewed the main results on the *symmetric quadratic knapsack problem* (see Section 3.5.2), focusing on approximation algorithms and their application to scheduling problems;
- Laabadi et al. [320] reviewed heuristic algorithms for variants of the *multidimensional knapsack problem* (see Section 3.3);
- recently (2019), Hojny et al. [261] provided a comprehensive overview of theoretical results on the polytopes of knapsack problems, to which the reader is referred for a deep analysis of these topics. After the publication of [261], relevant studies on the knapsack polytopes have been presented by Hojny [260] (polynomial-size formulations), Letchford and Souli [333] (knapsack cover inequalities), and Bienstock et al. [51] (polytope of the minimization version of the problem);
- the *geometric knapsack problem* (see Section 3.4) has been treated in a number of recent surveys, namely, Christensen et al. [114] (mainly devoted to the BPP),

Silva, Toffolo, and Wauters [451] (focused on exact methods for the three-dimensional version of the problem), Leao et al. [331] (devoted to the case of irregular shapes), Iori et al. [277] (focused on exact algorithms and mathematical models for the two-dimensional version of the problem).

We finally mention the special issue of *Computers & Operations Research* on knapsack problems and applications, edited by Hifi and M'Hallah [247].

2.3 0-1 knapsack problem

The KP01 is the most popular among knapsack problems and it has been the subject of intense research for decades. These investigations have produced a rich variety of theoretical, practical, and algorithmic results which have, to a certain extent, saturated this specific field. The most widely used approach to the exact solution of the problem is still the `Combo` algorithm, developed by Martello, Pisinger, and Toth [359], whose `C` code is available at `http://hjemmesider.diku.dk/~pisinger/codes.html`.

New branching strategies for *Branch-and-Bound* (B&B) approaches were developed by Morales and Martínez [379] and Yang, Boland, and Savelsbergh [493]. The sensitivity analysis to perturbations of item profits or weights was studied by Hifi, Mhalla, and Sadfi [245, 246] and by Belgacem and Hifi [47], while Pisinger and Saidi [406] investigated a particular sensitivity analysis (*tolerance analysis*) that can be performed in amortized time $O(c \log n)$ for each item. Improvements over existing *Fully Polynomial Time Approximation Schemes* (FPTAS) were recently developed by Chan [97] and by Jin [285].

In the next sections, we review recent results on relevant variants of the basic KP01.

2.4 Subset sum problem

When the profit and the weight of each item are identical, the problem, given by

$$\max \sum_{j=1}^{n} w_j x_j \qquad (2.4)$$
$$\text{s.t. } (2.2) - (2.3),$$

is denoted as the *Subset Sum Problem* (SSP). In the special case in which $c = \sum_{j=1}^{n} w_j/2$, the SSP is called the *Partition Problem* and is regarded as the "simplest" $\mathcal{NP}$-hard problem (see Garey and Johnson [199]). Differently from the KP01, the SSP has been the subject of intensive research in recent years too, probably also due to its connection to cryptography (see, e.g., Kate and Goldberg [294]).

Exact solution. The B&B algorithms for the exact solution of the SSP are normally initialized by sorting the items according to decreasing weight. Kolpakov and Posypkin [307] showed that this policy is the one requiring, in the worst case, the smallest number of iterations. Kolpakov, Posypkin, and Sin [308] proposed a variation of the B&B method that decreases the number of iterations by a factor of two. Curtis and Sanches [132] presented an improved version of a DP algorithm called `Balsub` (see [297], Section 4.1.5), and computationally compared it with other approaches from the literature on benchmarks of difficult SSP instances.

Approximation. The classical greedy algorithm for the SSP has worst-case performance ratio equal to 0.5. Using a multiple-pass variant of the algorithm, Martello

and Toth improved it to 0.75 (see [297], Section 4.5). More recently, Ye and Borodin [496] studied greedy variants in which decisions may be revoked, obtaining a performance ratio between 0.8 and $\delta \approx 0.893$. Gál et al. [193] adopted an unusual computation model (*input stream*) to obtain an FPTAS having space requirement $O(1/\varepsilon)$ (while that of previous FPTASs depends on n), where ε is the required accuracy. Pseudopolynomial-time algorithms for the SSP were presented at computer science conferences by Koiliaris and Xu [306] and by Bringmann [67].

Heuristics. Since most instances of the SSP can be exactly solved in very short computing times (see [297, 361]), no relevant results have been recently achieved by heuristic or metaheuristic algorithms. Ghosh, Chakravarti, and Sierksma [201] presented a sensitivity analysis of greedy heuristics for the KP01 and the SSP.

Variants and generalizations

In recent years, a number of variants of the SSP has been considered. Rohlfshagen and Yao [421] empirically analyzed the *dynamic* version of the SSP (in which the parameters change over time), investigating the correlation between the parameter change and the movement of the optimum.

Darmann et al. [145] studied a game theoretic variant (the *Subset Sum Game*), in which two decision makers compete for a common resource (the capacity), showed that finding an optimal sequence of decisions is an $\mathcal{NP}$-hard problem, and analyzed the worst-case performance of two natural heuristic strategies. Another game-theoretic issue (the *Fair Subset Sum Problem*) was studied by Nicosia, Pacifici, and Pferschy [386].

Kothari, Suri, and Zhou [311] introduced the *Interval Subset Sum Problem*: given a set of intervals and an integer target T, find a set of integers, at most one from each interval, such that their sum is closest to, without exceeding, T. They defined an efficient FPTAS for its approximate solution. Later, Diao, Liu, and Dai [164] proposed an improved FPTAS having almost the same time complexity but a significantly lower space complexity. Gourvès, Monnot, and Tlilane [212] considered a node-weighted digraph in which one has to select a subset of vertices with total weight not exceeding a given capacity and added additional precedence and maximality constraints, whose combination leads to four problem variants: they proved that all problems are $\mathcal{NP}$-hard and gave approximation results for special classes of digraphs.

We conclude this section by observing that the simple and neat structure of the SSP also attracted researchers outside the operations research community. In mathematics, properties of the SSP over finite fields were investigated by Li and Wan [338], Wang and Nguyen [477], and Choe and Choe [113]. The computer science community developed parallel algorithms for different architectures (Curtis and Sanches [131] and Sanches, Soma, and Yanasse [433, 434]).

2.5 Knapsack problems with item types

In this section, we examine variants of the KP01 in which a number of identical copies of each item is available. In these contexts, the term 'item' is normally replaced by *item type*.

2.5.1 Bounded knapsack problem

The generalization of the KP01 in which b_j identical copies of item type j are available ($j = 1, \ldots, n$) is known as the *Bounded Knapsack Problem* (BKP), formally defined by

$$\begin{aligned} &(2.1) - (2.2) \\ &0 \leq x_j \leq b_j,\ x_j \text{ integer} \qquad\qquad (j = 1, \ldots, n), \end{aligned} \tag{2.5}$$

where x_j represents the number of selected copies of item type j.

We are only aware of two recent results on the BKP. Tamir [459] proposed a pseudo-polynomial algorithm having time complexity $O(n^3 \max_j\{w_j\}^2)$, to be compared with the $O(nc)$ algorithm by Kellerer et al. (see [297], Section 7.2.2). Deineko and Woeginger [148] showed that all instances satisfying a set of special inequalities that relate weight ratios to profit ratios (*cross ratio ordered instances*) can be solved in $O(n)$ time.

Most results appeared in recent years concern the following special case of the BKP:

2.5.2 Unbounded knapsack problem

In this case, an unlimited number of copies of each item type are available. The *Unbounded Knapsack Problem* (UKP) is defined by

$$\begin{aligned} &(2.1) - (2.2) \\ &x_j \geq 0 \text{ and integer} \qquad\qquad (j = 1, \ldots, n). \end{aligned} \tag{2.6}$$

The UKP has a well-known characteristic: most of the contribution to the optimal solution profit comes from the item type, say s, with highest profit-to-weight ratio, provided the knapsack capacity is sufficiently large (see Hu, Landa, and Shing [267]). In particular, a number of studies has been devoted to finding, for a given set of item types, the capacity bound c_0 such that, for all $c \geq c_0$, the optimal solution value is $z^*(c) = z^*(c - w_s) + p_s$ (*periodicity property*; see [297], Section 8.2, for a more detailed treatment.) The survey by Hu, Landa, and Shing [267] (see Section 2.2) also includes a DP approach for evaluating the periodicity property. Improved periodicity bounds were proposed by Huang, Lawley, and Morin [268] and Huang and Tang [269], although the latter, which requires $O(n^2)$ time, can be time-consuming when $c < n$.

Poirriez, Yanev, and Andonov [408] proposed an algorithm based on a combination of DP, dominance rules, and B&B for the exact solution of the UKP. A hybrid approach, that also includes the generation of valid inequalities, was presented by He, Hartman, and Pardalos [236], who also extended it to the multidimensional version of the problem (see Section 3.3). Becker and Buriol [42] reported on an extensive computational experimentation (on classical and new benchmarks) of seven old and recent exact algorithms for the UKP, including the algorithm in [408] and commercial solvers CPLEX and Gurobi: quite surprisingly, a DP algorithm developed in 1966 by Gilmore and Gomory [203] (slightly improved by the authors) achieved the best results among all DP algorithms.

Jansen and Kraft [283] proposed an FPTAS for the UKP, whose time and space complexities improve those of classical FPTASs from the literature (see [297], Section 8.5). Deineko and Woeginger [149] investigated a special case of the UKP in which the item weights form an arithmetic sequence and proposed an exact $O(n^8)$ time algorithm.

2.5.3 Change-making problems

The name of this problem comes from an interpretation in which a cashier needs to make change for a certain sum using the minimum number of coins from a given set of coin denominations. Using the knapsack terminology, given n (different) item types, each having weight w_j ($j = 1, \ldots, n$), and a knapsack of capacity c, the *Unbounded Change-Making Problem* (UCMP) is:

$$\min \sum_{j=1}^{n} x_j \tag{2.7}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j = c \tag{2.8}$$

$$x_j \geq 0 \text{ and integer} \qquad (j = 1, \ldots, n). \tag{2.9}$$

It is normally assumed that $w_1 < w_2 < \cdots < w_n$ and that $w_1 = 1$ (so a solution to the problem always exists). The case in which there is a limited number of items of each type is known as the *Bounded Change-Making Problem* (BCMP).

A greedy algorithm for the UCMP iteratively selects the item type (coin) whose weight is no larger than the remaining capacity. Most recent results for the UCMP studied characterizations of coin systems for which the greedy algorithm is optimal. Pearson [394] gave an $O(n^3)$ algorithm to determine, for a given coin system, whether the greedy algorithm is optimal. Cowen, Cowen, and Steinberg [123] characterized *totally greedy* coin sets, i.e., instances of the UCMP such that, for each j, the greedy algorithm is optimal for the coin set $\{w_1 = 1, w_2, \ldots, w_j\}$. Adamaszek and Adamaszek [3] provided necessary conditions that must be satisfied by a coin system in order to ensure that the greedy algorithm is optimal. Goebbels et al. [206] studied approximate solutions for a generalization of the problem in which the sum of the weights may be larger than c.

2.6 Knapsack problems with setup

Although the first definition of setup knapsack problems dates back to the Nineties (see Chajakis and Guignard [96]) these problems have attracted more attention in the last decade. A number of different (although similar) definitions can be encountered in the literature, the most frequent one being the following. Consider a generalization of the KP01 in which items belong to F disjoint families and can be selected only if the corresponding family is activated. Family $i \in \{1, \ldots, F\}$ contains n_i items and has positive activation (setup) cost f_i and weight d_i. The profit and weight of an item j of family i are p_{ij} and w_{ij}, respectively. The *Knapsack Problem with Setup* (KPS) is then

$$\max \sum_{i=1}^{F} \sum_{j=1}^{n_i} (p_{ij} x_{ij} - f_i y_i) \tag{2.10}$$

$$\text{s.t.} \sum_{i=1}^{F} \sum_{j=1}^{n_i} (w_{ij} x_{ij} + d_i y_i) \leq c \tag{2.11}$$

$$x_{ij} \leq y_i \qquad (i = 1, \ldots, F; j = 1, \ldots, n_i) \tag{2.12}$$

$$x_{ij} \in \{0, 1\} \qquad (i = 1, \ldots, F; j = 1, \ldots, n_i) \tag{2.13}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, F), \tag{2.14}$$

where x_{ij} takes the value one iff item j of family i is selected and y_i takes the value one iff family i is activated. Constraints (2.12) impose the activation of a family if one of its items is selected. The KPS has a number of real world applications, e.g., in make-to-order production contexts in the management of different product categories.

Exact solution. The first exact approach to the KPS was proposed by Yang and Bulfin [494], who presented a B&B algorithm and experimentally tested its performance. Ichimura, Yokoyama, and Magori [273] proposed an extension of such method, especially aimed at solving large-scale instances. Chebil and Khemakhem [98] developed a DP approach with an original space reduction technique. Della Croce, Salassa, and Scatamacchia [156] presented an exact method based on the identification of sub-problems, tackled through a commercial solver (CPLEX). Furini, Monaci, and Traversi [191] studied alternative ILP formulations and implemented a parallel approach that executes, on a multi-thread computer, three algorithms (a B&B, a B&P, and a DP procedure) in parallel, and halts execution as soon as one of the three terminates. Although a precise computational comparison with other approaches from the literature is not immediate, this method appears to be the current state-of-the-art for the KPS.

Approximation. Pferschy and Scatamacchia [397] proved that no polynomial time approximation algorithm can exist for the KPS (unless $\mathcal{P} = \mathcal{NP}$) and analyzed three special cases of the problem that admit an FPTAS. They also proposed and computationally evaluated an improved DP algorithm for its exact solution.

Heuristics. Most exact algorithms solve large instances of the KPS within very short computing times. For example, the computational experiments in [191] refer to instances with up to 30 families and 10 000 items. Few heuristic approaches were also proposed: Khemakhem and Chebil [301] presented a method based on a truncated tree search approach and compared it with CPLEX on instances of the same size. Amiri [27] proposed an iterative approach based on a Lagrangian relaxation of the problem, followed by a greedy-type heuristic guided by the current Lagrangian multipliers to construct feasible solutions. The computational experiments in [27] show that the method produces good quality solutions for very large instances with up to 500 families and 2 000 000 items.

Variants and generalizations

A number of variants of the KPS can be found in the literature. Altay, Robinson Jr., and Bretthauer [24] considered a mixed-integer variant of the KPS where fractions of items are allowed to be selected. They developed exact and heuristic solution methods as well as a Benders decomposition approach for the continuous relaxation of the problem.

Akinc [12] studied the *fixed-charge knapsack problem*, a special case of the KPS with no setup capacity consumption, i.e., $d_i = 0$ $(i = 1, \ldots, F)$. He developed several algorithmic components to improve the efficiency of B&B, such as efficient procedures to obtain good candidate solutions and a set of rules to peg the set-up variables y_i to 1 or 0.

McLay and Jacobson [372] considered a generalization of the BKP (see Section 2.5.1) where each item type has a set-up weight and a (positive) set-up value that are activated if at least one copy of that item type is selected. They proposed three DP algorithms and an FPTAS. Al-Maliky, Hifi, and Mhalla [14] developed a sensitivity analysis of this problem to the perturbation of item profits or weights.

McLay and Jacobson [371] studied additional variants, one involving an unbounded number of copies of each item type, and one imposing that exactly k items are inserted into the knapsack. For each of these two variants, they proposed DP approaches, heuristics, and an FPTAS. Another unbounded variant of the problem considered in [372] was studied by Caserta, Rico, and Márquez Uribe [90], who proposed a metaheuristic algorithm based on a "cross entropy" scheme.

Michel, Perrot, and Vanderbeck [373] defined the *multiple-class integer knapsack problem with setups*, a variant of the KPS where there are multiple copies of each item, the item weights are multiples of a value associated with the class, and each class has an upper and a lower bound on the number of items to select. They studied this problem, as well as a number of its variants, showing how specialized B&B procedures derived from the classical Horowitz-Sahni algorithm for the KP01 (see [361], Section 2.5.1) can be extended to deal with them.

2.7 Multiple-choice knapsack problem

The *Multiple-Choice Knapsack Problem* (MCKP), also known as the *knapsack problem with generalized upper bound constraints*, is a generalization of the KP01 in which the item set is partitioned into ℓ *classes* $N_1, \ldots, N_\ell$ and it is requested to select exactly one item per class. Formally,

$$\begin{aligned} &(2.1) - (2.3) \\ &\sum_{j \in N_i} x_j = 1 \qquad\qquad (i = 1, \ldots, \ell). \end{aligned} \tag{2.15}$$

The problem is sometimes modeled with the '$\leq$' sign in (2.15). Such formulation can be transformed in an equivalent MCKP formulation by adding a dummy item, with null profit and weight, to each class. To the best of our knowledge, no relevant result on the exact solution of the MCKP appeared after the publication of monograph [297].

He et al. [235] presented an approximation algorithm that, for a prefixed positive integer parameter t, guarantees a worst-case ratio of $3 + (\frac{1}{2})^t$ and runs in $O(n(t + \log m))$ time. Bednarczuk, Miroforidis, and Pyzel [43] presented a heuristic approach that removes the capacity constraint (2.2) and solves a bi-objective problem that maximizes the total profit and minimizes the total weight: the corresponding solution set is then searched for Pareto efficient solutions which are feasible for the MCKP. Sbihi [438] developed a reactive Tabu search algorithm for a variant of the MCKP arising in budget planning over discrete periods espond to classes and have individual capacities.

Agra and Requejo [5] studied a special case of the MCKP (the *linking set problem*) which, under certain conditions, can be solved exactly in polynomial time. Zhong and Young [504] described a real-world case in which the decision on the allocation of funds to alternative projects was solved through an MCKP model.

Kozanidis and Melachrinoudis [315] and Kozanidis, Melachrinoudis, and Solomon [316] discussed continuous and mixed-integer knapsack problems that include multiple-choice type constraints imposing, for each class, an upper bound on the sum of the corresponding continuous variables. They proved that the former problem can be exactly solved by a two-phase greedy algorithm and developed a B&B approach for the exact solution of the latter.

2.8 Knapsack sharing problem

Another generalization of the KP01 is the *Knapsack Sharing Problem* (KSP). As in the MCKP, the item set is partitioned into ℓ *classes* $N_1, \ldots, N_\ell$ but the objective is to select a set of items that maximizes the minimum total profit of a class. Formally,

$$\max \min_{1 \leq i \leq \ell} \left\{ \sum_{j \in N_i} p_j x_j \right\} \tag{2.16}$$
$$\text{s.t. } (2.2) - (2.3).$$

Hifi, M'Halla, and Sadfi [244] proposed an exact algorithm based on a decomposition of the problem into ℓ KP01s, which are solved through DP for different tentative values of the capacity assigned to each subproblem. Similar decompositions and a sensitivity analysis have been studied by Hifi et al. [46, 243, 253, 255].

Haddar et al. [217] proposed a hybrid heuristic based on the combination of an LP-based heuristic and a metaheuristic approach, and evaluated its average performance through extensive computational experiments.

In a relevant variant of the problem, the *Generalized Knapsack Sharing Problem* (GKSP) (also referred to as the *knapsack sharing problem with common items*), there is an additional class N_0 and its profit is summed to the profit of each class N_i ($i = 1, \ldots, \ell$). Fujimoto and Yamada [187], who first defined this problem, proposed an exact algorithm based on the decomposition of the problem into a KP01 and a KSP and the enumeration of the possible capacities of the two subproblems. The method was improved by Dahmani, Hifi, and Wu [135] through upper bound computations and reduction procedures. A metaheuristic approach for the approximate solution of the GKSP was proposed by Haddar et al. [219].

2.9 Knapsack problem with conflict graph

The *Knapsack Problem with Conflict Graph* (KPCG), also referred to as the *knapsack problem with conflicts* or the *disjunctively constrained knapsack problem*, is a generalization of the KP01 in which a given undirected graph $G = (V, E)$ defines the pairs of incompatible items that cannot be simultaneously selected. Formally,

$$(2.1) - (2.3)$$
$$x_i + x_j \leq 1 \qquad (i, j) \in E. \tag{2.17}$$

An alternative formulation replaces constraints (2.17) with

$$\sum_{j \in V(i)} x_j \leq |V(i)| (1 - x_i) \qquad i \in V, \tag{2.18}$$

where $V(i)$ denotes the set of neighboring vertices of vertex $i \in V$. To possibly obtain a stronger LP-relaxation bound, constraints (2.17) can also be replaced by

$$\sum_{j \in C} x_j \leq 1 \qquad C \in \mathcal{C}, \tag{2.19}$$

where $\mathcal{C}$ denotes the family of cliques of G such that, for each edge $(i, j) \in E$, items i and j belong to some clique $C \in \mathcal{C}$.

The KPCG is a generalization of the *stable set problem* (given an undirected graph, find a maximal set of vertices no two of which are connected by an edge) and hence it is strongly $\mathcal{NP}$-hard, see Pferschy and Schauer [398].

Exact solution. Hifi and Michrafy [249] proposed a three-phase algorithm: a reactive local search algorithm, adapted from Hifi and Michrafy [248], computes a lower bound, reduction strategies are applied to fix some decision variables, and the reduced problem is solved by the commercial solver CPLEX. The algorithm was tested on instances with very sparse conflict graphs. Bettinelli, Cacchiani, and Malaguti [50] proposed a B&B algorithm based on model (2.1)-(2.3), (2.19): the branching phase makes use of DP for pruning decision nodes, and upper bounds are computed as an extension of the *weighted clique cover bound*, proposed in Held, Cook, and Sewell [238]. The algorithm was extensively tested on instances with conflict graph densities between 0.1 and 0.9, exhibiting better performance than the B&B approach proposed in Sadykov and Vanderbeck [429], in which KPCG arises as a subproblem of the *bin packing problem with conflicts*. Salem et al. [431] studied polyhedral aspects of the KPCG, presented new families of valid inequalities, and determined necessary and sufficient conditions for these inequalities to be facet defining. They also developed a *Branch-and-Cut* (B&C) algorithm that employs exact and heuristic separation procedures for the valid inequalities. Recently, Coniglio, Furini, and San Segundo [121] proposed a B&B algorithm which adopts a branching scheme similar to the one in [50], but makes use of bounds based on clique partition and on transformation to an MCKP. Extensive computational results show that this algorithm compares favorably with the one in [50]. Facet defining cutting planes for the KPCG were recently studied by Luiz, Santos, and Uchoa [347].

Approximation. Pferschy and Schauer [398] presented algorithms with pseudo-polynomial time and space complexity for special classes of conflict graphs (graphs with bounded tree-width and chordal graphs), and derived from these algorithms FPTASs for the same classes of graphs. In addition, they showed that the KPCG remains strongly $\mathcal{NP}$-hard for perfect graphs. Pferschy and Schauer [400] proposed, for special classes of graphs (bounded tree-width, chordal, weakly chordal, planar, perfect), several complexity results and approximation algorithms, both for the KPCG and for the *knapsack problem with forcing graph* (a KP01 with constraints requiring that, for each edge of the graph, at least one of the two items be selected). The same two problems were tackled by Gurski and Rehs [216], who provided pseudo-polynomial algorithms, based on DP, for the case of co-graphs as conflict and forcing graphs, and FPTASs for the case of graphs of bounded clique-width.

Heuristics. All heuristic algorithms in the literature were tested on instances with sparse graphs (density at most 0.4). The reactive local search algorithm in [248] applies a greedy algorithm followed by a swapping procedure and a diversification strategy. Other heuristics (local branching), metaheuristics (scatter search, local search, ant colony), and hybrid approaches have been examined by Hifi et al. [11, 242, 252, 254]. Quan and Wu [414] introduced a cooperative parallel adaptive neighborhood search algorithm, in which the cooperation stage collects and shares information on local optima found by subprocesses. Salem et al. [430] presented a probabilistic Tabu search heuristic with multiple neighborhood structures, a variant of Tabu search where the move is chosen probabilistically from a pool. The algorithm provided better results than those in [242] and [254].

2.10 Precedence constrained knapsack problem

Given a directed graph $G = (V, A)$ of n vertices, the *Precedence Constrained Knapsack Problem* (PCKP) is a generalization of the KP01 in which, for each arc $(i, j) \in A$, item j can only be selected if item i has been selected. Formally,

$$(2.1) - (2.3)$$
$$x_i \geq x_j \qquad \forall\, (i, j) \in A. \tag{2.20}$$

If G contains a cycle, the vertices of the cycle must either all be selected or all be excluded. It follows that the items in a cycle can be replaced by a single item, with cumulative profit and weight, so G can be assumed to be acyclic.

The PCKP is $\mathcal{NP}$-hard in the strong sense, as it can be shown (see, e.g., [297]) by reduction from the *clique problem* (given an undirected graph, find a maximal complete subgraph).

You and Yamada [499] presented a reduction procedure (pegging test) based on Lagrangian relaxation of the precedence constraints (2.20) and subgradient optimization. Computational experiments showed that most randomly generated instances with up to 2000 items can be optimally solved, after reduction, by a standard commercial solver.

Boland et al. [56] presented methods for determining facets of the PCKP polyhedron based on clique inequalities, and tested their effectiveness in reducing solution times when applied at the root node of a cutting plane approach and within a B&C framework. Espinoza, Goycoolea, and Moreno [172] provided a partial characterization of maximally violated inequalities and computationally evaluated their usefulness in improving the performance of a cutting plane algorithm.

Precedence constrained covering problems, which include the PCKP as a special case, were studied by McCormick et al. [370], who presented a strongly polynomial primal-dual approximation algorithm. A multi-period generalization of the PCKP, arising in the mining industry, was considered by Samavati et al. [432], who strengthened the LP relaxation of the problem so as to improve the efficiency of a classical sequencing heuristic for mine production scheduling.

2.11 Robust knapsack problems

Robust optimization is an approach to uncertain optimization, frequently adopted as an alternative to stochastic optimization. Loosely speaking, it consists in finding a solution that is "robust" (according to some criterion) against variations in the input data. For the knapsack problem, a set of data that defines an instance (profits, weights, capacity) is called a *scenario*. There is a vast literature on scenario-based robust optimization: we refer the interested reader to the classical books by Kouvelis and Yu [312] and Kasperski [292]. Three main research streams have been followed in the recent literature, as illustrated in the next sections.

2.11.1 Max-min knapsack problem

Consider a generalization of the KP01 in which we are given S scenarios, each characterized by a set of profits p_j^s $(j = 1, \ldots, n;\ s = 1, \ldots S)$. It is assumed that weights and capacity do not vary. The *Max-Min Knapsack Problem* (MMKP) consists in finding

a solution that maximizes the worst-case profit over all scenarios, i.e.,

$$\max \min_{1 \le s \le S} \left\{ \sum_{j=1}^{n} p_j^s x_j \right\} \tag{2.21}$$
$$\text{s.t. } (2.2) - (2.3).$$

The problem is $\mathcal{NP}$-hard in the strong sense, as it can be shown (see, e.g., [297]) by reduction from the *set covering problem* (given a set $\mathcal{S}$ and a family $\mathcal{F}$ of subsets of $\mathcal{S}$, find the minimum cardinality subfamily of $\mathcal{F}$ whose union is $\mathcal{S}$). In an earlier study on the MMKP, Yu [501] proved that the problem is strongly $\mathcal{NP}$-hard in the case of an unbounded number of scenarios. On the other hand, he showed that it can be solved in pseudo-polynomial time if the number of scenarios is bounded by a constant.

Taniguchi, Yamada, and Kataoka [461] introduced a particular surrogate relaxation and a reduction (pegging) procedure, and embedded them into a B&B algorithm that was computationally tested on a large set of benchmark instances. Goerigk [207] presented a new method to compute upper bounds and computationally proved that it considerably improves on the performance of B&B approaches to the MMKP. A different exact solution method, based on column generation and B&P, was developed by Pinto et al. [403] and computationally tested on large-size benchmark instances (with up to 20 000 items and 1000 scenarios).

Metaheuristic algorithms for the MMKP have been proposed by Sbihi [437] (greedy solution and Tabu search), Aldouri and Hifi [16] (hybrid reactive search), and Aldouri, Hifi, and Zissimopoulos [13] (greedy randomized search and path-relinking).

Taniguchi, Yamada, and Kataoka [462] adapted their algorithm [461] to the special case in which there are just two scenarios. A different exact approach for the two-scenarios case, based on mixed integer programming formulations and reduction procedures, was proposed by Hanafi et al. [232].

2.11.2 Min-max regret knapsack problem

In this case, the scenarios are defined by n intervals $[p_j^-, p_j^+]$, and the actual profit of an item j in scenario s can take any integer value, p_j^s, in the corresponding interval. As for the MMKP, weights and capacity do not vary across the scenarios. Each feasible solution x associated with scenario s has a value ($z^s(x) = \sum_{j=1}^{n} p_j^s x_j$) and a *regret*, $r^s(x) = z_\star^s - z^s(x)$, where $z_\star^s$ denotes the optimal solution value for scenario s. The (interval) *Min-Max Regret Knapsack Problem* (MMRKP) consists in finding a solution that minimizes the worst-case regret over all scenarios, i.e.,

$$\min \max_{1 \le s \le S} \{r^s(x)\} \tag{2.22}$$
$$\text{s.t. } (2.2) - (2.3).$$

The MMRKP is extremely challenging both from a theoretical and a practical point of view. Its precise complexity status is unclear. Being a generalization of the KP01 (the special case in which $p_j^- = p_j^+$ for all items) the MMRKP is $\mathcal{NP}$-hard, while it is an open question whether it is strongly $\mathcal{NP}$-hard. Deineko and Woeginger [147] proved that its decision version is complete for the complexity class Σ_2^p (see [199]), and hence is most likely not in $\mathcal{NP}$. Observe that even computing the regret of a single solution x is an $\mathcal{NP}$-hard problem, as it requires the solution of a KP01.

Furini et al. [192] evaluated the performance of standard approaches (Benders-like decomposition and B&C) when adapted to the MMRKP, and proposed a Lagrangian-based B&C algorithm, an iterated local search approach and an ILP-based heuristic. Extensive computational experiments showed that the method can solve instances with 50 items to proven optimality.

Kalaï and Vanderpooten [288] proposed the *lexicographic α-robust knapsack problem*, a variant that is somehow intermediate between the MMKP and the MMRKP, and presented an algorithm to determine the set of all α-robust solutions in pseudo-polynomial time.

In another variant, the *discrete min-max regret knapsack problem*, the profits do not vary in intervals but according to a discrete number of scenarios. Approximation results for this variant have been studied by Aissi, Bazgan, and Vanderpooten [8] and later surveyed by Candia-Véjar, Álvarez-Miranda, and MacUlan [75].

Wu et al. [485, 486] presented exact and heuristic algorithms for the extension of the MMRKP to the multiple knapsack problem (treated in Section 3.2) and to a further generalization of the problem (the GAP). Metaheuristics for another variant of the problem were recently presented by Wang et al. [476].

We finally mention that Conde [120] developed a linear-time algorithm for the min-max regret version of the *continuous* variant of the UKP (see Section 2.5.2).

2.11.3 Γ-robust knapsack problem

In this case, profits and capacity are constant, while the weight of each item j has a *nominal value* w_j and a variability range $[w_j - \underline{w}_j, w_j + \overline{w}_j]$. At most Γ weights can change from their nominal value to an arbitrary value in the interval. A solution is Γ-*robust* if it satisfies the capacity constraint (2.2) for any possible set of weights. The Γ-*Robust Knapsack Problem* (ΓRKP) is to find the maximum profit Γ-robust solution. Differently from the MMKP and the MMRKP, the ΓRKP is weakly $\mathcal{NP}$-hard.

Monaci, Pferschy, and Serafini [376] presented an FPTAS and a DP algorithm (with special techniques to reduce the space complexity). They computationally tested the resulting algorithm on a large set of randomly generated instances (up to 5000 items and $\Gamma = 50$). Monaci and Pferschy [375] analyzed the worst case ratio between optimal solution values of the KP01 and the ΓRKP, and extended their analysis to the fractional version of the problem (in which fractions of items may be packed).

Claßen, Koster, and Schmeink [117] considered a generalization of the ΓRKP, the *multi-band robust knapsack problem*, in which the variability range is subdivided into several smaller intervals (bands): they presented two DP algorithms and compared their performance by focusing on benchmarks with 2 bands. Büsing et al. [72] studied a different variant, the *recoverable* ΓRKP, in which it is possible to remove at most k items in order to restore the feasibility of any scenario: they presented different ILP formulations and computationally evaluated them on a large test-bed. A further recoverable ΓRKP variant, in which both weights and profits can vary, has been studied by Büsing, Koster, and Kutschka [71].

Goerigk et al. [208] considered a variant in which one is allowed to perform Q *queries* ($Q < n$): each query returns the actual weight of an item. The *robust knapsack problem with queries* consists in deciding which items have to be queried so as to maximize the optimal solution value to the resulting instance (in which the weight of the any non-queried item j is set to $w_j + \overline{w}_j$). They studied the *query competitiveness* (derived from *online optimization*, see Chapter 3) to evaluate the quality of an algorithm

by comparing its solution value for the RKQP to that produced by the best possible choice of items to query (if the real weights were known).

2.12 Compartmentalized knapsack problems

The problems treated in this section come from a number of real world applications, mostly related to two-phase steel roll cutting problems, in which the items to be produced are subdivided according to their thickness. As we will see, these problems were defined in various ways, induced by specific constraints appearing in different applications, and formalized through various modeling techniques, ranging from linear to nonlinear, from integer to mixed integer.

Consider a BKP (or an UKP, see Section 2.5) in which the item types are partitioned into ℓ classes $\{N_1, \ldots, N_\ell\}$ (corresponding to different typologies, e.g., steel thickness) and one is requested to build compartments inside the knapsack so that only items of the same class are loaded into a compartment. Building a compartment has a cost (depending on the specific application), the capacity of each compartment has a lower bound l_i and an upper bound u_i ($i = 1, \ldots, \ell$), and the creation of each compartment may produce a fixed loss of capacity of the original knapsack. The *Compartmentalized Knapsack Problem* (CKP) is to build compartments and assign the items in such a way that the overall profit (item profits minus building costs) is maximized. In the *Constrained Compartmentalized Knapsack Problem* (CCKP), the number of items of type j in the overall knapsack cannot exceed a given value β_j.

Hoto, Arenales, and Maculan [264] proposed exact algorithms for the CKP, based on the solution of a large number of knapsack problems (one for each feasible combination of weights of each class), and a heuristic in which the knapsack solutions are replaced by the computation of the Martello-Toth upper bound (see [361]). They also proposed a decomposition heuristic for the CCKP, in which feasible promising compartments are generated and a KP01 is solved to choose the compartments. Marques and Arenales [356] proposed various heuristics and an upper bound for the CCKP and performed extensive computational experiments. Leão et al. [330] presented an ILP formulation and a number of effective heuristics. Modified versions of the heuristics in [356] were proposed and computationally tested by Hoto and Bressan [265]. Other ILP models for the CCKP have been recently proposed by Inarejos, Hoto, and Maculan [274] and Quiroga-Orozco, Carvalho, and V. Hoto [417]. Approximation schemes for the CCKP (under a different name) were studied by Xavier and Miyazawa [488].

2.13 Bilevel knapsack problem

Bilevel programs model a hierarchical relationship between two decision-makers, a leader and a follower, who take their decisions in a sequential way: first, the leader (upper level), who has perfect knowledge of the follower's problem (objective function, constraints, and follower's decisions), takes an action to optimize her own objective, and then the follower (lower level) reacts, thereby influencing the decision of the leader (as the leader's objective depends on the follower's decision). These problems, taking into account two agents, each with her own individual objective, are known as *two-player Stackelberg games*. Although they belong to Game Theory, in the *Bilevel Knapsack Problems* (BLKP) the leader and the follower solve a combinatorial optimization problem, so we provide a description of the main results. The

interested reader is referred to the survey on bilevel programming by Labbé and Violin [322].

Several different definitions of the BLKP can be found in the literature. Brotcorne, Hanafi, and Mansi [68] proposed a two-phase DP algorithm for the BLKP originally introduced by Dempe and Richter [161], in which item profits vary with respect to the leader and the follower, and the leader first determines the knapsack capacity (within given lower and upper bound values), while the follower decides the subset of items to be selected by solving a KP01.

In a second version of the BLKP, introduced by Mansi et al. [352], the item set is split into two sets, and both players can select items from their respective sets to be inserted in a single knapsack: hence, the leader's decision interferes with the residual capacity available to the follower. In [352], a DP algorithm was presented, while Brotcorne, Hanafi, and Mansi [69] proposed, for a more general version, a two-step exact algorithm that (i) in the first step applies a DP procedure taking into account only the follower problem, and (ii) in the second step reformulates the bilevel problem as a one-level model, based on the optimal solutions computed in the first step.

Chen and Zhang [100] introduced another version with two knapsacks, one for each player: items can be selected simultaneously by both players, but each item profit depends on whether the item is selected by both the leader and the follower or by only one of them. Approximation algorithm were derived in [100] and improved by Qiu and Kern [412].

The most intensively studied version of the BLKP, introduced by DeNegre [162], is known as the *Bilevel Knapsack Problem with Interdiction Constraints* (BLKPI). In the BLKPI, two knapsacks, one for each player, with capacity c_1 and c_2, respectively, are considered, and there is a common set of items having the same profits but different weights for the two players: the item weight is v_j for the leader and w_j for the follower ($j = 1, \dots, n$). The interdiction constraints impose that if an item is selected by the leader, it cannot be selected by the follower. The objective of the follower is to maximize the total profit, while the objective of the hostile leader is to minimize it. Binary variables x_j and y_j represent the item selection for the leader and the follower, respectively. Formally,

$$\min \sum_{j=1}^{n} p_j y_j \tag{2.23}$$

$$\sum_{j=1}^{n} v_j x_j \leq c_1 \tag{2.24}$$

$$x_j \in \{0,1\} \qquad (j = 1, \dots, n), \tag{2.25}$$

where variables $y_1, \dots, y_n$ solve the follower's problem:

$$\max \sum_{j=1}^{n} p_j y_j \tag{2.26}$$

$$\sum_{j=1}^{n} w_j y_j \leq c_2 \tag{2.27}$$

$$y_j \leq 1 - x_j \qquad (j = 1, \dots, n) \tag{2.28}$$

$$y_j \in \{0,1\} \qquad (j = 1, \dots, n). \tag{2.29}$$

Caprara et al. [85] showed that the computational complexity of the decision version of three variants studied in [161], [352], and [162] is complete for the complexity class Σ_2^p, and hence there is no way of formulating each problem as a single-level integer program of polynomial size unless the polynomial hierarchy collapses. In addition, they studied these variants under so-called unary encodings, showing that the first two become polynomially solvable while the third one becomes $\mathcal{NP}$-hard. For the third variant, a *Polynomial Time Approximation Scheme* (PTAS) was derived, providing the first approximation scheme for a Σ_2^p-hard problem. Caprara et al. [86] proposed an exact algorithm that iteratively computes lower and upper bounds until optimality is reached. Upper bounds are derived by solving a single-level MILP model amended by *nogood cuts*. The algorithm was able to solve to optimality instances with up to 50 items. Fischetti et al. [178] studied a family of mixed integer linear bilevel problems known as *interdiction games*, which include the BLKPI, and proposed a Benders-like algorithm, in which the problem is reformulated as a single-level problem with an exponential number of constraints, called *interdiction cuts*. Additional families of modified or lifted interdiction cuts were presented, for which exact and heuristic separation procedures were developed. The algorithm was extensively tested on benchmark instances of the BLKPI showing significant better performance with respect to the method in [86]. Carvalho, Lodi, and Marcotte [89] proposed a polynomial algorithm with worst-case complexity $O(n^2)$ for the continuous relaxation of the BLKPI. Fischer and Woeginger [177] derived a faster algorithm that improves the worst-case complexity to $O(n \log n)$. Recently, Della Croce and Scatamacchia [157] proposed an exact algorithm that relies on an effective lower bound and computes leader's solutions by exploring the follower's subproblems that have better lower bounds. Computational experiments showed that the algorithm can solve instances with up to 500 items in very short computing times, thus significantly improving the results in [86] and [178], even though the tests were carried out on different machines. The algorithm was extended to the MMRKP (see Section 2.11.2) and computational results showed that it outperforms the approach in [192] on most instances.

Pferschy et al. [402] considered another variant in which the item set is partitioned into two sets, one for each player, and the leader can decide to assign an incentive to each of her own items, with the aim of influencing the follower's selection. The goal of the leader is to maximize the profit of the items selected from her set, reduced by the incentives, while the follower's goal is to maximize the profit of all selected items, whichever set they are taken from, increased by the incentives. The complexity is analyzed when the KP01 of the follower is solved to optimality, or when it is solved with greedy heuristics, and algorithms and ILP models are provided. In a companion paper, Pferschy, Nicosia, and Pacifici [396] considered the case where the weights can be modified by the leader (instead of the profits) and analyzed the complexity of the problem for three different solution strategies of the follower.

2.14 Extensions, generalizations, and research directions

This section lists a selection of less studied variants for which relatively few results have been published, and hence they could be promising research areas. For these problems we provide a concise description and references to the latest works.

In the *knapsack problem with minimum filling constraint* a minimum total weight of the selected items is imposed. Xu and Lai [491] developed an FPTAS for this

problem, which finds applications in auction clearing. In the *incremental knapsack problem*, the capacity is increasing over time periods, an item selected in a period cannot be removed afterwards and contributes with its profit for all time periods in which it is included, with different time multipliers for different periods. Della Croce, Pferschy, and Scatamacchia [152, 154] provided approximation results for the problem and for some variants. A PTAS for a variant of this problem was presented by Faenza and Malinovic [173].

In contrast to the previous variants, in the *temporal knapsack problem* the capacity does not change, but each item is active in a given time interval, and the goal is to select the maximum profit subset of items whose total weight respects the capacity at any point in time. Caprara, Furini, and Malaguti [77] and Caprara et al. [87] studied B&P algorithms, based on a Dantzig-Wolfe reformulation of the problem. Gschwind and Irnich [214] derived two types of column-generation stabilization methods. A DP algorithm for its exact solution was recently presented by Clautiaux, Detienne, and Guillot [118]. Slight variants of this problem can be encountered in the literature under different names. For example, Darmann, Pferschy, and Schauer [144] studied the case where all profits are one (under the name *resource allocation with time intervals*), obtaining a $(\frac{1}{2} - \varepsilon)$ approximation algorithm.

Two knapsack problems with neighbor constraints, in which dependencies between items are represented by adjacencies in a graph, were studied by Borradaile, Heeringa, and Wilfong [59] and, more recently, by Goebbels, Gurski, and Komander [205]: the 1-*neighbor knapsack problem*, in which an item can be selected only if at least one of its neighbors is also selected, and the *all-neighbors knapsack problem*, in which an item can be selected only if all its neighbors are also selected. Approximation and hardness results are provided in both works for several classes of graphs.

In the *penalized knapsack problem*, besides profit and weight, each item has a *penalty*, and the goal is to maximize the sum of the profits, decreased by the largest penalty value of the selected items. The problem was introduced by Ceselli and Righini [95], who presented an exact algorithm that performs an exhaustive search to identify the item with the largest penalty among items in the solution. Della Croce, Pferschy, and Scatamacchia [153] proposed a DP algorithm based on a core problem and on narrowing the relevant range of penalties.

In the *discounted knapsack problem*, a set of item groups is given: each group consists of three items where the third item represents a discounted offer, i.e., its weight is smaller than the sum of the weights of the first two items, while its profit coincides with the sum of the profits of the first two items. At most one item of each group can be selected, and the goal is to maximize the total profit while respecting the knapsack capacity. In Rong, Figueira, and Klamroth [424], an alternative core concept is proposed to partition the original problem into three sub-problems that are solved through DP. A DP algorithm with lower complexity was proposed by He et al. [237], who also derived an FPTAS, a 2-approximation algorithm, and a metaheuristic.

Malaguti et al. [350] defined the *fractional knapsack problem with penalties*, in which an item can be split at the expense of a penalty that depends on the fractional quantity, and presented mathematical models, an FPTAS, DP algorithms, and various heuristics. An improved FPTAS was developed by Kovalev [313].

Furini, Ljubić, and Sinnl [190] introduced the *minimum-cost maximal knapsack packing problem*: it consists in finding a maximal knapsack packing that minimizes the cost of the selected items. The authors proposed a DP algorithm, and showed that the problem is equivalent to a "dual" problem (*the maximum-cost minimal knapsack cover*).

In the *parametric knapsack problem*, profits are affine-linear functions of a parameter, and the goal is to compute the optimal solutions for all values of the parameter on the real line (or within a given interval). Approximation schemes for various cases were derived by Giudici et al. [204], and Holzhauser and Krumke [262]. Halman, Holzhauser, and Krumke [221] studied the case in which weights are affine-linear functions of a parameter, and presented an FPTAS for this problem.

In the *knapsack problem with qualitative levels,* each item has a "qualitative" (imprecise or vague) *level* instead of a numerical profit. Schäfer et al. [439] presented a DP approach to compute non-dominated solutions and two greedy algorithms to compute a single efficient solution.

We conclude with two variants which, although formally non-linear, have been tackled through combinatorial optimization techniques.

In the *collapsing knapsack problem*, the capacity is a non-increasing function of the number of selected items, i.e., it decreases when the number of selected items increases. Originally introduced as a nonlinear knapsack problem (see Wu and Srikanthan [483]), it has been solved: (i) by transformation into an equivalent KP01 (see [297], Section 13.3.7), with the drawback of the introduction of very large coefficients, and of a high correlation between profits and weights; (ii) more recently, through a very effective ILP formulation (see Della Croce, Salassa, and Scatamacchia [155], who also present a reduction procedure and an exact algorithm that can be extended to the multidimensional case).

In the *product knapsack problem,* the profits can have positive or negative value, and the goal is to maximize the product of the profits of the selected items. Halman et al. [222] showed that the problem is weakly $\mathcal{NP}$-hard. D'Ambrosio et al. [140] presented effective ILP models and a DP algorithm for its exact solution. The first FPTAS for this problem was recently presented by Pferschy, Schauer, and Thielen [401].

2.15 Conclusions

We have examined over two hundred results on single knapsack problems, mostly appeared in the last seventeen years.

The contributions we have examined show that knapsack problems frequently appear in real-world applications. The introduction to special issue [247] enumerates applications "encountered in numerous industrial sectors such as transportation, logistics, cutting and packing, telecommunication, reliability, advertisement, investment, budget allocation, and production management". In the previous sections, we have encountered other interesting application areas like, e.g., *make-to-order production* (the KPS), *finance* (the MCKP), *mine production* (the PCKP), and *steel industry* (the CKP).

Apart from being studied as standalone problems, knapsack problems frequently appear as subproblems in some relevant optimization problems from different areas, which further underlines the importance of investigating them. Notably, set covering formulations for bin packing and cutting stock problems are typically tackled through column generation, which requires to solve, at each iteration, a knapsack problem (see, e.g., [160]). Similarly, multidimensional cutting and packing problems frequently require the solution of knapsack problems. In addition, they often appear in algorithms to separate valid inequalities for optimization problems involving capacity constraints (see, e.g., [290] and [333]).

Further pointers to this huge literature will be provided in the following chapter, devoted to multiple, multidimensional, and quadratic knapsack problems, as well as to a succinct treatment of online and multiobjective knapsack problems.

Chapter 3

Knapsack problems - An Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic Knapsack Problems*

Chapter 2 treats the classical single knapsack problems and their variants. The present chapter covers multiple, multidimensional, and quadratic knapsack problems, as well as other relevant variants, such as, e.g., multiobjective and online versions.

3.1 Introduction

This is the second part of a survey aimed to review the developments appeared on knapsack problems after the publication of the books by Martello and Toth [361] in 1990 and Kellerer, Pferschy, and Pisinger [297] in 2004, until Summer 2021. We recall here the main definitions, referring the reader to Chapetr 2 for a general introduction to this research area.

The problem which originated this field is the famous *0-1 Knapsack Problem* (KP01): given a set of n items, each associated with a *profit* p_j and a *weight* w_j $(j = 1, \ldots, n)$, and a container (*knapsack*) of *capacity* c, find a subset of items with maximum total profit having total weight not exceeding the capacity.

A systematic research on the KP01 and its many variants started in the Fifties. It produced, over the next fifty years, an impressive number of scientific results, making this field a very relevant area of combinatorial optimization. The two mentioned monographs include in total about 700 bibliographic entries. The purpose of this survey is to review the subsequent developments, by mostly concentrating on the problems treated in the main chapters of Kellerer, Pferschy, and Pisinger [297] (which also correspond to the main chapters in Martello and Toth [361]) and on recent "hot" topics. We privilege problems with a clear combinatorial aspect, with a partial exception for non-linear knapsack problems: these are briefly described in order to introduce the *quadratic knapsack problems*, which are usually handled through combinatorial optimization tools. Other problems, closer to the computer science community (online knapsack problems) or belonging to the area of multiple criteria decision aiding (multi-objective knapsack problems), are succinctly treated in Section 3.6.

*The results of this chapter appears in: V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. "Knapsack problems - An Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic Knapsack Problems". In: *Computers & Operations Research* 143 (2022), 105693.

In Chapter 2, we have reviewed single knapsack problems. The present chapter is devoted to multiple, multidimensional, and quadratic knapsack problems, with a final section on the other knapsack problems mentioned above. *Multiple knapsack problems* are the natural generalization of the KP01: the items are packed into different knapsacks, each having its own capacity. For the *multidimensional knapsack problems*, the literature is instead ambiguous. Most of the papers use this term to refer to the case in which each item is characterized by two or more independent weighting functions (such as, for example, weight, volume, level of toxicity, etc.) and a capacity limit is imposed on each function. Other papers adopt the same name for the case in which both items and knapsack(s) are multidimensional rectangular boxes, and items have to be packed without overlapping. We refer to the former problem as the *multidimensional vector knapsack problem* (or the *multidimensional knapsack problem*, when no confusion arises), and to the latter problem as the *multidimensional geometric knapsack problem*.

Section 2.2 contains a thorough review of books, surveys, and special issues dedicated to knapsack problems. We only remind here the surveys specifically dealing with the arguments treated in this chapter:

- Bretthauer and Shetty [66], Li and Sun [337]: non-linear knapsack problems (continuous, integer, convex, nonconvex, separable, and nonseparable). We also refer to Ibaraki and Katoh [272] and Lin [343] for previously appeared surveys;
- Fréville [184] and Fréville and Hanafi [185]: *multidimensional knapsack problem* (exact, heuristic, approximation, and metaheuristic algorithms);
- Pisinger [405]: *quadratic knapsack problem* (with special emphasis on upper bounds computations);
- Lust and Teghem [349] considered the *multiobjective* version of single and multidimensional knapsack problems, reviewing exact, approximation, heuristic and metaheuristic algorithms;
- Kellerer and Strusevich [299]: *symmetric quadratic knapsack problem* (exact and approximation algorithms and their application on various scheduling problems);
- Christensen et al. [114]: *multidimensional geometric knapsack problem* (approximation and online algorithms);
- Laabadi et al. [320]: variants of the *multidimensional knapsack problem* (heuristic algorithms);
- Silva, Toffolo, and Wauters [451]: *three-dimensional geometric knapsack problem* (exact methods and extensive comparative experiments);
- Leao et al. [331]: *multidimensional geometric knapsack problem* (devoted to the case of irregular shapes);
- Iori et al. [277]: *two-dimensional geometric knapsack problem* (exact algorithms and mathematical models).

3.2 Multiple knapsack problems

Given m knapsacks with capacities c_i ($i = 1, \ldots, m$) and n items with profits p_j and weight w_j ($j = 1, \ldots, n$), in the *Multiple Knapsack Problem* (MKP) the goal is to select m disjoint subsets of items so that the total profit of the selected items is a maximum, and each subset is assigned to a knapsack whose capacity is no less than the total weight of the items in the subset. Let x_{ij} be a binary variable that takes the value one iff item j is assigned to knapsack i. The MKP can be modeled as:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{3.1}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_{ij} \leq c_i \qquad (i = 1, \ldots, m) \tag{3.2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad (j = 1, \ldots, n) \tag{3.3}$$

$$x_{ij} \in \{0, 1\} \qquad (i = 1, \ldots, m, j = 1, \ldots, n). \tag{3.4}$$

The MKP is $\mathcal{NP}$-hard in the strong sense, as it can be shown by reduction from the *3-partition problem* (see, e.g., [361], Section 1.3).

Exact solution. Exact *Branch-and-Bound* (B&B) algorithms for the MKP were proposed by Fukunaga and Korf [189], who developed a method for pruning nodes through dominance criteria between assignments of items to knapsacks. Fukunaga [188] presented improved B&B approaches making use of dominance criteria and symmetry breaking strategies, integrated with the B&B technique proposed in the Eighties by Martello and Toth [360]. The resulting algorithms were successfully tested on instances with up to 10 knapsacks and 100 items. Sitarz [454] developed an exact method based on a multiple criteria *Dynamic Programming* (DP). Computational tests showed, however, that the developed method was slower than the direct solution of (3.1)–(3.4) by a commercial software. Hickman and Easton [240] introduced a new class of valid inequalities obtained by merging two lower-dimension inequalities over the MKP polyhedron and presented conditions to check if the produced inequalities are facet defining. Dell'Amico et al. [158] solved the MKP by means of a pseudo-polynomial arc-flow model (inspired by Carvalho [88]). In this model, the m item subsets assigned to the knapsacks are represented as m paths in a graph where nodes are partial knapsack fillings and arcs are items. The model was improved by the inclusion of additional optimization techniques, such as partial B&B, primal decomposition, Benders cuts, and a graph reduction procedure. Computational tests proved the effectiveness of the resulting algorithm on instances with up to 500 items and 50 knapsacks, or 300 items and 150 knapsacks. A new upper bound for the MKP was recently developed by Detti [163].

Approximation. In contrast with the KP01, the MKP does not admit a *Fully Polynomial Time Approximation Scheme* (FPTAS) unless $\mathcal{P} = \mathcal{NP}$. Chekuri and Khanna [99] presented a *Polynomial Time Approximation Scheme* (PTAS) running in $n^{O(\log(1/\varepsilon)/\varepsilon^8)}$ time. Jansen [282] improved this result by proposing an *Efficient Polynomial Time Approximation Scheme* (EPTAS) with running time $2^{O(\log(1/\varepsilon)/\varepsilon^5)} \cdot poly(n) + O(m)$. Wang and Xing [478] proposed an approximation algorithm that iteratively fills the knapsacks, according to nondecreasing capacity, by selecting for each knapsack the subset of items with highest profit by means of an exact KP01 procedure, and provided a

worst-case analysis for the cases $m = 2$ and $m = 3$. Khutoretskii, Bredikhin, and Zamyatin [302] developed a 0.5-approximation algorithm with time complexity $O(mn)$ (excluding an initial sorting of items and knapsacks), based on specific lexicographic orderings of both knapsacks and items.

Heuristics. The MKP has been a playground for a variety of heuristics, including: population-based (Shah-Hosseini [446]), recursive constructive procedures (Lalami et al. [326]), artificial bee colony (Wei and Zhang [479] and Sabet, Shokouhifar, and Farokhi [427]) and artificial fish swarm (Liu et al. [344]).

3.2.1 Multiple subset sum problem

The *Multiple Subset Sum Problem* (MSSP) is a relevant special case of the MKP in which all knapsacks have the same capacity and the item profits are equal to their weights. The problem is $\mathcal{NP}$-hard in the strong sense, as it can be proved through the same reduction mentioned for the MKP.

A PTAS for the MSSP was presented by Caprara, Kellerer, and Pferschy [79], who also described a $\frac{2}{3}$-approximation algorithm for the bottleneck version of the problem (where the minimum total weight contained in any bin is to be maximized). They also showed that this is the best possible performance ratio achievable for the problem in polynomial time (unless $\mathcal{P} = \mathcal{NP}$). The PTAS was then generalized by Caprara, Kellerer, and Pferschy [78] to the case of different knapsack capacities. Later, Caprara, Kellerer, and Pferschy [80] introduced a polynomial time $\frac{3}{4}$-approximation algorithm for the MSSP, with running time that is linear in the number of items and quadratic in the number of knapsacks.

Kellerer, Leung, and Li [296] studied the MSSP with inclusive assignment set restrictions. In this problem, the assignment set of an item (i.e., the set of knapsacks that the item may be assigned to) is either a subset or a superset of the assignment set of another item. They proposed a PTAS and an efficient 0.6492-approximation algorithm. Pan and Zhang [391] showed how to solve MSSP instances with low density with an oracle for the *Shortest Vector Problem* (SVP), where the density d is measured by $d = n/(m \log_2 \bar{w})$, with $\bar{w} = \max_j\{w_j\}$, and they focused on instances having $d \leq 0.9408$.

3.2.2 Multiple knapsack assignment problem

Kataoka and Yamada [293] formulated the *Multiple Knapsack Assignment Problem* (MKAP) as an extension of the MKP in which the items are partitioned into disjoint sets and each knapsack may only be assigned items from one of the sets in the partition. Having the MKP as a special case, the problem is obviously strongly $\mathcal{NP}$-hard. The authors provided upper and lower bounds and used them to develop a heuristic that was computationally tested on randomly generated instances. More effective approaches were later presented by Martello and Monaci [358], who developed Lagrangian and surrogate relaxations, a constructive heuristic and a metaheuristic refinement procedure. Computational tests on benchmark MKAP instances proved the effectiveness of the algorithms.

Related problems having applications in real-world contexts were studied by Dimitrov et al. [166], who focused on variants arising in emergency relocation, and by Homsi et al. [263], who tackled a variant, arising in military and humanitarian situations, which involves loading constraints. The former article provides a heuristic algorithm, whereas the latter presents a mathematical model, Lagrangian and surrogate relaxations, and heuristic and matheuristic algorithms. Other MKP variants

arising in military contexts were studied by Simon, Apte, and Regnier [453], who extended the MKP in several ways so as to model constraints on self-sufficiency (i.e., capacity of maintaining operations with external aid) of a squad of Marines. They developed mathematical models and computationally tested them on a variety of randomly generated instances.

3.2.3 Multiple knapsack problems with special constraints

The *Multiple Knapsack Problem with Conflicts* (MKPC), in which pairs of items cannot be placed together in the same knapsack, was addressed by Basnet [39]. He developed constructive heuristic algorithms and tested them on instances with up to 500 items and 15 knapsacks.

The *Multiple Knapsack Problem with Setup* (MKPS) is an extension of the *Knapsack Problem with Setup* (KPS) treated in Section 2.6 to the case of multiple knapsacks: the items are characterized by a knapsack-dependent profit and belong to disjoint families, each one associated with a knapsack-dependent setup cost. Lahyani et al. [323] proposed a two-phase matheuristic algorithm, and a decomposition-based Tabu search matheuristic approach that extends the former. Amiri and Barkhi [28] recently studied a Lagrangian relaxation that decomposes the problem into a set of m independent single problems, and developed a greedy heuristic to produce feasible solutions. Both methods were evaluated through extensive computational experiments, showing that good quality solutions can be obtained in reasonable CPU times.

Motivated by a real application in the steel industry, Forrest, Kalagnanam, and Ladanyi [183], formulated the *Multiple Knapsack Problem with Color Constraints* (MKPCC), in which a color is associated with each item and the number of colors in any knapsack is restricted. They presented computational results for two very difficult MKPCC instances, solved using a column-generation approach.

Yamada and Takeoka [492] formulated the *Fixed-Charge Multiple Knapsack Problem* (FCMKP) as an extension of the MKP in which a fixed cost f_i must be paid if knapsack i is used in the solution. The problem is to decide the set of knapsacks to use, and to assign items to them, so that the total net profit (item profits minus knapsack costs) is maximized. A B&B algorithm presented in [492] was able to solve within 10 seconds almost all FCMKP instances with up to 32 000 items and 50 knapsacks. You and Yamada [500] introduced the *Budget-Constrained Multiple Knapsack Problem* (BCMKP), where again knapsack i costs f_i, but there is a prefixed budget to buy the knapsacks, and the objective is to maximize the total profit of the selected items. The problem was solved with a B&B algorithm making use of tools similar to those employed in [492] for the FCMKP.

Chen and Zhang [101] studied a generalization of the MKP in which the set of items is partitioned into groups, and, while each item has its own weight, the profit of a group is obtained only if every item of the group is packed. They derived both approximation and inapproximability results for a parameterized version of the problem where the total weight of the items in each group is bounded by a factor $\delta \in (0, 1)$ of the total capacity of all knapsacks.

Laalaoui and M'Hallah [321] proposed a variable neighborhood search algorithm for a particular MKP arising in a single machine scheduling problem where jobs must be processed in multiple time-windows and machine unavailability periods must be taken into account.

Nip and Wang [387] presented three approximation algorithms for the *two-phase knapsack problem*: given an MKP instance with an additional container of given capacity, the items have to be packed into the knapsacks and, in a second phase, the knapsacks have to be packed into the container so that the profit of the selected items is maximized.

3.3 Multidimensional (vector) knapsack problems

Given a knapsack having d different capacities c_i $(i = 1, \dots, d)$ and n items having profits p_j $(j = 1, \dots, n)$ and weight w_{ij} for the i-th capacity $(i = 1, \dots, d; j = 1, \dots, n)$, the *Multidimensional Knapsack Problem* (MdKP) consists in determining a subset of items such that its total i-th weight does not exceed the i-th capacity $(i = 1, \dots, d)$ and its total profit is a maximum. The MdKP can be modeled as:

$$\max \sum_{j=1}^{n} p_j x_j \tag{3.5}$$

$$\text{s.t.} \sum_{j=1}^{n} w_{ij} x_j \leq c_i \qquad (i = 1, \dots, d) \tag{3.6}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \dots, n). \tag{3.7}$$

As already reported in Section 3.1, surveys devoted to the MdKP have been published by Fréville [184], Fréville and Hanafi [185], and Laabadi et al. [320]. The MdKP is in practice an *Integer Linear Programming* (ILP) problem with binary variables and non-negative coefficients. It is strongly $\mathcal{NP}$-hard, although some authors classify it as weakly $\mathcal{NP}$-hard. The confusion probably arises from the fact that: (i) the general problem (3.5)-(3.7), in which n and d are input values, is indeed strongly $\mathcal{NP}$-hard (see, e.g., Garey and Johnson [199], Problem [MP1]); (ii) if instead d is a constant, DP solves the problem in pseudo-polynomial time $O(n\bar{c}^d)$ (where $\bar{c} = \max_{i \in \{1,\dots,d\}} c_i$). (A similar situation occurs for the MKP, although, to the best of our knowledge, all authors classify it as strongly $\mathcal{NP}$-hard.)

Over the years, the MdKP has been addressed with some exact techniques and many (meta)heuristic methods.

Exact solution. Martello and Toth [362] proposed a B&B algorithm for the case $d = 2$, making use of heuristics, reduction techniques, and multiple relaxations. Vimont, Boussier, and Vasquez [471] developed an implicit enumeration scheme which uses reduced cost constraints to fix non-basic variables and to prune nodes of the search tree. Puchinger, Raidl, and Pferschy [410] presented different exact and heuristic algorithms, all based on adapting to the MdKP the classical *core* concept for the KP01 (limit the search, at least initially, to a small set of core items, while setting all variables corresponding to items outside the core to their presumably 0-1 optimal values). The core problem was also used by Della Croce and Grosso [150] in a reduction method, and by Mansini and Speranza [354] in an exact approach based on a recursive variable-fixing process. Other B&B methods were proposed by Boussier et al. [61], who improved the algorithm in [471] through a multi-level search strategy, and by Boyer, El Baz, and Elkihel [62], who developed a method combining B&B and DP. Bektas and Oğuz [45] proposed a simple separation procedure to identify cover inequalities.

With the aim of producing good upper bounds, Kaparis and Letchford [289] developed a cutting plane method based on lifted cover inequalities. Balev et al. [38]

presented a preprocessing procedure based on the iterated use of DP and *Linear Programming* (LP) relaxations. Gu [215] studied reduction criteria based on the concept of core problem.

Very recently, Setzer and Blanc [444] studied the geometric aspect of the MdKP solution space, and proposed an empirical orthogonal constraint generation method aimed to reduce the number of capacity constraints, producing some new best-known upper bounds and proving optimality for an open instance. They computationally tested this procedure on benchmarks proposed by Chu and Beasley [116] (90 instances with $d = 30$ and $n \in \{100, 250, 500\}$) and compared it with other approaches from the literature, especially with [471]. Their computational results showed that, for small instances ($n = 100$), the original MdKP formulation can be easily solved by means of a commercial solver (Gurobi). Instead, the more complex approaches proposed in [471] and [444] payed off in cases of large instances ($n \in \{200, 500\}$) for which the optimal solution is known by now just for 9 out of the 60 instances.

Heuristics. Already in 2004, Kellerer, Pferschy, and Pisinger [297] mentioned that: *"In recent years (MdKP) turned out to be one of the favourite playgrounds for experiments with metaheuristics, in particular Tabu search and genetic algorithms"*. In the last years, this trend has increased enormously, and a large variety of metaheuristics (sometimes fanciful, sometimes equipped with strong mathematical background) has been developed to tackle the MdKP. We next provide some details on the methods obtained from relevant mathematical frameworks. For the sake of completeness, we give in the next two paragraphs a concise list of other heuristics and nature inspired approaches. Moraga, Depuy, and Whitehouse [378] proposed a metaheuristic to construct and improve feasible solutions by means of randomized priority rules and local search techniques. Vasquez and Vimont [470] introduced a hybrid method which combines a limited B&B variable fixing heuristic with Tabu search. Thiongane, Nagih, and Plateau [465] proposed a hybrid Lagrangian heuristic for the case $d = 2$. Boyer, Elkihel, and El Baz [63] presented a heuristic approach based on surrogate relaxation, DP and *Branch-and-Cut* (B&C). Wilbaut, Salhi, and Hanafi [481] proposed an iterative scheme based on a dynamic fixing of the variables through LP relaxations. Al-Shihabi and Ólafsson [15] designed a hybrid algorithm that combines a nested partition method with a binary ant system and LP. Angelelli, Mansini, and Speranza [30] proposed a heuristic based on a kernel search framework. Hanafi and Wilbaut [231] described a heuristic based on an iterative generation of upper and lower bounds. Della Croce and Grosso [151] proposed a heuristic that combines core problem approaches and a branching scheme. Yoon, Kim, and Moon [498] and Hill, Cho, and Moore [258] studied Lagrangian-based heuristics.

Various kinds of metaheuristics for the MdKP have been studied: harmony search (Kong et al. [310]), scatter search (Hanafi and Wilbaut [230]), multi-verse optimization (Abdel-Basset et al. [2]), local search (Wang, Wang, and Xu [475]), genetic and randomized approaches (Jalali Varnamkhasti and Lee [281], Lai, Yuan, and Yang [324], Ünal and Kayakutlu [468], García et al. [196], Martins and Ribas [366]). The main nature inspired paradigms adopted for the MdKP are particle swarm (Chih (et al.) [110–112], Haddar et al. [218], Mingo López, Gómez Blas, and Arteta Albert [374]) and ant colony optimization (Kong, Tian, and Kao [309], Ke et al. [295]).

Variants and generalizations

A number of MdKP variants and generalizations have been investigated in the recent literature.

In the *Unbounded Multidimensional Knapsack Problem* (UMdKP), modeled by the MdKP with (3.7) replaced by

$$x_j \geq 0 \text{ and integer} \qquad (j = 1 \ldots, n),$$

the number of copies that can be selected for each item is unlimited. The problem was tackled by He, Hartman, and Pardalos [236] through an extension of their approach to the *unbounded knapsack problem* (see Section 2.5.2).

Quadri, Soutif, and Tolla [413] developed a B&B algorithm for a quadratic variant of the MdKP which calls for the maximization of a concave separable quadratic objective function.

The *Multidemand Multidimensional Knapsack Problem* (MMKP) extends the MdKP by including q additional dimensions, for each of which a demand constraint must be satisfied. This implies adding

$$\sum_{j=1}^{n} w_{ij} x_j \geq c_i \qquad (i = d+1, \ldots, d+q)$$

to model (3.5)-(3.7). The problem has been attacked through metaheuristics: Tabu search algorithms were developed by Cappanera and Trubian [76], Arntzen, Hvattum, and Løkketangen [32], and Lai, Hao, and Yue [325], while a scatter search approach was proposed by Hvattum and Løkketangen [271].

The *Multiple Multidimensional Knapsack Problem* (MMdKP) generalizes both the MdKP and the MKP (Section 3.2) by considering multiple knapsacks having multiple dimensions. Ahuja and Cunha [6] solved it by means of a very large-scale neighborhood search. Ang, Cao, and Ye [29] formulated a multi-period sea cargo mix problem as an MMdKP and solved it by means of two heuristic algorithms. A further generalization of the problem, the *Multiple Multidimensional Knapsack with Family-Split Penalties* (MMdKPF), was proposed by Mancini, Ciavotta, and Meloni [351] who solved it through a combinatorial Benders decomposition approach.

3.3.1 Multidimensional multiple-choice knapsack problem

The *Multidimensional Multiple-Choice Knapsack Problem* (MdMCKP) is a generalization of the MdKP in which, as for the *multiple-choice knapsack problem* (see Section 2.7), the set of items is partitioned into ℓ classes $N_1, \ldots, N_\ell$, and exactly one item of each class must be selected. This implies adding

$$\sum_{j \in N_i} x_j = 1 \qquad (i = 1, \ldots, \ell) \tag{3.8}$$

to model (3.5)-(3.7).

Exact solution. Sbihi [436] proposed a B&B algorithm that generates an initial lower bound and determines an upper bound for each level of exploration according to a best-first strategy. Extensive computational experiments showed that the algorithm can solve instances with up to 50 classes, 20 items per class, and 7 capacity constraints. Han, Leblet, and Simon [223] proposed new methods to generate hard benchmark instances of the MdMCKP, and provided an extensive experimental evaluation. Ghasemi and Razzazi [200] extended the core concept (see Section 3.3) to the MdMCKP, and used it in a B&B algorithm. Extensive computational experiments showed that the algorithm can solve large size instances when there is no correlation

between profits and weights (up to 100 classes, 100 items per class, and 15 capacity constraints), but it can only solve small-size instances when such correlation exists. Hifi and Wu [257] proposed an alternative model obtained by using information collected from the optimal Lagrangian solution. Computational experiments showed that the CPLEX solver is more efficient when the new model is used. Gokce and Wilhelm [209] provided valid inequalities for the minimization version of the MdMCKP, and two lifting procedures to strengthen them. The usefulness of these inequalities was evaluated by embedding them in a branch-and-cut algorithm. Voß and Lalla-Ruiz [472] compared the above model (3.5)-(3.8) to an alternative formulation which uses binary variables x_{ij} taking the value 1 iff item j from class i is selected: computational experiments showed that the former obtains slightly better results. Mansini and Zanotti [355] proposed an algorithm based on the concept of core (see Section 3.3), which iteratively constructs and solves a sequence of sub-problems by means of a recursive variable-fixing procedure, until an optimality condition is satisfied. Computational tests showed that the algorithm could solve to proven optimality ten open benchmark instances (proposed by Hifi, Michrafy, and Sbihi [251], Shojaei et al. [450], and Mansi et al. [353]) and improve the best-known values for many other instances.

Heuristics. Hifi, Michrafy, and Sbihi [250] presented a guided local search metaheuristic based on a penalization strategy, and tested it on 13 instances with up to 10 capacity constraints and 4000 items. These results were improved by Hifi, Michrafy, and Sbihi [251], who proposed a reactive local search algorithm and introduced twenty new instances with up to 30 capacity constraints and 7000 items. Akbar et al. [10] presented a polynomial time heuristic based on the iterative construction of convex hulls, and tested it on randomly generated instances. Hiremath and Hill [259] provided an analysis of the 13 MdMCKP benchmark instances, and additionally proposed a new test set of 270 instances, which, however, were rarely used in subsequent papers. Shahriar et al. [447] proposed a parallel version of a heuristic algorithm previously developed by Akbar et al. [9]. Cherfi and Hifi [109] developed a rounding procedure combined with a truncated B&B algorithm applied, at selected nodes, to a restricted model handled in a column generation fashion. Their approach improved the computational results in [251]. Cherfi and Hifi [108] developed an approach that combines local branching with a truncated B&B algorithm, based on their column generation procedure [109]. Feng, Ren, and Zhan [175] presented a method merging ant colony optimization and Lagrangian relaxation, and compared it with the algorithm in [251], obtaining comparable results. Crévits et al. [128] proposed four variants of a heuristic method based on the iterated solution of semi-continuous relaxations where only a subset of variables is restricted to binary values. The algorithms were enhanced by a local search procedure and by reduction rules, improving the results in [108]. Mansi et al. [353] improved the results in [109] and [128] with a hybrid heuristic that iteratively refines the global upper bound value through LP relaxations, and the lower bound value by solving reduced problems. Heuristics based on the principles of Pareto algebra were proposed by Shojaei et al. [450] and Zennaki [503]. Htiouech, Bouamama, and Attia [266] developed and computationally tested an algorithm that alternates between constructive and destructive phases. Chen and Hao [102] presented two variants of a "reduce and solve" heuristic that combines problem reduction with CPLEX solver and favorably compared it with the methods in [353], [450] and [108]. Hifi and Wu [256] developed a heuristic based on a Lagrangian relaxation used to define a neighbourhood search and successfully compared it with the methods in [128] and [353]. Gao et al.

[195] proposed an algorithm based on the concept of *pseudo-gap*, a hypothesized gap between upper and lower bounds. A pseudo-gap enumeration was combined with cuts based on reduced cost constraints. The current state-of-the-art is an effective kernel search heuristic recently presented by Lamanna, Mansini, and Zanotti [327].

Variants. Caserta and Voß [91] addressed the MdMCKP with uncertainty on parameters w_{ij}, by adopting a robust optimization approach. They developed a matheuristic algorithm to tackle both the nominal and the robust versions of the problem, and tested it on benchmark instances.

In the *Multidimensional Knapsack Problem with Generalized Upper Bound Constraints* (MdKP-GUB), items belong to disjoint families and it is required that at most one item per family is chosen. The MdKP-GUB was invented and attacked through various metaheuristics in a series of papers by Li (et al.) [339–341].

3.4 Multidimensional geometric knapsack problems

In *Cutting and Packing* (C&P) problems one is given a set of geometric objects in multidimensional real space which have to be packed, without overlapping, into a given set of multidimensional containers (knapsacks). Equivalently, the containers have to be cut in order to produce the items, which explains why these two application domains, despite being quite different in practice, are studied together and lead to similar mathematical models and solution algorithms.

The C&P literature addresses several multidimensional geometric knapsack problems. They have in common the fact that each item is associated with a profit and there is a single knapsack where to pack/cut the items so as to maximize the total profit, but they differ in a number of characteristics, such as shape and number of dimensions of items and knapsacks, as well as additional geometric constraints.

Revising the complete literature on this area is beyond the scope of this survey, so we limit our review to the case of *orthogonal* packing, where items and knapsack are rectangular shapes (rectangles or boxes) and the items have to be packed/cut with their edges parallel to those of the knapsack. This represents by far the most studied field in the C&P area. For the case of irregular shapes, we refer the reader to the very recent survey by Leao et al. [331], who presented a complete review of mathematical models and solution techniques.

3.4.1 Two-dimensional knapsack problem

The most famous multidimensional geometric knapsack problem is known as the *Two-Dimensional Knapsack Problem* (2D-KP), and consists in finding a maximum profit subset of a set of rectangular items that can be orthogonally packed into a single rectangular knapsack. The 2D-KP is strongly $\mathcal{NP}$-hard, since in the special case in which all item heights are equal, the problem of testing whether the whole set of items fits in the knapsack is equivalent to the well-known (one-dimensional) *bin packing problem* (pack n items of weight w_j ($j = 1, \ldots, n$) into the minimum number of knapsacks of capacity c).

Exact solution. For exact algorithms and mathematical models we refer the reader to the very recent survey by Iori et al. [277], who also implemented a library, `2DPackLIB`, of benchmarks and links for two-dimensional orthogonal cutting and packing problems (see `http://or.dei.unibo.it/library/2dpacklib-2-dimensional-cutting-and-packing-library`). To the best of our knowledge, the only relevant work published on the 2D-KP after this survey is the one by Cunha, Lima, and Queiroz

[129], who presented different reduction procedures based on several types of grids of points, and used them to improve the performance of a mathematical model.

Approximation. Caprara and Monaci [82] presented an approximation algorithm that guarantees a worst-case ratio of $3+\varepsilon$. Jansen and Zhang [284] improved the result with an algorithm having a worst-case ratio at most $2+\varepsilon$. For the special 2D-KP case in which items and knapsack are squares, Harren [233] introduced a $(5/4+\varepsilon)$-approximation algorithm, whereas Heydrich and Wiese [239] introduced an EPTAS with running time $O_\varepsilon(1)n^{O(1)}$, where $O_\varepsilon(1)$ denotes a value that is constant for constant ε. For the case in which the weight of each item is equal to its area, Lan et al. [328] presented a EPTAS having running time $O(n\log n+4^{\varepsilon^{-6}})$. For a more detailed overview on approximation and online algorithms for the 2D-KP, we refer the reader to the survey by Christensen et al. [114] (mainly devoted to the *bin packing problem* but including a section on geometric knapsack problems).

Heuristics. A number of heuristic and metaheuristic algorithms was introduced in recent years for the 2D-KP. Beasley [41] proposed a successful population heuristic based upon a non-linear formulation and tested it on several benchmarks involving up to 4000 items. Improved computational results were obtained by Alvarez-Valdés, Parreño, and Tamarit [26] (Tabu search), Hadjiconstantinou and Iori [220] (genetic algorithm), Leung et al. [335] (simulated annealing), Kierkosz and Luczak [303] (hybrid evolutionary algorithm), and Shiangjen et al. [449] (iterative bidirectional heuristic).

Several variants of the 2D-KP have been also considered in the literature. Silveira, Xavier, and Miyazawa [452] presented a $(4+\varepsilon)$-approximation algorithm for the 2D-KP with unloading constraints and a $(3+\varepsilon)$-approximation algorithms for two other special cases. Zhou et al. [506] presented mathematical models and computational experiments for the 2D-KP with block packing constraints, in which the knapsack is divided into disjoint blocks and each packed item has to be allocated inside a block. Queiroz et al. [416] proposed ILP models, a B&C algorithm, and metaheuristics for the 2D-KP with conflicts, in which some pairs of items cannot both be packed into the knapsack. For the special case in which the knapsack is a square and the objective is to pack all the items into the smallest possible square, lower bounds, mathematical models, and an exact algorithm have been presented in Caprara et al. [84] and Martello and Monaci [357].

3.4.2 Two-dimensional knapsack problems with guillotine constraints

The most widely studied variant of the 2D-KP is the *Two-Dimensional Knapsack Problem with Guillotine Constraints* (2D-KPGC), in which the items must be obtained from the knapsack through a series of *guillotine cuts* (i.e., edge-to-edge cuts parallel to the edges of the knapsack). Each series of parallel cuts is called a *stage*, and when a limit k (frequently encountered in real-world applications) is imposed on the maximum number of stages, the problem is called the *k-staged* 2D-KPGC. In the constrained version of the 2D-KPGC the number of items that can be selected for each item type is bounded. While this version is strongly NP-hard, the *unconstrained* version (here referred to as the 2D-UKPGC) can be solved to optimality in pseudo-polynomial time through DP (see Iori et al. [277]).

Exact solution. For what concerns exact algorithms and mathematical models, we again refer the reader to the thorough recent survey by Iori et al. [277].

Approximation. As regards approximation algorithms, Caprara, Lodi, and Monaci

[81] considered the special case of the 2-staged 2D-KPGC in which the profit of each item coincides with its area and generalized the approximation scheme presented in [79], obtaining an (absolute) approximation scheme.

Heuristics. Alvarez-Valdés, Parajón, and Tamarit [25] presented a *Greedy Randomized Adaptive Search Procedure* (GRASP) and a Tabu search algorithm for the 2D-KPGC. Hifi [241] introduced a hybrid approach which combines depth-first search using hill-climbing strategies and DP. Bortfeldt and Winter [60] proposed a genetic algorithm for the 2D-KP and the 2D-KPGC, also able to handle both the 2D-UKPGC and the case where the items can be rotated by 90°. Borgulya [58] introduced, for the 2D-KPGC, an evolutionary method based on an estimation of a probability distribution from a set of solutions, and computationally compared it with the approach in [60].

3.4.3 Geometric knapsack problems in higher dimensions

The *Three-Dimensional Geometric Orthogonal Knapsack Problem* (3D-KP) is the extension of the 2D-KP to the case in which knapsack and items are three-dimensional boxes. The literature on this problem and on related variants is quite scarce compared to the one on the 2D-KP.

For what concerns exact methods, a recent review, also providing the outcome of extensive comparative experiments, was presented by Silva, Toffolo, and Wauters [451].

Diedrich et al. [165] proposed a $(7+\varepsilon)$-approximation algorithm.

Egeblad and Pisinger [168] generalized to the 3D-KP an iterative heuristic for 2D-KP. They adopted the sequence pair representation by Murata et al. [381], which represents a solution by two item permutations, and obtained feasible packings through the algorithm by Pisinger [405].

Queiroz et al. [415] developed a DP algorithm based on reduced raster points for a k-staged three-dimensional knapsack problem with guillotine constraints.

Baldi, Perboli, and Tadei [37] introduced the 3D-KP with balancing constraints, where the packing center of mass is forced to lie into a specific boxed domain inside the knapsack. They presented a mixed-integer linear programming model and a heuristic algorithm.

We finally mention the *d-Dimensional Orthogonal Knapsack Problem* (D-KP), a further extension of the 2D-KP to the case in which items and knapsack are d-dimensional rectangles. Harren [233] provided a $(1+1/2^d+\varepsilon)$-approximation algorithm for the D-KP.

3.5 Quadratic knapsack problems

The quadratic knapsack problem is a special case in the area of nonlinear knapsack problems, characterized by a combinatorial structure. Before analyzing the problem in detail, we briefly introduce this general area in the next section.

3.5.1 Nonlinear knapsack problems

As previously stated, this survey is mainly addressed to the combinatorial optimization community, so we just mention in a succinct way arguments with a clear continuous, non-linear structure like those treated in this section. Consider a general

optimization problem of the form

$$\max f(x) \tag{3.9}$$
$$\text{s.t. } g(x) \leq c \tag{3.10}$$
$$x \in D \tag{3.11}$$

where $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $f(x)$ and $g(x)$ are continuous differentiable functions, c is a non-negative value, and $D \subseteq \mathbb{R}^n$. Now consider the special case where both $f(x)$ and $g(x)$ are separable functions, and D includes bounds and integrality requirements on (part of) the variables. Using the knapsack terminology, the resulting problem can be defined as follows. Given n items, with item j having a *profit function* $f_j(x_j)$ and a *weight function* $g_j(x_j)$, associated with n variables x_j limited by lower bounds l_j and upper bounds u_j $(j = 1, \ldots, n)$, determine non-negative x_j values such that the total weight does not exceed a given *capacity* c and the total produced profit is a maximum. A subset $\overline{N}$ of the x_j variables can be restricted to take integer values. Formally, the *Non-Linear Knapsack Problem* (NLKP) is:

$$\max \sum_{j \in N} f_j(x_j) \tag{3.12}$$
$$\text{s.t. } \sum_{j \in N} g_j(x_j) \leq c \tag{3.13}$$
$$l_j \leq x_j \leq u_j \qquad (j = 1, \ldots, n) \tag{3.14}$$
$$x_j \text{ integer} \qquad (j \in \overline{N} \subseteq \{1, \ldots, n\}), \tag{3.15}$$

where $f_j(x_j)$ and $g_j(x_j)$ are nonlinear, non-negative, non-decreasing functions. Note that, in general, there is no further assumption on functions $f_j(x_j)$ and $g_j(x_j)$, i.e., they can be nonconvex and nonconcave.

The NLKP has many applications in various fields such as portfolio selection, stratified sampling, resource-allocation, production planning, and resource distribution. Nonlinear knapsack problems have been studied in the books by Ibaraki and Katoh [272], and Li and Sun [337]. For a general introduction to these problems, the reader is referred to the classical survey by Bretthauer and Shetty [66] as well as to Lin [343] who reviewed a number of knapsack problem variants, among them the MKP, the MdKP, and the *quadratic knapsack problem* discussed in the next section. D'Ambrosio and Martello [137] proposed a fast and effective heuristic algorithm enriched by a local search post-optimization procedure for the NLKP. Relaxations and heuristics for the natural extension of the NLKP to the case of multiple knapsacks (*Non-Linear Multiple Knapsack Problem*, NLMKP) were proposed by D'Ambrosio, Martello, and Mencarelli [138]. Other variants and generalizations of the NLKP and the NLMKP have been studied by Stefanov [458], Elbassioni, Karapetyan, and Nguyen [171], Goos et al. [211], and D'Ambrosio, Martello, and Monaci [139].

3.5.2 Quadratic knapsack problem

In the *Quadratic Knapsack Problem* (QKP), one is given a knapsack with capacity c and n items having profit p_j and weight w_j $(j = 1, \ldots, n)$. An extra non-negative profit p_{ij} is earned if both items i and j are selected $(i, j = 1, \ldots, n; i \neq j)$. The objective is to find a subset of items of total weight not exceeding the capacity, which maximizes the overall profit, calculated as the sum of the profits of the selected items and of

their pairwise profits. Formally

$$\max \sum_{j=1}^{n} p_j x_j + \sum_{j=1}^{n-1} \sum_{i=1+j}^{n} p_{ij} x_i x_j \tag{3.16}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq c \tag{3.17}$$

$$x_j \in \{0,1\} \qquad (j = 1, \ldots, n). \tag{3.18}$$

The problem is strongly $\mathcal{NP}$-hard, as it can be shown (see, e.g., [297]) by reduction from the *clique problem* (given an undirected graph, find a maximal complete subgraph). A thorough review of the QKP was provided by Pisinger [405]. The `C` code of an effective algorithm (`Quadknap`) for the exact solution of the problem, developed by Caprara, Pisinger, and Toth [83], is available at `http://hjemmesider.diku.dk/~pisinger/codes.html`.

Exact solution. Billionnet and Soutif [53] presented a B&B algorithm based on the computation of an upper bound by means of a Lagrangian decomposition previously studied in Billionnet, Faye, and Soutif [52]. The method, evaluated on a large set of randomly generated benchmarks, solved almost all instances with up to 150 variables, and with up to 300 variables for medium and low density profit matrices. Billionnet and Soutif [54] proposed three ways of linearizing the QKP into an equivalent *Mixed-Integer Program* and solved it with a commercial software. Pisinger, Rasmussen, and Sandvik [407] proposed an exact algorithm based on a variable fixing procedure, called *aggressive reduction*: it employs the upper bound proposed in [53] and another upper bound introduced in [83], as well as several heuristic algorithms to compute lower bounds. The procedure is followed by a B&B algorithm. The algorithm was able to solve large-size instances with up to 1500 variables. Wang, Kochenberger, and Xu [474] compared the performance of the CPLEX optimizer for quadratic integer programs and that of the method in [53], showing that the former outperforms the latter. Létocart, Nagih, and Plateau [334] introduced a reoptimization method that improves the efficiency of the solution of the numerous continuous linear knapsack problems generated by the Lagrangian approaches proposed in Billionnet, Faye, and Soutif [52] and [83]. Rodrigues et al. [420] presented a linearization method which replaces the quadratic terms of the objective function with a set of linear constraints, and applied it within a B&B algorithm. Computational experiments showed that the algorithm is able to provide better results than that of [407] for all the tested instances with low density and up to 200 variables. Fampa et al. [174] introduced a parametric convex quadratic relaxation of the problem, developed a primal-dual interior point method to obtain the best possible bound, and proposed valid inequalities for the convex quadratic model. Worth is mentioning however that Schauer [440] criticized the randomly generated instances normally adopted for computational experiments on algorithms for the QKP: he showed that, for a large family of classical test instances used in the literature, a basic greedy algorithm (consecutively insert the items sorted by non-increasing weight as long as they fit) produces solutions of value asymptotically very close to the optimum as the instance size tends to infinity. In [440] an additional class of instances, for which finding a good solution is much harder, was introduced. Fomeni, Kaparis, and Letchford [180] presented a Cut-and-Branch algorithm that embeds a cutting-plane phase, primal heuristics, reduction rules, and a B&B phase, and tested it also

on instances introduced in [440].

Approximation. Taylor [463] presented an approximation algorithm that guarantees, for any given $\varepsilon > 0$, an approximation ratio within $O(n^{2/5+\varepsilon})$ and a run time within $O(n^{9/\varepsilon})$. The quadratic profit p_{ij} can be seen as the cost of the edge between vertices i and j of an undirected weighted graph whose vertices represent the knapsack items. Pferschy and Schauer [399] studied the QKP on special graph classes. They provided an FPTAS for the QKP on graphs of bounded treewidth, a PTAS for the QKP on planar graphs, and they showed that the QKP is strongly $\mathcal{NP}$-hard on 3-book embeddable graphs (generalizations of planar graphs). Wu, Jiang, and Karimi [487] sudied the convergency property of a logarithmic descent direction approximation algorithm.

Heuristics. Patvardhan, Prakash, and Srivastav [393] proposed a quantum inspired evolutionary algorithm, later improved and parallelized in Patvardhan, Bansal, and Srivastav [392]. Yang, Wang, and Chu [495] presented a GRASP and a Tabu search algorithm and successfully compared them with a heuristic approach developed by Xie and Liu [489]. Azad, Rocha, and Fernandes [36] proposed an artificial fish swarm algorithm having good computational performance, although not outperforming the method in [489]. Fomeni and Letchford [181] presented an effective DP heuristic obtained by modifying the classical DP approach for the KP01. The algorithm was enhanced by considering specific item ordering and a tie-breaking rule, giving solutions very close to the optimal ones. Toumi, Cheikh, and Jarboui [466] introduced two variable neighborhood search heuristics and successfully compared them with the method in [495]. Cunha, Simonetti, and Lucena [130] investigated two Lagrangian heuristics and compared them with the method in [181], showing that all these approaches provide good quality solutions. Chen and Hao [105] proposed a method which introduces an additional cardinality constraint to decompose the problem into several disjoint sub-problems. The most promising sub-problems are then solved through reduction procedures and Tabu search. The resulting algorithm was compared with the algorithms in [489], [495], and [181].

Variants and generalizations

Kellerer and Strusevich [298] presented an FPTAS for a variant of the QKP, the *Symmetric Quadratic Knapsack Problem* (SQKP), which has several applications in machine scheduling. In the SQKP, the extra profit p_{ij} is obtained by multiplying the weight w_i by a non-negative integer coefficient, and it is earned if both i and j are selected or neither i nor j are selected. This FPTAS was later improved by Xu [490]. A generalization and extension of these works, also relevant for the SQKP, appeared in Kellerer and Strusevich [300]. In their survey, Kellerer and Strusevich [299] reviewed the main results on the SQKP and reformulated a number of scheduling problems through it.

Schulze et al. [441] introduced the *Rectangular Knapsack Problem* (RKP) as a special case of the QKP in which the items have profit $p_j = 0$, weight $w_j = 1$ (so that the capacity constraint corresponds to a cardinality constraint), and the extra profit has the form $p_{ij} = a_i b_j$, with $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ positive integer vectors. They designed a polynomial time algorithm that provides a constant approximation ratio of 4.5, and performed computational experiments on randomly generated instances with different correlation structures.

The *Quadratic Knapsack Problem with Conflict Graphs* (QKPCG) is a generalization of the QKP in which, as for the *knapsack problem with conflict graph* (see Section

2.9), a given undirected graph $G = (V, E)$ defines the pairs of incompatible items that cannot be simultaneously selected. Shi, Wu, and Meng [448] presented a large neighborhood search algorithm for the QKPCG. Dahmani and Hifi [134] proposed a descent-based heuristic that combines local search and reactive search for diversification. The approach obtained better results than those in [448]. Dahmani et al. [136] developed a method based on particle swarm combined with local search, and further improved the results in [134].

The *Quadratic Knapsack Problem with Multiple Knapsack Constraints* (MdQKP) is a generalization of the QKP in which, as for the MdKP (see Section 3.3), the knapsack has d different capacities. The corresponding quadratic mathematical model is thus defined by (3.16), (3.6), and (3.7). The MdQKP was introduced by Wang, Kochenberger, and Glover [473], who compared the quadratic model with three alternative linearizations, solved with CPLEX on test instances with up to 800 variables. The computational experiments showed that, for large instances, the quadratic model outperforms by a large margin the linearized ones both in terms of solution quality and solution time.

Quadratic multiple knapsack problem

The *Quadratic Multiple Knapsack Problem* (QMKP) is a generalization of the QKP in which, as for the MKP (see Section 3.2), one is given m knapsacks with capacities c_i $(i = 1, \dots, m)$ and n items having profit p_j and weight w_j $(j = 1, \dots, n)$. In addition, as for the QKP, each pair of distinct items i and j gives an extra profit p_{ij} if both items are assigned to the same knapsack. The objective is to select m disjoint subsets of items so that each subset is assigned to a knapsack whose capacity is no less than the total weight of its items, and the total profit (sum of the profits of the selected items and of the pairwise profits of items assigned to the same knapsack) is maximized. Formally

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} + \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \sum_{i=1}^{m} p_{kj} x_{ij} x_{ik} \tag{3.19}$$

$$\text{s.t. } (3.2) - (3.4).$$

Exact solution. The first exact solution approach to the QMKP is due to Bergman [49]. He introduced an exponential-size ILP model solved through a of a *Branch-and-Price* (B&P) algorithm. Computational tests showed that the method outperforms commercial optimization solvers. The B&P algorithm was also applied to the automated table event seating problem (a variant of the QMKP in which each item is required to be placed in some knapsack). Galli et al. [194] studied polynomial size formulations and upper bounds for the QMKP. They proposed surrogate and Lagrangian relaxations, providing theoretical properties and dominances, as well as extensive computational experiments. An effective B&B algorithm for the QMKP was recently presented by Fleszar [179].

Heuristics. García-Martínez, Rodriguez, and Lozano [197] and García-Martínez et al. [198] introduced two metaheuristics for the QMKP, one based on strategic oscillation, one consisting of a Tabu-enhanced greedy approach. They showed that their methods are competitive with the state-of-the-art algorithms. Chen and Hao [103] presented an algorithm that combines a threshold based exploration with a descent based improvement and successfully compared it with the methods in [197, 198]. Chen, Hao, and Glover [106] proposed an evolutionary path relinking approach and

showed that it successfully competes with the algorithms in [197, 198], and [103]. Peng et al. [395] proposed an ejection chain method with an adaptive perturbation mechanism and positively compared it with the algorithms in [197, 198]. Qin et al. [411] presented a Tabu search based on local search algorithms that consider both feasible and infeasible solutions, and favourably compared it with the previous literature on the QMKP.

Variants and generalizations

Olivier, Lodi, and Pesant [388] introduced the *Quadratic Multiknapsack Problem with Conflicts and Balance Constraints* (QMPCBC) as a generalization of the QMKP which includes pairwise conflicts between items and asks for packing all the items into the knapsacks in such a way that the load of every knapsack is within a lower and an upper bound. They proposed a constraint programming model and two quadratic integer programming formulations. The best computational results were obtained by a column generation algorithm that solves one of the quadratic formulations and adopts a constraint program for the pricing subproblem.

Saraç and Sipahioglu [435] introduced the *Generalized Quadratic Multiple Knapsack Problem* (GQMKP), which extends the QMKP as follows: (i) the items are partitioned into classes, and a setup cost has to be paid when at least one item from a class is assigned to a knapsack; (ii) additional assignment constraints impose that some items and classes cannot be assigned to certain knapsacks; (iii) the profits depend on both items and knapsacks. The authors introduced a mathematical model and proposed two heuristic approaches: a genetic algorithm and a hybrid method which combines the former with a subgradient approach. Chen and Hao [104] developed a memetic algorithm which combines a crossover operator with multi-neighbourhood simulated annealing, and successfully compared it with the methods in [435]. Avci and Topaloglu [33] developed a multi-start iterated local search algorithm embedding an adaptive perturbation mechanism and a Tabu list. The proposed approach was favourably compared with the previous literature. Adouani, Jarboui, and Masmoudi [4] introduced a new ILP model and a matheuristic approach which integrates variable local search techniques, an adaptive perturbation mechanism, and quadratic integer programming. A hybrid evolutionary search approach that embeds a number of metaheuristic techniques was recently presented by Zhou, Hao, and Wu [505]. Experimental results show that this is the algorithm providing so far the best results in the literature for the GQMKP.

3.6 Other knapsack problems

3.6.1 Online knapsack problems

In the *Online Knapsack Problem* (OKP), the information on the items is given one by one during the packing process. More specifically, only after a decision is made on a given item, the information on the next one is then available and it has to be decided immediately whether to pack this item (and keep it in the knapsack forever) or discard it (without the possibility of packing it again). As for the offline version (KP01), the objective of the OKP is to maximize the profit under the capacity constraint.

An online algorithm is said to be *r-competitive* if there exists a constant r (the *competitive ratio*) such that, for any possible incoming sequence of items, the ratio between its performance and the performance achieved by an exact offline algorithm

is bounded by r. Han, Kawase, and Makino [226] provided a simple randomized 2-competitive algorithm for the online version of the SSP (referred in the literature as the *unweighted* case, i.e., the case in which each item has weight equal to its profit), and showed that it is the best possible. Böckenhauer et al. [55] analyzed complexity issues of the OKP.

Other minor variants of the OKP have been studied. Han and Makino [228] showed that no algorithm has a bounded competitive ratio for the minimization version of the OKP, in which the goal is to find a subset of items such that the sum of their weights is at least equal to the knapsack capacity and the sum of their profits is minimized. Cygan, Jeż, and Sgall [133] investigated the online variant of the MKP. Thielen, Tiedemann, and Westphal [464] considered the OKP *with incremental capacity*, in which the knapsack capacity increases by a constant amount in each time period. Navarra and Pinotti [384] addressed the *knapsack of unknown capacity problem*, an online variant of the KP01 where the capacity c of the knapsack is unknown and it is revealed during the packing process. When the knapsack capacity is revealed, no other item can be inserted and the first item in the sequence of packed items that does not completely fit in the knapsack is removed, as well as all the subsequent inserted items.

Iwama and Taketomi [279] introduced the *Online Removable Knapsack Problem* (ORKP), a version of the OKP in which the already stored items can be removed during the packing process in order to accept the current one. They provided an r-competitive algorithm for the unweighted case, where $r = (1+\sqrt{5})/2 \approx 1.618$. For the general case, they showed that no algorithm can have a bounded competitive ratio. They also provided some tight bounds for the ORKP with resource augmentation (see [280]), which allows online algorithms to use more capacity than offline algorithms. Han et al. [229] addressed the ORKP under a convex function $f(\cdot)$ such that $p_j = f(w_j)$ $(j = 1, \ldots, n)$. They proposed a greedy online algorithm with competitive ratio 2 and an improved online algorithm with competitive ratio $5/3$. Han, Kawase, and Makino [226] introduced a simple randomized 2-competitive algorithm and provided a lower bound $1 + 1/e$ for the competitive ratio. For the unweighted case, they proposed a randomized $10/7$-competitive algorithm and proved that there exists no randomized online algorithm with competitive ratio less than $5/4$. Han, Chen, and Makino [224] studied the ORKP under a concave function $f(\cdot)$ such that $p_j = f(w_j)$ $(j = 1, \ldots, n)$, provided a lower bound $\max\{q, \frac{f'(0)}{f(1)}\}$, and showed that, for the concave case, the competitive ratio of the online algorithm proposed in [279] is $\frac{f'(0)}{f(1/q)}$. In addition, they proposed another online algorithm with a competitive ratio $\frac{f'(0)}{f(1)} + 1$.

Several variants of the ORKP have been also considered in the literature. Han and Makino [227] studied the ORKP with limited cuts, in which items are allowed to be cut/packed at most $k \geq 1$ times. Han, Kawase, and Makino [225] addressed the unweighted case of the ORKP with removal cost. In this variant, if an item is packed into the knapsack making the current total weight to exceed the knapsack capacity, then some items in the knapsack have to be removed, paying some additional removal costs. The aim is to maximize the sum of the total weight of the packed items minus the total removal cost. Han, Kawase, and Makino [225] provided competitive ratios for the cases of proportional (to the weights) or unit removal costs. Han and Makino [228] addressed the minimization version of the ORKP and obtained the following results: they provided an 8-competitive deterministic algorithm, proposed

a $2e$-competitive randomized algorithm, derived a lower bound of 2 for deterministic algorithms, and proposed a 1.618-competitive deterministic algorithm for the unweighted case, showing that this is the best possible.

3.6.2 Multiobjective knapsack problems

The *Multiobjective Knapsack Problem* (MOKP) is a generalization of the KP01 in which each item has $t \geq 2$ non-negative integer profits p_{jk} $(j = 1, \ldots, n; k = 1, \ldots, t)$ and the total profit objectives are maximized in the Pareto sense, i.e.,

$$\max(\sum_{j=1}^{n} p_{j1}x_j, \ \ldots, \ \sum_{j=1}^{n} p_{jt}x_j). \tag{3.20}$$

Most results in the literature concern the case in which $t = 2$, usually called the *Biobjective Knapsack Problem* (BOKP). Both the MOKP and the BOKP generalize the KP01 and hence are $\mathcal{NP}$-hard. We refer the reader to the fundamental book by Ehrgott [169] on multicriteria optimization for a thorough discussion on the complexity status of the problems. More recently, a specific survey on the MOKP has been presented by Lust and Teghem [349]. In the following, we mostly discuss papers appeared after its publication (2012).

Exact solution. A number of exact methods for the BOKP have been proposed. Rong et al. [422, 425] presented a DP algorithm and evaluated different reduction techniques aimed at decreasing the number of states explored by the algorithm. Algorithmic improvements of these methods were proposed by Figueira et al. [176] and Rong and Figueira [423]. The use of special data structures for the BOKP was evaluated by Correia, Paquete, and Figueira [122]. Daoud and Chaabane [143] proposed a reduction strategy for the BOKP and showed, through computational experiments, its effectiveness in reducing the search space.

Approximation and heuristics. Cerqueus et al. [94] studied the effectiveness of various branching heuristics for the BOKP and proposed a dynamic adaptive branching strategy. Their computational experiments showed however that the proposed techniques are not useful in practice, as they require high CPU times. The MOKP has been also tackled through metaheuristics: we mention in particular a quantum-inspired evolutionary algorithm by Lu and Yu [346], a Bayesian estimation of distribution algorithm by Martins et al. [367], and a swarm intelligence approach by Zouache, Moussaoui, and Ben Abdelaziz [507].

Variants and generalizations. Kozanidis [314] proposed an algorithm for the biobjective version of the multiple-choice knapsack problem studied in Kozanidis and Melachrinoudis [315] (see Section 2.7). Rong, Klamroth, and Figueira [426] proposed hybrid solution approaches for a variant of the MOKP which also includes so-called k-min objectives aiming at the maximization of the k-th smallest objective coefficients. Castillo-Zunino and Keskinocak [92] studied a multiple version of the MOKP, in which the items are partitioned into groups and either all or none of the items from a group are assigned.

Multidimensional multiobjective knapsack problems

Few results on the multidimensional version of multiobjective knapsack problems appeared after the publication of the survey by Lust and Teghem [349]. Cerqueus, Przybylski, and Gandibleux [93] studied the bidimensional version of the BOKP:

they introduced a convex surrogate upper bound and presented two exact algorithms and a heuristic approach for its computation. Shah and Reed [445], Mavrotas, Florios, and Figueira [369], and Luo, Ji, and Zhu [348] presented extensive computational evaluations of various metaheuristic approaches.

3.7 Conclusions

After over sixty years of intensive research, knapsack problems are a lively research area in a number of scientific fields. This work covers the recent developments in the richest of such fields, combinatorial optimization. The two parts of the survey list over 450 different papers, mostly appeared after 2004, the publication year of the latter of the two classical books specifically dedicated to these topics.

Chapter 2 covers the classical single knapsack problems and their many variants and generalizations: subset sum, item types, setup, multiple-choice, conflict graphs, precedences, sharing, bilevel, robust, among others. The section on extensions and generalizations provides pointers to the variants that are especially attractive from the point of view of possible future investigations.

Chapter 3 is mainly devoted to multiple, multidimensional (vector and geometric), and quadratic knapsack problems, but also contains a succinct treatment of online and multiobjective knapsack problems. We have mentioned, in the concluding remarks of Chapter 2, a number of relevant real-world applications for knapsack problems. In this chapter, we noticed other relevant applications, arising in emergency military and humanitarian contexts (the MKAP), steel production (the MKPCC), machine scheduling (the MKP), cutting of material (the 2D-KPGC), and container loading (the 3D-KP), among others. We believe that new applications will continue to appear in the next years.

Recent research has produced a good number of new exact algorithms, but there is still room for improvement on many problem variants. While some knapsack problems, as the KP01 and the SSP, are solved extremely well by the algorithms available in the literature, and difficult instances involve very large coefficients and a huge number of items, for other problems, as the 2D-KP, instances with just 30 items are still unsolved to proven optimality despite decades research. We did not include here a section on hot topics as the publication dates of most reviewed articles (over 70% of the articles in the bibliography appeared in the last decade) indicate that all the main variants and generalizations of these problems still undergo intensive analysis and hence are attractive research areas to researchers interested in pursuing investigations in this fascinating area.

Chapter 4

Tool switching problems with tool order constraints*

This chapter addresses four different variants of the well-known Tool Switching Problem (ToSP). For each variant, we discuss its complexity and propose a mathematical formulation. Motivated by a real-world application in the colour printing industry, the third and fourth variants introduce a novel requirement into ToSP: the tool order constraint. Under this requirement, during the processing of each job, the selected tools must be sorted along the slot sequence in the machine, and the machine will use them for processing the job applying the tools in that order. We show that the new problem variants are $\mathcal{NP}$-hard even when the job sequence is given as part of the input and the setup times are binary. We solve them by using dedicated arc flow models. We evaluate the effectiveness of the models on several instances that were generated with the aim of covering different scenarios of interest. Our code finds proven optimal solutions for most of the instances with up to 30 jobs, 60 tools and 10 slots.

4.1 Introduction

Consider the following general class of combinatorial optimization problems. Let $J = \{1, \ldots, n\}$ be a set of jobs to be processed sequentially on a single machine, $C = \{1, \ldots, m\}$ be a set of different tools, and $K = \{1, \ldots, k\}$ be a set of slots available on the machine and used to allocate the tools. The machine can execute one job at a time, and the execution of a job cannot be interrupted until it is completed. Each job $j \in J$ requires a subset of tools $C_j \subseteq C$ to be loaded in the magazine of the machine before starting the execution of the job. We assume that $|C_j| \leq |K|$ holds for all $j \in J$. In practical situations, the magazine cannot hold all tools at once (i.e., $|C| > |K|$); hence, some tool switches may be necessary when performing two consecutive jobs. During the processing of a job, the machine sequentially uses the involved tools, stored in the magazine, in accordance with its slot sequence. After finishing the execution of a job and before starting the execution of the next job, one must decide for each slot whether to preserve the current tool or to switch to a different one. Replacing the tool in a slot with another tool out of the magazine is a tool switching operation. Moving a tool from one slot to another slot in the magazine is considered in our context as two tool switching operations. Tool switching operations may take a long time, with the effect of significantly raising the setup time, while preserving a tool in a slot does not require any setup time. Since tool switches imply a reduction

*The results of this chapter were submitted for publication to international journal: M. Iori, A. Locatelli, M. Locatelli, and J. J. Salazar-González. "Tool switching problems with tool order constraints". *Technical report*, (2022).

Table 4.1: Computational complexity of ToSP variants.

Variant	CUF	GUF	GOF	GOV	CUV	GUV	COF	COV
Complexity	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{NP}$-hard	Strongly $\mathcal{NP}$-hard	$\mathcal{NP}$-hard	Strongly $\mathcal{NP}$-hard	unknown	unknown
Reference	Section 4.2	Section 4.3	Section 4.4	Section 4.5	Calmels [73]	Calmels [73]	unknown	unknown

in productivity, minimizing them is desirable. The aim is to find the job sequence and the tools to be in each slot of the magazine at each time. This problem can be seen as a variant of the well-known Tool Switching Problem (ToSP, see, e.g., Calmels [73]).

In the ToSP, for each job $j \in J$, the order of the tools along the slot sequence in the magazine is irrelevant, and one just decides the $|K|$ tools in the magazine so that C_j is loaded. In this chapter, we address four problem variants of increasing difficulty. The third and fourth variants address the case where the tools required by a job must be placed in the machine in a prespecified order, which means that they must be sorted along the slot sequence so that the machine will use them in the required order for the job execution.

To distinguish between the problem variants, we use an acronym composed of three letters:

1. as the initial letter, we use C and G to denote the problem with *constant* and *general* (i.e. non-uniform) non-negative setup times, respectively;

2. as the second letter, we use U when the set C_j is *unordered* for each job $j \in J$, i.e., the tools of job j can be placed in the slots in any order; we use O, instead, when the set C_j is *ordered*, i.e., the tools of job j are to be placed in the slots in the order specified in C_j, although not necessarily in consecutive slots;

3. as the final letter, we use F and V when the sequence of jobs is *fixed* and *variable*, respectively. "Variable" means that the job sequence must be determined as part of the combinatorial problem.

Observe that we have introduced eight different variants, each corresponding to a different three-letter acronym. Table 4.1 reports all the variants and their complexity. The complexity of CUV-ToSP and GUV-ToSP are known from the literature (see, e.g., Calmels [73]), while the complexity of CUF-ToSP, GUF-ToSP, GOF-ToSP, and GOV-ToSP are stated in this chapter. To the best of our knowledge, the complexity of COF-ToSP and COV-ToSP is still unknown.

The first and simplest problem is CUF-ToSP, where we assume constant setup times for tool switching operations and a fixed job sequence (i.e., job $j \in J$ is executed in position j of the schedule). The second problem is GUF-ToSP, where we assume non-uniform setup times for tool switching operations and fixed job sequence. The third addressed problem is GOF-ToSP, in which we assume non-uniform setup times for tool switching operations and a fixed job sequence. In addition, in GOF-ToSP we assume that the tools required by a job must be sequentially used by the machine in a given order, which means that they must be sorted along the slot sequence so that the machine will use them in the required order. The fourth and most difficult problem is GOV-ToSP where the job sequence is not fixed. We also mention that a further variant (CUV-ToSP) is known in the literature as the Job Sequencing and Tool Switching Problem (SSP). Such a problem has been addressed in different papers, including Crama et al. [125], Laporte, Salazar-González, and Semet [329], and

Calmels [73]. Crama et al. [125] provided a formal proof that SSP (aka CUV-ToSP) is $\mathcal{NP}$-hard. On the other hand, the first problem addressed in this chapter, CUF-ToSP, can be solved in polynomial time by means of a Keep Tool Needed Soonest policy (see Tang and Denardo [460]). For this problem, in Section 4.2 we show that the binary linear programming formulation of SSP proposed in Tang and Denardo [460] is a perfect formulation, so that we also provide a different proof that CUF-ToSP can be solved in polynomial time.

The second problem addressed in this chapter is GUF-ToSP, the generalization of CUF-ToSP in which the setup time for tool switching operations is non-uniform. In Section 4.3, we show that GUF-ToSP can be solved in polynomial-time by formalizing it as a minimum-cost flow problem (see, e.g., Bazaraa, Jarvis, and Sherali [40]) for which we provide a perfect arc flow formulation. Crama et al. [127] proved that GUF-ToSP with non-uniform tool sizes (i.e., the generalization of GUF-ToSP in which the number of slots occupied by a tool does not need to be the same for all tools) is strongly $\mathcal{NP}$-complete.

The third problem is GOF-ToSP, obtained by introducing the tool order constraint into GUF-ToSP. GOF-ToSP is the main motivation of this chapter since the problem originates from a real-world application arising in a food packaging company located in the city of Reggio Emilia (Iori, Locatelli, and Locatelli [275]). The specific technology used in this application is the flexographic printing (see, e.g., Kipphan [305]). A flexographic printer machine can simultaneously accommodate a limited number of colours (which can be treated as tools) and must sequentially perform a finite set of jobs. Each job is associated with a subset of colours that the printer machine must have stored in its magazine before starting the execution. During the processing of a job, the printer machine sequentially applies the involved colours, stored in the magazine, on the printed material in accordance with its slot sequence. In many industrial printing technologies, the overlapping order of colours plays a crucial role in achieving print quality (see, e.g., Boora and Verma [57]) and a violation of the chromatic order occurring during the printing of a job may lead to colour deviations, colour mixing, and reverse printing failures in the final products. Thus, the colours must be sorted along slots in the magazine so that the printer will apply them in the required order, which is fixed in advance. In addition, non-uniform setup times for colour switching operations are caused by the fact that the process to wash a slot depends directly on the specific colours involved. For instance, if the process requires switching a black ink cartridge to a white one, complete cleaning of the corresponding slot is necessary to preserve the white purity, with the effect of a significant increase in the setup time. On the other hand, if the process requires changing a slot from pale green ink to dark green ink, then the setup time is smaller because the dark green ink is not easily altered by the residuals of the pale green ink and, therefore, the relative slot needs only partial washing.
The real-world problem faced by the company generalizes the GOF-ToSP to a great extent (as it includes parallel heterogeneous machines, release and due dates, and several additional complicating features) and was solved by Iori, Locatelli, and Locatelli [275] using a constructive heuristic. Section 4.4 states the $\mathcal{NP}$-hardness of GOF-ToSP and presents an arc flow model. Moreover, we provide a preprocessing method that allows us to potentially reduce the number of jobs.

The fourth problem addressed in this chapter is GOV-ToSP, the generalization of GOF-ToSP in which the sequence of jobs is not given in advance. In Section 4.5, we state that GOV-ToSP is strongly $\mathcal{NP}$-hard and provide a descriptive arc flow model.

In this chapter, we introduce three novel arc flow formulations in the area of

ToSP. An arc flow formulation is a network flow formulation where decision variables correspond to non-negative flows on each arc of the network (see Lima et al. [342] for a recent survey). Arc flow formulations have been recently applied with success to many important combinatorial optimization problems, such as bin packing and cutting stock (see, e.g., Delorme and Iori [159] and Martinovic, Scheithauer, and Carvalho [364]), skiving stock (see, e.g., Martinovic and Scheithauer [363] and Martinovic et al. [365]), and parallel machine scheduling problems (see, e.g., Kramer, Dell'Amico, and Iori [317] and Kramer, Iori, and Lacomme [318]). To the best of our knowledge, no previous papers on arc flow formulations have been published in the area of ToSP.

In Section 4.6, we perform extensive computational experiments to evaluate the effectiveness of the arc flow models proposed for GOF-ToSP and GOV-ToSP. Such experiments have been carried out on instances that have been generated starting from realistic data features and with the aim of covering different scenarios of interest. Finally, in Section 4.7 we draw some concluding remarks and provide some hints for future research.

4.2 CUF-ToSP

In CUF-ToSP, the order of the tools in the magazine is irrelevant, the job sequence is fixed in advance (thus, job j is executed in position j), and all tool switches have identical setup time, which, w.l.o.g., can be assumed equal to one (i.e., $c_{uv} = 1$ if $u \neq v$ and 0 if $u = v$). Thus, CUF-ToSP aims at determining the subsets of tools loaded in the magazine during the processing of each job $j \in J$ to minimize the total number of tool switches. For the notation it is convenient to extend J with a dummy job 0 with no need of tools (i.e. $C_0 = \emptyset$) obtaining $J = \{0, 1, \ldots, n\}$.

4.2.1 Two-index formulation for CUF-ToSP

The formulation that we propose for CUF-ToSP requires two sets of decision variables: a binary decision variable y_j^u taking the value 1 if tool u is loaded on the magazine during the processing of job j, and 0 otherwise; and a binary decision variable z_j^u taking the value 1 if tool u is switched with another tool loaded on the magazine before starting job $j \in J$. In other words, $z_j^u = 1$ corresponds to the occurrence of a tool switch. The formulation is then:

$$\min \sum_{u \in C} \sum_{j \in J \setminus \{0\}} z_j^u \tag{4.1}$$

$$\sum_{u \in C} y_j^u \leq |K| \qquad j \in J \tag{4.2}$$

$$y_j^u = 1 \qquad j \in J,\ u \in C_j \tag{4.3}$$

$$y_j^u - y_{j-1}^u \leq z_j^u \qquad j \in J \setminus \{0\},\ u \in C \tag{4.4}$$

$$y_j^u \in \{0, 1\} \qquad j \in J,\ u \in C \tag{4.5}$$

$$z_j^u \in \{0, 1\} \qquad j \in J \setminus \{0\},\ u \in C. \tag{4.6}$$

The objective function (4.1) minimizes the total number of tool switches. Constraints (4.2) guarantee that the magazine capacity is never exceeded, while constraints (4.3) guarantee that, during the processing of job $j \in J$, each tool $u \in C_j$ takes exactly

a slot. Constraints (4.4) impose a switch operation when tool u is in the magazine when processing job j and it is not in the magazine when processing job $j-1$.

We remark that the model above is the model for SSP proposed in Tang and Denardo [460] after fixing in that model the values of the binary variables that define the sequence of jobs.

We recall that a perfect formulation of a set $S \subseteq \mathbb{R}^n$ is a linear system of inequalities $Ax \leq b$ such that the convex hull of S coincides with $\{x \in \mathbb{R}^n : Ax \leq b\}$. The following theorem proves that CUF-ToSP is solvable in polynomial time.

Theorem 4.2.1. *A perfect formulation of the set defined by constraints (4.2)–(4.6) is obtained by relaxing the binary requirements $y_j^u, z_j^u \in \{0,1\}$ into $y_j^u, z_j^u \in [0,1]$.*

Proof. We can restrict the attention to constraints (4.2) and (4.4), since all other constraints can be viewed as bound constraints for the variables. The right-hand side vector of these constraints has integer coordinates, so, to prove the result, we only need to prove that the corresponding constraint matrix is *Totally Unimodular* (TU). We prove this by exploiting Ghouila-Houri's characterization of TU matrices (see Ghouila-Houri [202]). Namely, a matrix A is TU if and only if for each subset I of its rows, it is possible to partition it into two subsets I_1, I_2 such that for each column j of the matrix it holds that

$$\left| \sum_{i \in I_1} A_{ij} - \sum_{i \in I_2} A_{ij} \right| \leq 1. \tag{4.7}$$

The matrix associated with constraints (4.2) and (4.4) has $n+1 = |J|$ rows, denoted by $r_0, \ldots, r_n$ and corresponding to the first set of constraints, and n collections of rows $S_1, \ldots, S_n$, where each collection S_j is made up of $m = |C|$ rows (one for each $u \in C$). The columns of the matrix are subdivided into $n+1$ collections $C_0, \ldots, C_n$, where each collection C_j is made up by m columns (one for each $u \in C$), and a further set of n collections $D_1, \ldots, D_n$, where each collection is also made up of m columns. The first set of collections of columns corresponds to the set of variables y_j^u, $j \in J$, $u \in C$, while the second set of collections corresponds to the set of variables z_j^u, $j \in J \setminus \{0\}$, $u \in C$.

Now we immediately notice that condition (4.7) is always satisfied by the columns in the collections $D_1, \ldots, D_n$ for any possible partition I_1, I_2, because each column in these collections contains a single non-zero element (equal to -1). Therefore, we restrict the attention to columns in the collections $C_0, \ldots, C_n$. We introduce a procedure, described by Algorithm 1, to build a partition for a given subset I of the rows. For each $j \in \{0, 1, \ldots, n\}$, the non-zero entries of columns C_j lie in row r_j (coefficient $+1$), rows S_j (coefficient $+1$), and rows S_{j+1} (coefficient -1). Therefore, for these columns we need to prove that for the resulting partition it holds that

$$\left| \sum_{i \in I_1 \cap [\{r_j\} \cup S_j \cup S_{j+1}]} A_{ij} - \sum_{i \in I_2 \cap [\{r_j\} \cup S_j \cup S_{j+1}]} A_{ij} \right| \leq 1, \tag{4.8}$$

where we set $S_0 = S_{n+1} = \emptyset$. First we notice that, with the given initialization of I_1 and I_2 at steps 1-2 of Algorithm 1, the columns in C_0 satisfy (4.8). Now, for $j \geq 1$ and columns in C_j, we have that if $r_j \notin I$, then rows $S_j \cap I$ and $S_{j+1} \cap I$ are placed in the same member of the partition (either I_1, step 9, or I_2, step 17) in order to fulfill (4.8). Instead, if $r_j \in I$, then we can fulfill (4.8) by placing row r_j and all rows $S_{j+1} \cap I$ in the same member of the partition, while all rows $S_{j+1} \cap I$ are placed in the other member of the partition (steps 6 and 9). □ □

Algorithm 1: Procedure to partition rows in I in such a way that (4.7) is fulfilled.

Input: I

Set $I_1 = I \cap [\{r_0\} \cup S_1]$;
Set $I_2 = \emptyset$;
for $j \leftarrow 1$ ***to*** n **do**
  if $I \cap [\{r_{j-1}\} \cup S_j] \subseteq I_1$ **then**
    if $r_j \in I$ **then**
      $I_2 = I_2 \cup \{I \cap [\{r_j\} \cup S_{j+1}]\}$;
    end
    else
      $I_1 = I_1 \cup \{I \cap [\{r_j\} \cup S_{j+1}]\}$;
    end
  end
  else
    if $r_j \in I$ **then**
      $I_1 = I_1 \cup \{I \cap [\{r_j\} \cup S_{j+1}]\}$;
    end
    else
      $I_2 = I_2 \cup \{I \cap [\{r_j\} \cup S_{j+1}]\}$;
    end
  end
end
return I_1, I_2;

We illustrate Algorithm 1 over an example. For the instance with the constraint matrix given in Table 4.2, the procedure is illustrated over the subset of the rows obtained by omitting row r_2 and the second row in S_2 (we denote by S_2' such a subset of S_2). We first consider the columns in C_0 and we assign row r_0 and both rows in S_1 to I_1. Next, we move to the columns in C_1. Since row $r_1 \in I$, we assign rows $\{r_1\} \cup S_2'$ to I_2. Next, we move to the columns in C_2. Since $r_2 \notin I$, we assign the two rows in S_3 to I_2. Now, we move to the columns in C_3. Since $r_3 \in I$, we assign rows $\{r_3\} \cup S_4$ to I_1. Finally, we move to the columns in C_4. Since $r_4 \in I$, we need to assign the single row r_4 to I_2. In conclusion, we found the following partition

$$I_1 = \{r_0, r_3\} \cup S_1 \cup S_4, \quad I_2 = \{r_1, r_4\} \cup S_2' \cup S_3.$$

fulfilling (4.7) for all columns.

4.3 GUF-ToSP

In GUF-ToSP, the order of the tools in the magazine is irrelevant and the job sequence is fixed in advance (thus, job j is executed in position j). We are also given a general setup time c_{uv} required to switch from tool u to tool v. More specifically, for all $u, v \in C$, the setup time c_{uv} is always non-negative and it is negligible if the two tools u and v coincide (i.e., $c_{uu} = 0$ for all $u \in C$). Moreover, we assume that the triangular inequality holds for the setup times (i.e., $c_{uw} \leq c_{uv} + c_{vw}$ for all $u, v, w \in C$). GUF-ToSP consists of determining the assignment of the tools to the slots of the magazine during the processing of each job $j \in J$ to minimize the total setup time.

Table 4.2: Constraint matrix for an instance with $n = 4$ (i.e., 5 jobs) and $m = 2$ tools.

	C_0		C_1		C_2		C_3		C_4		D_1		D_2		D_3		D_4	
r_0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r_1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r_2	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
r_3	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
r_4	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
S_1	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0
	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
S_2	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0
	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0
S_3	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0
	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0
S_4	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0
	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1

4.3.1 Four-index Arc Flow Formulation for GUF-ToSP

Given a job sequence J, a solution of GUF-ToSP can be represented as $|K|$ paths from a source to a destination visiting disjoint nodes of a specific network $\mathcal{N}$. Let $J^* = \{0, 1, \ldots, n+1\}$, where the two additional jobs 0 and $n+1$ are two dummy jobs with $C_0 = C_{n+1} = \{0\}$, where 0 is a dummy tool, and represent the source and destination node, respectively.
We consider the network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ with set of nodes $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$ and set of directed arcs $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \{((0,0)(0,n+1))\}$, where:

- $\mathcal{V}_1$ is composed of all couples (u, j) with $j \in J^*$ and $u \in C_j$;
- $\mathcal{V}_2$ is composed of all copies $(u, j)'$ of the nodes $(u, j) \in \mathcal{V}_1$ such that $j \in J$;
- $\mathcal{A}_1$ is composed of all arcs from the source node $(0, 0)$ to nodes $(u, i)' \in \mathcal{V}_2$, i.e.:

$$\mathcal{A}_1 = \{((0,0), (u,i)') : \ (u,i)' \in \mathcal{V}_2\};$$

- $\mathcal{A}_2$ includes, for each $i \in J$, all arcs from a node $(u, i) \in \mathcal{V}_1$ to its copy node $(u, i)' \in \mathcal{V}_2$, i.e.:

$$\mathcal{A}_2 = \{((u,i), (u,i)') : \ (u,i) \in \mathcal{V}_1, \ i \in J\};$$

- $\mathcal{A}_3$ is composed of all arcs from node $(u, i)' \in \mathcal{V}_2$ to node $(v, j) \in \mathcal{V}_1$ such that $i \in J$, $j \in J^* \setminus \{0\}$, and $j > i$, i.e.:

$$\mathcal{A}_3 = \{((u,i)', (v,j)) : \ (u,i)' \in \mathcal{V}_2, \ i \in J, \ (v,j) \in \mathcal{V}_1, \ j \in J^* \setminus \{0\}, \ j > i\}.$$

From the source node $(0, 0)$ an amount of flow $|K|$ is sent toward the sink node $(0, n+1)$. A cost function $w : \mathcal{A} \to \mathbb{R}^+$ maps each arc $((u,i)', (v,j)) \in \mathcal{A}_3$ to a cost c_{uv} (i.e., the setup time required for switching tool u of job i with tool v of job j), and all arcs in $\mathcal{A}_1 \cup \mathcal{A}_2$ to a null cost.

Now, we can formulate the GUF-ToSP as a minimum-cost flow problem on $\mathcal{N}$. The decision variables of the problem are the arc flows, which are denoted by ϕ_e for each arc $e \in \mathcal{A}$. In particular:

- variable $\phi_{((0,0)(u,i)')}$ is equal to 1 if tool u of job i is the first tool placed in a slot;

- variable $\phi_{((0,0)(0,n+1))}$ is the only integer and not binary variable, and indicates the number of slots that are never used;
- variable $\phi_{((u,i)',(v,j))}$, $j > i$, is equal to 1 if tool v of job j replaces tool u of job i in one of the slots;
- each variable $\phi_{((u,i),(u,i)')}$ is, in fact, a constant equal to 1 that is used to impose that each required tool takes exactly one slot.

The resulting mathematical model is:

$$\min \sum_{((u,i)',(v,j))\in\mathcal{A}_3} c_{uv}\phi_{((u,i)',(v,j))} \tag{4.9}$$

$$\sum_{\substack{(v,j):\\((0,0),(v,j))\in\mathcal{A}_1}} \phi_{((0,0),(v,j))} = |K| \tag{4.10}$$

$$\phi_{((0,0),(0,n+1))} + \sum_{\substack{(v,j):\\((v,j)',(0,n+1))\in\mathcal{A}_3}} \phi_{((v,j)',(0,n+1))} = |K| \tag{4.11}$$

$$\phi_{((0,0),(u,i))} + \sum_{\substack{(v,j):\\((v,j)',(u,i))\in\mathcal{A}_3}} \phi_{((v,j)',(u,i))} = \phi_{((u,i),(u,i)')} \qquad (u,i)\in\mathcal{V}_1 \tag{4.12}$$

$$\phi_{((u,i),(u,i)')} = \sum_{\substack{(v,j):\\((u,i)',(v,j))\in\mathcal{A}_3}} \phi_{((u,i)',(v,j))} \qquad (u,i)\in\mathcal{V}_1 \tag{4.13}$$

$$\phi_{((u,i),(u,i)')} = 1 \qquad ((u,i),(u,i)')\in\mathcal{A}_2 \tag{4.14}$$

$$\phi_{((0,0),(u,i))} \in \{0,1\} \qquad ((0,0),(u,i))\in\mathcal{A}_1 \tag{4.15}$$

$$\phi_{((u,i)',(v,j))} \in \{0,1\} \qquad ((u,i)',(v,j))\in\mathcal{A}_3 \tag{4.16}$$

$$\phi_{((0,0),(0,n+1))} \in \mathbb{Z}_+. \tag{4.17}$$

The objective function (4.9) minimizes the total setup time. Constraint (4.10) imposes that there is an exit flow equal to $|K|$ from the source node. Constraint (4.11) imposes that there is an entering flow equal to $|K|$ into the sink node. Constraints (4.12) together with constraints (4.14) guarantee that, during the processing of job $i \in J$, each tool required by i takes exactly a slot. Moreover, constraints (4.12) together with constraints (4.13) guarantee flow conservation at all intermediate nodes (u,i) and $(u,i)'$. Note that binary constraints are imposed for all variables, except $\phi_{((0,0),(0,n+1))}$, which is required to be a nonnegative integer.

Theorem 4.3.1. *A perfect formulation of the set defined by constraints* (4.10)-(4.17) *is obtained by relaxing the binary requirements* $\phi_e \in \{0,1\}$, *for each* $e \in \mathcal{A} \setminus \{((0,0)(0,n+1))\}$ *into* $\phi_e \in [0,1]$, *and the integer requirement* $\phi_{((0,0),(0,n+1))} \in \mathbb{Z}_+$ *into* $\phi_{((0,0),(0,n+1))} \geq 0$.

Proof. The proof easily follows from the observation that the underlying constraint matrix is TU (see, e.g., Nemhauser and Wolsey [385]) and all right-hand sides are integer values. □

Note that CUF-ToSP is the sub-case of GUF-ToSP where all weights c_{uv} are equal to one. Thus, in that case, the above formulation is a perfect formulation alternative to (4.2)-(4.6), which is, however, more compact.

4.4 GOF-ToSP

Motivated by a real-world application in the colour printing industry (see Section 4.1), we assume that tools must be sorted along the slot sequence in a required order during the processing of each job. To model this aspect, we introduce a precedence relation $\prec^j$ in each set $C_j \subseteq C$. We read $u \prec^j v$ as "tool u must be in the slot sequence before tool v when the machine executes job j", for each $u, v \in C_j$. Thus, each job $j \in J$ is associated with a strict partially ordered set $(C_j, \prec^j)$ of tools. In GOF-ToSP, we are also given a setup time c_{uv} required to switch from tool u to tool v. More specifically, for all $u, v \in C$, the setup time c_{uv} is always non-negative and it is negligible if the two tools u and v coincide (i.e., $c_{uu} = 0$ for all $u \in C$). In addition, we assume that the job sequence is fixed in advance. GOF-ToSP consists of determining the assignment of the tools to the slots of the magazine during the processing of each job $j \in J$ to minimize the total setup time. The following theorem states the complexity of GOF-ToSP in the special case of binary setup times.

Theorem 4.4.1. *GOF-ToSP is $\mathcal{NP}$-hard even in the case of binary setup times (i.e., $c_{uv} \in \{0, 1\}$ for all $u, v \in C$).*

Proof. We prove the $\mathcal{NP}$-hardness through a polynomial reduction from the Disjoint Matching (DM) problem. In such a problem, given two sets P, Q with equal cardinality and given $A_1, A_2 \in P \times Q$, we would like to establish whether there exist two disjoint matchings $M_1 \subseteq A_1$ and $M_2 \subseteq A_2$ or not. The DM problem has been proven to be $\mathcal{NP}$-complete in [186] (see also [182] for a simpler proof).

We prove the $\mathcal{NP}$-hardness of GOF-ToSP by showing that there exists a polynomial reduction of DM into it. Let $m = |P| = |Q|$. We define an instance of GOF-ToSP with $n = 2m^2 + 2$ jobs and $k = m^2 + 1$. The set R_1 has cardinality k and its tools are

$$\xi_1, \ldots, \xi_m, \nu, \alpha_{11}, \ldots, \alpha_{1,m-1}, \ldots, \alpha_{k1}, \ldots, \alpha_{k,m-1}, \ldots, \alpha_{m1}, \ldots, \alpha_{m,m-1}.$$

Most of the other sets of tools are singletons. In particular, for all $h, r \in \{1, \ldots, m\}$ we have:

$$R_{1+(h-1)m+r} = \{\beta_{hr}\}, \quad R_{2+m^2+(h-1)m+r} = \{\gamma_{hr}\}.$$

The remaining set is defined as follows :

$$R_{2+m^2} = \{\nu, \mu_{11}, \ldots, \mu_{1,m-1}, \mu_{21}, \ldots, \mu_{2,m-1}, \ldots, \mu_{m1}, \ldots, \mu_{m,m-1}\}.$$

We set

$$C^1_{ij} = \begin{cases} 0 & \text{if } (i,j) \in A_1 \\ 1 & \text{otherwise} \end{cases} \qquad C^2_{ij} = \begin{cases} 0 & \text{if } (i,j) \in A_2 \\ 1 & \text{otherwise.} \end{cases}$$

Then, we set the following setup costs:

- $c(\xi_j, \beta_{ij}) = C^1_{ij}$ for all $i, j \in \{1, \ldots, m\}$;
- $c(\alpha_{ip}, \beta_{ij}) = 0$ for all $i, j \in \{1, \ldots, m\}$ and $p \in \{1, \ldots, m-1\}$;
- $c(\beta_{ij}, \mu_{ip}) = 0$ for all $i, j \in \{1, \ldots, m\}$ and $p \in \{1, \ldots, m-1\}$;
- $c(\mu_{ip}, \gamma_{ir}) = 0$ for all $i, r \in \{1, \ldots, m\}$ and $p \in \{1, \ldots, m-1\}$;
- $c(\beta_{ij}, \gamma_{rj}) = C^2_{rj}$ for all $i, j, r \in \{1, \ldots, m\}$, with $i \neq r$.

All other setup costs between distinct tools are set equal to 1, while those between identical tools are always set equal to 0. Next, for each $i \in \{1, \dots, m\}$ we define the following sets of jobs:

$$\mathcal{J}_i = \{1 + (i-1)m + 1, \dots, 1 + mi\};$$
$$\bar{\mathcal{J}} = \{m^2 + 2\};$$
$$\mathcal{J}'_i = \{2 + m^2 + (i-1)m + 1, \dots, 1 + m^2 + mi\}.$$

For $i, j \in \{1, \dots, m\}$, we refer to the j-th job in $\mathcal{J}_i$, namely job $1 + (i-1)m + j$, as job J_{ij}. Similarly, we refer to the j-th job in $\mathcal{J}'_i$, namely job $1 + m^2 + (i-1)m + j$, as job J'_{ij}. Finally, we refer to the single job in $\bar{\mathcal{J}}$ as $\bar{J}$. We denote by $\mathcal{S}_0$ the set of the first m slots, then we have the single slot $m+1$, and finally we define the following other sets of $m-1$ slots for each $i \in \{1, \dots, m\}$:

$$\mathcal{S}_i = \{m + (i-1)(m-1) + 2, \dots, m + i(m-1) + 1\}.$$

Note that, since all setup costs are nonnegative, the optimal value of the GOF-ToSP instance is nonnegative. Now, let us consider any pair (M_1, M_2) where M_1 and M_2 are *disjoint* perfect matchings between members of P and Q. Then, we can define the following solution for GOF-ToSP. For each $(i, j) \in M_1$, we leave ξ_j in the j-th slot, until we get to job J_{ij}, where we place tool β_{ij} in the slot. Then, we do not change tool in this slot for the remaining jobs in $\mathcal{J}_i$. In the slots in $\mathcal{S}_i$ we place tools β_{ih}, $h \neq j$ (always placing them in the first slot where an α tool is present), as we process the jobs in $\mathcal{J}_i \setminus \{J_{ij}\}$. Next, we process the single job $\bar{J}$ and we put tool ν in slot $m+1$ and tools μ_{ip}, $p \in \{1, \dots, m-1\}$, in all slots in $\mathcal{S}_i$, $i = 1, \dots, m$. Next we start processing the jobs in $\mathcal{J}'_i$, $i \in \{1, \dots, m\}$. For each $(h, j) \in M_2$ we do not put any tool in slot j until we reach job J'_{hj}, where we place tool γ_{hj} in the slot. Then, we do not put any further tool in this slot. In the slots in $\mathcal{S}_h$ we place tools γ_{hs}, $s \neq j$ (no matter in which order), as we process the jobs in $\mathcal{J}'_h \setminus \{J'_{hj}\}$. If we compute the overall setup cost of this solution, this is equal to

$$Setup(M_1, M_2) = \sum_{(i,j) \in M_1} C^1_{ij} + \sum_{(i,j) \in M_2} C^2_{ij}. \tag{4.18}$$

An example is displayed in Table 4.3 for a case with $m = 3$. We notice that $Setup(M_1, M_2) \geq 0$ and, in view of the definitions of C^1_{ij} and C^2_{ij}, it is equal to 0 if and only if $M_1 \subseteq A_1$, $M_2 \subseteq A_2$. In other words, if DM has a `yes` answer, then the optimal value of the GOF-ToSP instance is equal to 0.

Now we need to prove the reverse, i.e., we assume that the optimal value of the GOF-ToSP instance is equal to 0 and show that DM has a `yes` answer. We first notice that, in view of the nonnegativity of the setup costs, under the assumption that GOF-ToSP has an optimal value equal to 0, it holds that the optimal solution cannot contain any setup cost equal to 1. In particular, this implies that two β tools cannot belong to the same column, and, similarly, two γ tools cannot belong to the same column. Now, we observe that when we process the jobs in $\mathcal{J}_i$, each tool β_{ij}, $j \in \{1, \dots, m\}$, can only be placed either in one slot in $\mathcal{S}_0$, i.e., in one of the first m slots, or in one of the slots in $\mathcal{S}_i$. Since $|\mathcal{S}_i| = m - 1$, at least one of these tools has to be placed in one of the first m slots. In fact, *exactly* one of these tools has to be placed there. Indeed, if we place more than one, taking into account that among all tools β_{rh}, $r \neq i$, at least $m-1$ will be placed in the first m slots, then there would be one slot in $\mathcal{S}_0$ with two distinct β tools, which implies a non-optimal setup cost of

Table 4.3: Solution of GOF-ToSP corresponding to $M_1 = \{(1,3)(2,2)(3,1)\}$ and $M_2 = \{(1,1)(2,3)(3,2)\}$.

Set	Job	$\mathcal{S}_0$			4	$\mathcal{S}_1$		$\mathcal{S}_2$		$\mathcal{S}_3$	
	1	ξ_1	ξ_2	ξ_3	ν	α_{11}	α_{12}	α_{21}	α_{22}	α_{31}	α_{32}
$\mathcal{J}_1$	2	-	-	-	-	β_{11}	-	-	-	-	-
	3	-	—	-	-	-	β_{12}	-	-	-	-
	4	-	-	β_{13}	-	-	-	-	-	-	-
$\mathcal{J}_2$	5	-	-	-	-	-	-	β_{21}	-	-	-
	6	-	β_{22}	-	-	-	-	-	-	-	-
	7	-	-	-	-	-	-	-	β_{23}	-	-
$\mathcal{J}_3$	8	β_{31}	-	-	-	-	-	-	-	-	-
	9	-	-	-	-	-	-	-	-	β_{32}	-
	10	-	-	-	-	-	-	-	-	-	β_{33}
$\bar{\mathcal{J}}$	11	-	-	-	ν	μ_{11}	μ_{12}	μ_{21}	μ_{22}	μ_{31}	μ_{32}
$\mathcal{J}'_1$	12	γ_{11}	-	-	-	-	-	-	-	-	-
	13	-	-	-	-	γ_{12}	-	-	-	-	-
	14	-	-	-	-	-	γ_{13}	-	-	-	-
$\mathcal{J}'_2$	15	-	-	-	-	-	-	γ_{21}	-	-	-
	16	-	-	-	-	-	-	-	γ_{22}	-	-
	17	-	-	γ_{23}	-	-	-	-	-	-	-
$\mathcal{J}'_3$	18	-	-	-	-	-	-	-	-	γ_{31}	-
	19	-	γ_{32}	-	-	-	-	-	-	-	-
	20	-	-	-	-	-	-	-	-	-	γ_{33}

1. Stated in another way, the slots in $\mathcal{S}_i$ are used as a bin where all unmatched pairs involving $i \in P$ are placed, while the single member $j \in Q$ matched to i in matching M_1 will be recognized because of the tool β_{ij} inserted in slot j.
Job $\bar{J}$ plays an important role. Exploiting the peculiarity of tool ν, which, at optimal solutions, cannot be placed in a slot different from $m+1$, job $\bar{J}$ allows the replacement of the β tools in slots $\mathcal{S}_1, \ldots, \mathcal{S}_m$ with μ tools, which play the same role as the α tools. In this way, when we process the jobs $\mathcal{J}'_i$, similar to what we have already seen with the β tools, tool γ_{ij} can only be placed either in one of the first m slots, or in one of the slots in $\mathcal{S}_i$. Again, at least one of the tools γ_{ih}, $h \in \{1, \ldots, m\}$ has to be placed in one of the first m slots. For the same reason as before, *exactly* one of these tools has to be placed there, say γ_{ir}. This also means that the remaining tools γ_{is}, $s \neq r$, are placed in the slots in $\mathcal{S}_i$, again used as a bin for unmatched pairs, with an overall setup cost from such slots equal to 0. Taking into account that such setup cost has to be paid for all $i \in \{1, \ldots, m\}$, we have an overall contribution to the overall setup cost from the slots $m+1$ up to m^2, i.e., the slots in $\cup_{i=1}^{m} \mathcal{S}_i$, equal to 0. The tool γ_{ir}, i.e., the tool placed in one of the first m slots, could only be placed in slot $r \in \mathcal{S}_0$ and only if slot r contains tool β_{hr} with $h \neq i$. Its presence there means that in the second matching we match member $i \in P$ with member $r \in Q$. In this case we pay C^2_{ir}, which must be equal to 0, or, stated in another way, $(i, r) \in A_2$.
Note that the non-optimal setup cost $c(\gamma_{ir}, \beta_{ir}) = 1$ guarantees that the two matchings defined by the β and γ tools in the first m slots are disjoint, as required in DM. Therefore, optimal solutions of GOF-ToSP with an objective function value equal to 0 can only be those corresponding to disjoint matchings M_1 and M_2 such that $M_1 \subseteq A_1$, $M_2 \subseteq A_2$, which implies that the DM problem has a `yes` answer. □

The instance of GOF-ToSP to which the DM instance was reduced does not fulfill

the triangular inequality. Indeed, we have, e.g., that $c(\beta_{ij}, \gamma_{ij}) = 1$, while $c(\beta_{ij}, \mu_{ip}) = c(\mu_{ip}, \gamma_{ij}) = 0$ for all $i, j \in \{1, \ldots, m\}$ and $p \in \{1, \ldots, m-1\}$. However, a similar reduction into an instance fulfilling the triangular inequality can be defined if we allow for general, rather than binary, setup costs. We briefly sketch the proof. It is enough to change the following setup costs with respect to the previous reduction:

- $c(\beta_{ij}, \mu_{ip}) = 1$ for all $i, j \in \{1, \ldots, m\}$ and $p \in \{1, \ldots, m-1\}$;
- $c(\nu, \zeta) = c(\zeta, \nu) = m^2$ for each tool ζ different from ν.

Then, we consider the complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ is the set of all tools, and we associate a distance equal to the corresponding setup cost with each arc. Finally, we define new setup costs $c'(i, j)$ for each pair $i, j \in \mathcal{V}$, $i \neq j$, by setting $c'(i, j)$ equal to the shortest path between i and j. Thus, we are sure that the new setup costs fulfill the triangular inequality. In this case it can be proven that the DM instance has a `yes` answer if and only if the corresponding GOF-ToSP instance has optimal value $m(m-1)$, equivalent to the minimum setup costs to be paid for the single job in $\bar{\mathcal{J}}$. The proof is completely analogous to the proof given above.

It is worthwhile to observe that, while in the case of fixed setup times and unordered C_j a fixed sequence of jobs simplifies the problem (CUF-ToSP is solvable in polynomial-time, while CUV-ToSP is $\mathcal{NP}$-hard), a fixed sequence of tools in the slots (as in GOF-ToSP with binary setup times) makes the problem hard even when the sequence of jobs is fixed.

4.4.1 Five-index Arc Flow Formulation for GOF-ToSP

The mathematical model proposed for GOF-ToSP recalls the one introduced in Section 4.3.1 for GUF-ToSP, but with an additional index, referring to the slots, for the variables, which is needed to preserve the order of the tools within the slots. Given a job sequence J, a solution of GOF-ToSP can be represented as $|K|$ paths from a source to a destination visiting disjoint nodes of a specific network $\mathcal{N}$. Let $J^* = \{0, 1, \ldots, n+1\}$, where the two additional jobs 0 and $n+1$ are dummy jobs, each with a set of tools made of the single dummy tool 0, i.e., $C_0 = C_{n+1} = \{0\}$.
We consider the network $\mathcal{N}$ to be composed of:

- a directed graph $G = (\mathcal{V}; \mathcal{A})$, where $\mathcal{V}$ is composed of all couples (u, j) with $j \in J^*$ and $u \in C_j$, and $\mathcal{A}$ includes all arcs from node $(u, i) \in \mathcal{V}$ to node $(v, j) \in \mathcal{V}$ such that $i \in J^* \setminus \{n+1\}$, $j \in J^* \setminus \{0\}$, and $j > i$;
- a cost function $w : \mathcal{A} \to \mathbb{R}^+$ that maps each arc $((u, i), (v, j)) \in \mathcal{A}$ to a cost c_{uv} (i.e., the setup time required for switching tool u of job i with tool v of job j).

The formulation considers a copy of graph G for each slot $k \in K$ and requires a binary variable $\phi^k_{((u,i),(v,j))}$ taking the value 1 if tool $u \in C_i$ is loaded in the k-th slot and is switched with tool $v \in C_j$, 0 otherwise. We obtain:

$$\min \sum_{k \in K} \sum_{((u,i),(v,j)) \in \mathcal{A}} c_{uv} \phi^k_{((u,i),(v,j))} \tag{4.19}$$

$$\sum_{\substack{(v,j):\\ ((v,j),(u,i)) \in \mathcal{A}}} \phi^k_{((v,j),(u,i))} - \sum_{\substack{(v,j):\\ ((u,i),(v,j)) \in \mathcal{A}}} \phi^k_{((u,i),(v,j))} = \begin{cases} -1, & \text{if } i = 0, \\ 1, & \text{if } i = n+1, \\ 0, & \text{otherwise,} \end{cases} \qquad k \in K, (u, i) \in \mathcal{V} \tag{4.20}$$

$$\sum_{k\in K}\sum_{\substack{(v,j):\\((u,i),(v,j))\in\mathcal{A}}}\phi^k_{((u,i),(v,j))}=1 \qquad (u,i)\in\mathcal{V}: i\notin\{0,n+1\} \tag{4.21}$$

$$\sum_{h\geq k}\sum_{\substack{(u,i):\\((u,i),(v,j))\in\mathcal{A}}}\phi^h_{((u,i),(v,j))}+ \\ +\sum_{h\leq k}\sum_{\substack{(u,i):\\((u,i),(w,j))\in\mathcal{A}}}\phi^h_{((u,i),(w,j))}\leq 1 \qquad \begin{array}{l}k\in K,\ (v,j),(w,j)\in\mathcal{V}:\\ v\prec^j w\end{array} \tag{4.22}$$

$$\phi^k_{((u,i),(v,j))}\in\{0,1\} \qquad ((u,i),(v,j))\in\mathcal{A}, k\in K. \tag{4.23}$$

The objective function (4.19) minimizes the total setup times. Flow conservation is imposed by constraints (4.20). Constraints (4.21) guarantee that, during the processing of job $j\in J$, each tool required by j takes exactly a slot. Constraints (4.22) guarantee that the precedence relation $\prec^j$ at each job j is fulfilled.

4.4.2 Preprocessing

We introduce a method that relies on the triangular inequality assumption for the setup times and allows us to potentially reduce the number of jobs in an instance. Let us introduce the concept of a mergeable job. A job i is said to be *mergeable* in job j if $C_i\subseteq C_j$ and, for each $u,v\in C_i$ such that $u\prec^i v$, it also holds that $u\prec^j v$. In such a case, if job i is next to job j, it is possible to merge job i into job j, i.e., consider them as a unique job, having the characteristics of job j.

Proposition 1. Let us consider an instance of GOF-ToSP for which the triangular inequality holds. Assume job $j+1$ is mergeable in job j. Then, there exists an optimal solution without any tool switching operation between jobs j and $j+1$, i.e., we can merge job $j+1$ into j. Similarly, if job j is mergeable in job $j+1$, then, there exists an optimal solution without any tool switching operation between jobs j and $j+1$, i.e., we can merge job j into $j+1$.

Proof. We prove the result only for the case where $j+1$ is mergeable in job j. The proof for the other case is analogous. An optimal solution S defines, for each job $i\in J$ and each slot $k\in K$, a tool u^k_i that lies in the k-th slot just before processing job i. Now, we consider the solution $\overline{S}$ obtained from S imposing that $u^k_j=u^k_{j+1}$ for each $k\in K$. Since job $j+1$ is mergeable in job j, we have, in view of the triangular inequality, that:

$$\sum_{k\in K}\sum_{i=0}^{n}c_{u^k_iu^k_{i+1}}=\sum_{k\in K}\sum_{i=0,\ i\neq j,j+1}^{n}c_{u^k_iu^k_{i+1}}+\sum_{k\in K}(c_{u^k_ju^k_{j+1}}+c_{u^k_{j+1}u^k_{j+2}})\geq \\ \geq\sum_{k\in K}\sum_{i=0,\ i\neq j,j+1}^{n}c_{u^k_iu^k_{i+1}}+\sum_{k\in K}c_{u^k_ju^k_{j+2}}$$

i.e., the objective function value of the feasible solution $\overline{S}$ is at least as good as the objective function value of the optimal solution S. Thus, $\overline{S}$ is also an optimal solution, for which no tool switching operation occurs between jobs j and $j+1$. □

The operation of merging jobs can be iteratively repeated. Thus, given a job j, we merge jobs $j+1,\ldots,j+h$ if they are all mergeable into j. Similarly, we merge jobs $j-1,\ldots,j-r$ if they are all mergeable into j.

4.5 GOV-ToSP

GOV-ToSP is the generalization of GOF-ToSP in which the job sequence is not fixed. Concerning its complexity we can prove the following result.

Theorem 4.5.1. *GOV-ToSP is strongly $\mathcal{NP}$-hard even when $|K| = 1$.*

Proof. This result easily follows from the observation that there exists a polynomial-time reduction of the Traveling Salesman Problem (see, e.g., Applegate et al. [31]) into GOV-ToSP with a single slot. □

Note that the above result also states the $\mathcal{NP}$-hardness of GUV-ToSP, since for $|K| = 1$ the order of the tools is irrelevant.

4.5.1 Six-index Arc Flow Formulation for GOV-ToSP

A solution of GOV-ToSP can be represented as $|K|$ paths from a source to a destination visiting disjoint nodes of a specific network $\mathcal{N}$. In this section, we present an arc flow formulation, similar to the one described in Section 4.4.1 for GOF-ToSP, but containing an additional time-index with respect to that formulation. In doing so, the planning horizon is discretized into $|T|$ time periods (where $T = \{1, \dots, n\}$), each corresponding to a time period in which a job can be processed. Let T^* be the set of time periods in T with two additional times 0 and $n+1$, i.e., the processing times of dummy jobs 0 and $n+1$, respectively.
Since in GOV-ToSP the job sequence is not fixed, the directed graph $G = (\mathcal{V}; \mathcal{A})$ of $\mathcal{N}$ is different from the graph defined for the GOF-ToSP model. In this case, $\mathcal{V} = \{(u,i,t) : u \in C, i \in J, t \in T\} \cup \{(0,0,0), (0,n+1,n+1)\}$, where additional nodes $(0,0,0)$ and $(0,n+1,n+1)$ correspond to the source and the sink of all paths, respectively. Set $\mathcal{A}$ includes all arcs from nodes $(u,i,t) \in \mathcal{V}$ to nodes $(v,j,t+1) \in \mathcal{V}$ such that $i \in J^* \setminus \{n+1\}$, $j \in J^* \setminus \{0,i\}$, and $t \in T^* \setminus \{n+1\}$. Similar to the GOF-ToSP model, a cost function $w : \mathcal{A} \to \mathbb{R}^+$ maps each arc $((u,i,t),(v,j,t+1)) \in \mathcal{A}$ to a cost c_{uv}.
The following formulation considers a copy of graph G for each slot $k \in K$ and requires a binary variable $\phi^k_{((u,i,t),(v,j,t+1))}$ taking the value 1 if between time t and $t+1$ we switch from job i to job j and from tool u to tool v in the k-th tool group.

$$\min \sum_{((u,i,t),(v,j,t+1)) \in \mathcal{A}} c_{uv} \sum_{k \in K} \phi^k_{((u,i,t),(v,j,t+1))} \tag{4.24}$$

$$\sum_{\substack{(v,j,t-1):\\ ((v,j,t-1),(u,i,t)) \in \mathcal{A}}} \phi^k_{((v,j,t-1),(u,i,t))} - \sum_{\substack{(v,j,t+1):\\ ((u,i,t),(v,j,t+1)) \in \mathcal{A}}} \phi^k_{((u,i,t),(v,j,t+1))} =$$

$$= \begin{cases} -1, & \text{if } i = 0, \\ 1, & \text{if } i = n+1, \\ 0, & \text{otherwise,} \end{cases} \qquad \begin{array}{l} k \in K, \\ (u,i,t) \in \mathcal{V} \end{array} \tag{4.25}$$

$$\sum_{k \in K} \sum_{(u,i,t) \in \mathcal{V}} \sum_{\substack{(v,j,t+1):\\ ((u,i,t),(v,j,t+1)) \in \mathcal{A}}} \phi^k_{((u,i,t),(v,j,t+1))} = 1 \qquad i \in J, u \in C_i \tag{4.26}$$

$$\sum_{h \geq k} \sum_{\substack{(u,i,t-1):\\ ((u,i,t-1),(v,j,t)) \in \mathcal{A}}} \phi^h_{((u,i,t-1),(v,j,t))} +$$

$$+ \sum_{h \leq k} \sum_{\substack{(u,i,t-1):\\ ((u,i,t-1),(w,j,t)) \in \mathcal{A}}} \phi^h_{((u,i,t-1),(w,j,t))} \leq 1 \qquad \begin{array}{l} k \in K, \\ (v,j,t),(w,j,t) \in \mathcal{V} : \\ v \prec^j w \end{array} \tag{4.27}$$

$$\sum_{k \in K} \sum_{\substack{(v,j,t+1):\\ ((u,i,t),(v,j,t+1)) \in \mathcal{A}}} \phi^k_{((u,i,t),(v,j,t+1))} +$$

$$+\sum_{k\in K}\sum_{\substack{(v,j,t'+1):\\((u',i,t'),(v,j,t'+1))\in\mathcal{A}}}\phi^k_{((u',i,t'),(v,j,t'+1))}\leq 1 \qquad \begin{array}{l}(u,i,t)\in\mathcal{V},\\(u',i,t')\in\mathcal{V}:t\neq t'\end{array} \tag{4.28}$$

$$\phi^k_{((u,i,t),(v,j,t+1))}\in\{0,1\} \qquad \begin{array}{l}k\in K,\\((u,i,t),(v,j,t+1))\in\mathcal{A}.\end{array} \tag{4.29}$$

The objective function (4.24) minimizes the total setup times. Constraints (4.25) impose flow conservation. Constraints (4.26) guarantee that, during the processing of job $j \in J$, each tool required by the job takes exactly a slot. Constraints (4.27) guarantee that the precedence relation between tools in C_j is respected for all jobs $j \in J$. Constraints (4.28) guarantee a consistent sequencing of jobs, i.e., among all nodes associated with a given job i, only those corresponding to a single time period can have a positive out-flow.

4.6 Computational Results

This section illustrates computational experiments designed and conducted to assess the effectiveness of the proposed models. The formulations of Sections 4.2.1, 4.4.1, and 4.5.1 have been implemented in Phyton by using Gurobi 9.1.2 as mixed integer linear programming solver. The resulting algorithms are called CUF-Alg, GOF-Alg, and GOV-Alg. The tests have been executed on a 2.3GHz Intel Xeon Gold 6252N with 16 GB of memory. We let Gurobi use all four available threads. The CPU time limit was set to 600 seconds per instance.

4.6.1 Test Instances

To evaluate the effectiveness of the models, a number of instances have been generated with the aim of covering different scenarios of interest. It has been necessary to do this as the instances for CUV-ToSP available in the literature (see Calmels [73]) are difficult to adapt and extend to our problems.
The generation of a set of instances has the following input parameters: the number $|J|$ of jobs, the number $|C|$ of available tools, and the number $|K|$ of available slots. For all tools $u, v \in C$, $u \neq v$, the setup time c_{uv} is set to an integer value uniformly distributed in $[1; |C|]$ (and it is set to zero if the two tools u and v coincide). Then, since the triangular inequality holds for the setup times, if $c_{uw} > c_{uv} + c_{vw}$ for some $u, v, w \in C$, we set $c_{uw} = c_{uv} + c_{vw}$ and we repeat this operation until $c_{uw} \leq c_{uv} + c_{vw}$ holds for all triplets. For each instance, $|J|$ jobs are created as follows: the number l of tools required by a job j is an integer value uniformly distributed in $[1; |K|]$; and the set C_j is generated by randomly selecting l different tools from C and randomly ordering them.

A job is added to J if it is not mergeable in any job previously added to J. Thus, the preprocessing method described in Section 4.4.2 is used to generate instances that cannot be reduced in terms of number of jobs. Computational tests were performed on a set of 90 instances generated with various combinations of $|J| \in \{5, 7, 10, 20, 30\}$, $|C| \in \{5, 10, 30, 40, 50, 60\}$, and $|K| \in \{2, 3, 10, 15\}$. The generated instances are divided into 5 groups of 10 instances each having the same number of jobs, tools, and slots. All the generated instances are publicly available at https://github.com/regor-unimore/Tool-switching-problems.

4.6.2 Computational Evaluations

Theorem 4.2.1 implies that the CUF-ToSP can be solved as a linear program. Therefore, large instances can be solved in very short run-time. For the sake of completeness, we report that all the largest instances that we proposed (i.e., with $|J| = 30$, $|C| = 60$, and $|K| = 15$) are solved in less than one tenth of a second by means of CUF-Alg.

Tables 4.4 and 4.5 highlight the results obtained by GOF-Alg and GOV-Alg, respectively. Entries in columns z^* and LB exhibit the integer optimal solution value (or the value of the best solution found when optimality is not proven) and the final lower bound obtained by the search tree, respectively. The percentage gap, shown in column GAP, is computed as $100(z^* - LB)/z^*$, while column *sec* exhibits the run-time required to solve the instances (expressed in seconds). We also provide some information about the continuous relaxations of the models: z_{LP} is the value of the optimal solution at the root node of the branch-decision tree, i.e., the linear-programming relaxation value (which we obtained by turning off the Gurobi presolver functionality); GAP_{LP} is computed as $100(z^* - z_{LP})/z^*$; and sec_{LP} exhibits the run-time required to solve the linear-programming relaxation (expressed in seconds). Each row in the table gives the results obtained on an instance. In addition, for each group, entries in row AVG exhibit the average values of each column. Symbol "–" indicates that no feasible solution was produced by the algorithm.
Table 4.4 shows that GOF-Alg is capable to find a solution within the time limit for almost all instances. It is worth observing that the size of the proposed instances is compatible with real data (see Iori, Locatelli, and Locatelli [275]). Indeed, a flexographic printer is usually capable of holding at most 9 colours at a time, 10 jobs correspond approximately to a week of production, and no more than 30 colours are usually required for printing 10 different jobs. As expected, the run-time increases more with the increase of the parameter $|K|$, on which the number of variables of the formulation in Section 4.4.1 mostly depends. Indeed, in the worst case, the number of variables is equal to $N = |K|(|K|^2(|J|^2 - |J|)/2 + 2|K||J| + 1)$, which is $|K|$ times the number of arcs of the directed graph $G = (\mathcal{V}; \mathcal{A})$ described in Section 4.4.1 for the case $|C_i| = |K|$ for $i = 1, \ldots, n$. We can also observe that N does not depend on the size of set C (i.e., on the number of total tools).
It follows from Theorem 4.5.1 that GOV-ToSP is the most difficult problem in theory. This result is also confirmed in practice by computational experiments. GOV-Alg is able to find the minimum solution within the time limit just for small instances (i.e., the instances reported in Table 4.5) and in many cases the initial z_{LP} bound is trivially equal to 0. For larger instances, GOV-Alg failed to find even a feasible solution within the time limit, mainly because of the large number of variables and constraints of the model.

4.7 Conclusions and Future Research

In this chapter, we addressed four different combinatorial optimization problems of increasing difficulty: CUF-ToSP, GUF-ToSP, GOF-ToSP, and GOV-ToSP. In particular, GOF-ToSP is motivated by a real-world application in a company that operates in the packaging industry by producing and printing packaging materials for food products. For each problem we discussed its complexity and proposed a mathematical programming model. Experiments over realistic instances confirmed the theoretical findings: GOV-ToSP is the most difficult problem and the proposed method is able to find the minimum solution within the time limit just for the first and smaller

Table 4.4: Computational results for GOF-ToSP.

Inst.	$\|J\|$	$\|C\|$	$\|K\|$	LP			ILP			
				z_{LP}	GAP_{LP}	sec_{LP}	z^*	LB	GAP	sec
TI_Ins01_10_30_10	10	30	10	163.9	7.9%	0.9	178	178	0.0%	26.9
TI_Ins02_10_30_10	10	30	10	131.4	1.9%	0.6	134	134	0.0%	5.7
TI_Ins03_10_30_10	10	30	10	155.5	2.2%	0.5	159	159	0.0%	5.8
TI_Ins04_10_30_10	10	30	10	132.2	6.9%	0.9	142	142	0.0%	21.0
TI_Ins05_10_30_10	10	30	10	243.0	0.8%	0.9	245	245	0.0%	5.1
TI_Ins06_10_30_10	10	30	10	156.9	1.3%	0.8	159	159	0.0%	4.3
TI_Ins07_10_30_10	10	30	10	180.0	0.0%	0.8	180	180	0.0%	2.7
TI_Ins08_10_30_10	10	30	10	187.6	3.3%	1.0	194	194	0.0%	7.4
TI_Ins09_10_30_10	10	30	10	230.7	7.0%	1.5	248	248	0.0%	33.0
TI_Ins10_10_30_10	10	30	10	158.8	7.7%	1.0	172	172	0.0%	14.0
AVG				174.0	3.9%	0.9	181.1	180.9	0.0%	12.6
TI_Ins01_20_40_10	20	40	10	251.3	4.4%	4.3	263	263	0.0%	100.2
TI_Ins02_20_40_10	20	40	10	433.2	0.2%	14.3	434	434	0.0%	42.0
TI_Ins03_20_40_10	20	40	10	279.6	3.6%	9.8	290	290	0.0%	69.8
TI_Ins04_20_40_10	20	40	10	380.8	0.6%	11.9	383	383	0.0%	33.7
TI_Ins05_20_40_10	20	40	10	347.2	1.9%	10.8	354	354	0.0%	50.2
TI_Ins06_20_40_10	20	40	10	370.8	0.3%	11.0	372	372	0.0%	35.7
TI_Ins07_20_40_10	20	40	10	265.8	5.1%	4.6	280	280	0.0%	96.8
TI_Ins08_20_40_10	20	40	10	285.0	3.1%	6.7	294	294	0.0%	78.0
TI_Ins09_20_40_10	20	40	10	309.1	2.8%	9.6	318	318	0.0%	80.8
TI_Ins10_20_40_10	20	40	10	300.3	5.0%	8.2	316	316	0.0%	106.0
AVG				322.3	2.7%	9.1	330.4	330.4	0.0%	69.3
TI_Ins01_20_50_10	20	50	10	247.3	2.6%	5.3	254	254	0.0%	47.7
TI_Ins02_20_50_10	20	50	10	235.7	2.2%	6.0	241	241	0.0%	28.8
TI_Ins03_20_50_10	20	50	10	214.1	11.5%	6.4	242	222	8.9%	600.8
TI_Ins04_20_50_10	20	50	10	283.5	0.5%	5.4	285	285	0.0%	21.2
TI_Ins05_20_50_10	20	50	10	250.3	3.3%	5.7	259	259	0.0%	46.6
TI_Ins06_20_50_10	20	50	10	314.7	1.0%	11.0	318	318	0.0%	34.7
TI_Ins07_20_50_10	20	50	10	337.8	2.9%	12.3	348	348	0.0%	72.8
TI_Ins08_20_50_10	20	50	10	267.9	7.6%	8.1	290	290	0.0%	483.2
TI_Ins09_20_50_10	20	50	10	204.3	5.0%	5.3	215	215	0.0%	117.4
TI_Ins10_20_50_10	20	50	10	361.9	3.0%	11.7	373	373	0.0%	86.8
AVG				271.8	4.0%	7.7	282.5	280.4	0.9%	154.0
TI_Ins01_10_50_15	10	50	15	401.9	1.5%	72.5	408	408	0.0%	129.0
TI_Ins02_10_50_15	10	50	15	82.6	14.0%	4.3	96	96	0.0%	294.9
TI_Ins03_10_50_15	10	50	15	77.4	14.0%	3.0	90	90	0.0%	172.8
TI_Ins04_10_50_15	10	50	15	156.8	5.6%	9.3	166	166	0.0%	141.9
TI_Ins05_10_50_15	10	50	15	142.2	11.1%	11.2	160	160	0.0%	378.7
TI_Ins06_10_50_15	10	50	15	173.3	3.7%	10.6	180	180	0.0%	119.6
TI_Ins07_10_50_15	10	50	15	72.2	22.4%	2.5	93	82	13.4%	600.1
TI_Ins08_10_50_15	10	50	15	70.8	11.6%	1.7	80	80	0.0%	131.5
TI_Ins09_10_50_15	10	50	15	243.4	4.6%	19.8	255	255	0.0%	280.2
TI_Ins10_10_50_15	10	50	15	252.4	3.3%	27.8	261	261	0.0%	232.6
AVG				167.3	9.2%	16.3	178.9	177.7	1.4%	248.1
TI_Ins01_30_60_10	30	60	10	344.2	5.4%	13.7	364	356	2.1%	600.3
TI_Ins02_30_60_10	30	60	10	242.1	6.2%	19.5	258	248	4.0%	600.2
TI_Ins03_30_60_10	30	60	10	394.8	3.2%	28.0	408	408	0.0%	189.0
TI_Ins04_30_60_10	30	60	10	481.1	3.8%	51.7	500	496	0.9%	600.4
TI_Ins05_30_60_10	30	60	10	420.6	2.6%	29.9	432	432	0.0%	257.7
TI_Ins06_30_60_10	30	60	10	487.1	2.4%	46.7	499	499	0.0%	409.9
TI_Ins07_30_60_10	30	60	10	418.8	3.5%	45.4	434	434	0.0%	480.6
TI_Ins08_30_60_10	30	60	10	347.5	4.8%	17.8	365	360	1.4%	600.4
TI_Ins09_30_60_10	30	60	10	397.5	8.0%	36.2	432	403	7.1%	601.0
TI_Ins10_30_60_10	30	60	10	376.9	3.8%	23.1	392	392	0.0%	357.6
AVG				391.1	4.4%	31.2	408.4	402.9	1.5%	469.7

Table 4.5: Computational results for GOV-ToSP.

Inst.	$\|J\|$	$\|C\|$	$\|K\|$	LP			ILP			
				z_{LP}	GAP_{LP}	sec_{LP}	z^*	LB	GAP	sec
TI_Ins01_5_5_2	5	5	2	1.5	96.9%	0.2	48	48	0.0%	5.1
TI_Ins02_5_5_2	5	5	2	0.0	100.0%	0.1	25	25	0.0%	5.0
TI_Ins03_5_5_2	5	5	2	0.0	100.0%	0.1	23	23	0.0%	5.3
TI_Ins04_5_5_2	5	5	2	6.0	83.3%	0.1	36	36	0.0%	6.8
TI_Ins05_5_5_2	5	5	2	8.0	63.6%	0.2	22	22	0.0%	4.8
TI_Ins06_5_5_2	5	5	2	0.0	100.0%	0.1	47	47	0.0%	5.6
TI_Ins07_5_5_2	5	5	2	7.5	79.7%	0.2	37	37	0.0%	6.0
TI_Ins08_5_5_2	5	5	2	0.0	100.0%	0.1	26	26	0.0%	5.3
TI_Ins09_5_5_2	5	5	2	0.0	100.0%	0.1	46	46	0.0%	3.8
TI_Ins10_5_5_2	5	5	2	4.1	92.8%	0.2	57	57	0.0%	7.2
AVG				2.7	83.3%	0.2	36.7	36.7	0.0%	5.5
TI_Ins01_5_10_2	5	10	2	0.7	95.8%	0.4	16	16	0.0%	28.9
TI_Ins02_5_10_2	5	10	2	12.5	67.9%	0.5	39	39	0.0%	29.5
TI_Ins03_5_10_2	5	10	2	6.0	59.7%	0.5	15	15	0.0%	9.4
TI_Ins04_5_10_2	5	10	2	11.2	76.7%	0.5	48	48	0.0%	28.2
TI_Ins05_5_10_2	5	10	2	1.0	97.1%	0.4	35	35	0.0%	45.3
TI_Ins06_5_10_2	5	10	2	6.1	69.7%	0.4	20	20	0.0%	28.5
TI_Ins07_5_10_2	5	10	2	0.0	100.0%	0.3	27	27	0.0%	37.4
TI_Ins08_5_10_2	5	10	2	0.0	100.0%	0.4	14	14	0.0%	90.3
TI_Ins09_5_10_2	5	10	2	0.8	97.6%	0.4	33	33	0.0%	37.8
TI_Ins10_5_10_2	5	10	2	1.7	86.7%	0.4	13	13	0.0%	13.2
AVG				4.0	85.1%	0.4	26.0	26.0	0.0%	34.9
TI_Ins01_5_10_3	5	10	3	0.0	100.0%	0.6	27	27	0.0%	136.3
TI_Ins02_5_10_3	5	10	3	0.0	100.0%	0.6	37	37	0.0%	127.0
TI_Ins03_5_10_3	5	10	3	0.0	100.0%	0.5	9	9	0.0%	171.5
TI_Ins04_5_10_3	5	10	3	0.0	100.0%	0.4	17	17	0.0%	143.3
TI_Ins05_5_10_3	5	10	3	0.0	100.0%	0.5	28	28	0.0%	119.6
TI_Ins06_5_10_3	5	10	3	0.0	100.0%	0.6	34	34	0.0%	132.4
TI_Ins07_5_10_3	5	10	3	0.0	100.0%	0.5	5	5	0.0%	145.6
TI_Ins08_5_10_3	5	10	3	9.3	90.1%	1.0	93	93	0.0%	89.4
TI_Ins09_5_10_3	5	10	3	0.0	100.0%	0.6	13	13	0.0%	88.9
TI_Ins10_5_10_3	5	10	3	0.0	100.0%	0.6	30	30	0.0%	88.6
AVG				0.9	99.1%	0.6	29.3	29.2	0.0%	124.3
TI_Ins01_7_10_2	7	10	2	5.9	89.8%	2.3	58	58	0.0%	468.4
TI_Ins02_7_10_2	7	10	2	4.1	93.4%	2.1	62	62	0.0%	319.6
TI_Ins03_7_10_2	7	10	2	2.0	94.3%	1.9	35	35	0.0%	388.3
TI_Ins04_7_10_2	7	10	2	0.6	98.9%	1.9	51	51	0.0%	468.0
TI_Ins05_7_10_2	7	10	2	2.5	92.4%	2.0	33	33	0.0%	398.0
TI_Ins06_7_10_2	7	10	2	7.7	83.6%	2.3	47	47	0.0%	354.9
TI_Ins07_7_10_2	7	10	2	0.0	100.0%	1.4	32	32	0.0%	355.2
TI_Ins08_7_10_2	7	10	2	1.0	98.5%	1.9	65	65	0.0%	534.0
TI_Ins09_7_10_2	7	10	2	1.0	97.4%	1.8	38	16	57.9%	600.2
TI_Ins10_7_10_2	7	10	2	2.1	91.4%	1.9	24	24	0.0%	251.4
AVG				2.7	93.9%	1.9	44.5	42.3	5.8%	413.8
TI_Ins01_7_10_3	7	10	3	0.0	-	2.7	-	0	-	600.4
TI_Ins02_7_10_3	7	10	3	0.0	100.0%	2.7	58	0	100.0%	600.2
TI_Ins03_7_10_3	7	10	3	0.0	-	2.5	-	0	-	600.4
TI_Ins04_7_10_3	7	10	3	0.0	-	2.8	-	0	-	600.6
TI_Ins05_7_10_3	7	10	3	0.0	-	2.8	-	0	-	600.6
TI_Ins06_7_10_3	7	10	3	0.0	-	2.6	-	0	-	600.5
TI_Ins07_7_10_3	7	10	3	0.0	-	3.7	-	20	-	600.8
TI_Ins08_7_10_3	7	10	3	0.0	-	2.8	-	4	-	600.5
TI_Ins09_7_10_3	7	10	3	0.0	-	2.5	-	0	-	600.5
TI_Ins10_7_10_3	7	10	3	0.0	-	3.2	-	0	-	600.3
AVG				0.0	100.0%	2.8	58.0	2.4	100.0%	600.5

group of instances.
On the other hand, the proposed approach to solve GOF-ToSP is able to find optimal solutions for realistic instances in a short time and thus can provide quick support to the company on its weekly decisions. It is worthwhile to remark that the computing times are currently not a major issue for the company. Indeed, taking into account that the planned activities of the proposed instances cover approximately one week, larger computing times could still be feasible in practice. We also tried to implement alternative formulations for GOV-ToSP, having fewer variables and constraints: a four-index formulation and a five-index arc flow formulation. The behavior of these two models was worse than that of GOV-Alg, mainly because the relaxed linear programming bound was zero for most of the instances.
To the best of our knowledge, the complexity of COF-ToSP and COV-ToSP is still unknown. A thorough study of such complexity results represents an interesting possible future research direction. As a further possible topic for future research, finding new valid inequalities to strengthen the proposed formulations for GOV-ToSP and GOF-ToSP is also of high interest.

Another interesting future research direction is the development of algorithms for GOV-ToSP. These could be obtained by improving the formulation we proposed, devising new formulations, or developing heuristic strategies, especially to solve large instances.

Data Availability Statement

The data used in this research are available at https://github.com/regor-unimore/Tool-switching-problems.

Acknowledgment

This research was partially supported by the Spanish research project PID2019-104928RB-I00. We also acknowledge financial support from Istituto Stampa s.r.l. (Italy).

Chapter 5

A GRASP for a real-world scheduling problem with unrelated parallel print machines and sequence-dependent setup times*

In this chapter we consider a real-world scheduling problem arising in the color printing industry. The problem consists in assigning print jobs to a heterogeneous set of flexographic printer machines and finding a processing sequence for the jobs assigned to each machine. The machines are characterized by a limited sequence of color groups and can equip additional components (e.g., embossing rollers and perforating rolls) to process jobs that require specific treatments. The process to equip a machine with an additional component or to clean a color group takes a long time, with the effect of significantly raising the setup times. The aim is to minimize a weighted sum of total weighted tardiness and total setup time. The problem derives from the activities of an Italian food packaging company. To solve it, we developed a greedy randomized adaptive search procedure equipped with several local search procedures. The excellent performance of the algorithm is proved by extensive computational experiments on real-world instances, for which it produced good-quality solutions within a limited computing time. The algorithm is currently in use at the company to support their weekly scheduling decisions.

5.1 Introduction

The food packaging industry has undergone a remarkable evolution over the last years. For capturing and establishing a lasting and strong bond with customers, the graphic design and choice of materials for food packages has become more and more important (see, e.g., Paine and Paine [390]). As a consequence, the color printing industry has been forced to satisfy increasingly demanding market requirements dictated by the food industry and its customers. For printing flexible food packages, rotogravure and flexography printing are the most used technologies. More specifically, flexography is faster, less expensive, and allows for printing on almost any type of material such as plastic, paper, cellophane, and aluminum foil (see, e.g., Kipphan [305]). In addition, even if rotogravure printing was frequently used to guarantee a better quality, nowadays flexography has reached an equivalent quality.

*The results of this chapter appears in: M. Iori, A. Locatelli, and M. Locatelli. A GRASP for a real-world scheduling problem with unrelated parallel print machines and sequence-dependent setup times". In: *International Journal of Production Research* (2022), (in press).

A flexographic printer machine is a flexible manufacturing machine (see, e.g., Crama [124]) able to handle different types of operations (i.e., coloring, embossing, and perforating) which are performed by the available tools installed in its magazine. The number of tools that a machine can equip is limited by its magazine capacity. For instance, the number of different ink cartridges that a machine can simultaneously load is bounded by its number of color groups.

A customer order, simply denoted as a *job* in the following, is characterized by a number of characteristics, namely: order quantity, due date, release date, job-machine incompatibilities, different processing time on the (compatible) machines, and a set of required tools (e.g., ink cartridges, embossing rollers, and perforating rolls).

The printing process of a job consists of two phases: the *machine setup phase* and the *printing phase* (see, e.g., Schuurman and Van Vuuren [442]).

The machine setup phase consists in preparing the machine for printing a new job and may require to perform some tool switching operations. Indeed, the limited magazine capacity of the machines and the demand to process various types of jobs often induce to replace the currently installed tools with other tools required to process the next jobs. Since the variety of colors required by the jobs is enormous, changing the ink cartridges in the color groups of a machine is definitely the most frequent switching operation performed between two jobs.

This operation requires a washing process to remove the previous color residue from the inner surface of the color group, because such residual could affect the quality of the next job. The process to wash a color group takes a long time and depends directly on the specific colors involved. Moreover, if a printer machine equips an automatic washing system, the color groups can be washed in parallel, otherwise they have to be washed one by one. Hence, the time required for the machine setup phase is machine dependent, job dependent, and job sequence dependent. Indeed, it depends on the washing system of the machine, on the tools (i.e., ink cartridges, embossing rollers, and perforating rolls) available in the magazine, and on the tools required by the next job.

On the other side, the printing phase is just machine dependent and job dependent. In particular, it only depends on the printing speed of the machine, on the type of printed material (e.g., plastic, paper, cellophane, and aluminum foil), and on print length (i.e., the length in meters of the job). This can be summarized by simply computing a processing time of the given job on each machine that can process it.

Since tool switching operations are strongly time consuming, the machine setup phase takes a considerable part of the entire production time. In addition, a large part of the tool switching operations can be performed manually, thus requiring more human labor and forcing the machines to remain in a non-printing mode to ensure workers' safety. Thanks to the introduction of automatic washing systems, setup times have been considerably reduced in modern flexographic presses. Despite this, the growing demand for customized packaging has increased the demand of small-length jobs as well as the complexity of the jobs, with a consequent increase of the impact of the setup times on the total production time. Indeed, in the real-world applications that we face, setup time requires around 65% of the total production time. This makes scheduling decisions a crucial aspect in terms of productivity.

Motivated by a real-world application arising in a food packaging company located in the city of Reggio Emilia (Italy), in this chapter we consider a scheduling problem that consists in assigning printing jobs to a heterogeneous set of parallel flexographic printer machines, with the aim of minimizing a weighted sum of total weighted tardiness and total setup times. The problem also requires to determine

the schedule of the jobs on each machine, and the location of the tools in the magazines before the execution of each job takes place.

The resulting problem is not only $\mathcal{NP}$-hard (see Section 5.2), but also very difficult in practice. To solve it, we developed a GRASP, a metaheuristic that obtained good computational results on a large number of optimization problems (see, e.g., Resende and Ribeiro [419]). The GRASP uses a tailored randomized procedure to build initial solutions, and then brings them to local optima by means of a set of local search procedures. In addition, the GRASP is also equipped with a preprocessing method that allows to potentially reduce scheduling complexity by grouping specific jobs that are similar in their required tools. To obtain the best balance between computational effort and solution quality, the assignment of the tools to the magazines is computed in a heuristic way that resembles the manual approach in use at the company. After extensive computational testing, the GRASP has been embedded into a simple interface and delivered to the company, which is now using it on a regular basis to support their weekly scheduling decisions.

The remainder of the chapter is organized as follows. In Section 5.2, we briefly review the related literature. In Section 5.3, we formally describe the problem that we face. In Section 5.4, we present the details of the GRASP algorithm that we implemented. In Section 5.5, we provide the outcome of computational experiments carried out on a set of 25 real-world instances, each corresponding approximately to a week of production and derived from the everyday activity of the company. We compare the performance of the GRASP algorithm with the constructive heuristic CGHA and the local search procedure ISHA presented as a preliminary work at AMPS 2021 (Iori, Locatelli, and Locatelli [275]). Numerical tests demonstrate the effectiveness of the GRASP algorithm, which outperforms by a considerable margin both previous heuristics. Moreover, we conduct a comparative analysis with company solutions which attests that the GRASP algorithm is always capable of providing better quality solutions and outperforms, by a considerable margin, the quality of the company solutions. Finally, in Section 5.6, we draw some concluding remarks and future research directions.

5.2 Brief literature review

The problem that we face, called the *Parallel Print Machine Problem with Setup Times* (PMPST for short in the following), requires to (i) assign printing jobs to heterogeneous machines, (ii) establish the sequence in which the jobs are processed on each machine, and (iii) determine the positions of the required tools in the magazines of the machines, with the aim of minimizing a weighted sum of total weighted tardiness and total setup time. The problem we solve, indeed, represents a large variety of scheduling applications on parallel machines, where tool management represents a major source of complexity, and decision makers aim at minimizing both setup times and weighted tardiness. In this section we provide a brief review of the related applications.

The PMPST lies in the field of unrelated parallel machine problems with sequence-dependent setup times, for which we refer the interested reader to the surveys by Allahverdi et al. [17, 18, 22] and by Durasević and Jakobović [167]. Within this field, and using the three-field notation for scheduling problems by Graham et al. [213], the problem that most resembles the PMPST is the $R|ST_{sd}|\alpha TWT + \beta TST$, where TWT denotes the total weighted tardiness, TST the total setup time, and α and β are input parameters used to combine tardiness and setup times into a

unique objective function. The major difference between the two problems is that the $R|ST_{sd}|\alpha TWT + \beta TST$ considers setup times that only depend on the pair of consecutive jobs. This is equivalent to considering just the tools required by the current job and by the preceding one to calculate the setup times between the two jobs. In the PMPST, instead, it is often beneficial to leave a certain tool unused in a printer magazine only to use it again a few jobs later.

We could not find literature aimed at solving the $R|ST_{sd}|\alpha TWT + \beta TST$, but there are works devoted to the $R|ST_{sd}|C_{\max}$, i.e., the problem variant in which the goal is to minimize the makespan. In detail, Huang et al. [270] introduced a firefly algorithm with courtship learning, whereas Jovanovic and Voß [287] proposed a GRASP enhanced by a *Fixed Set Search* (FSS) metaheuristic. The basic concept of the FSS heuristic consists in fixing a set of elements that frequently appear in high-quality solutions and generating new solutions that contain such elements. In Section 5.4.2, we apply a similar, although simpler, concept to group similar jobs in a preprocessing phase.

In the production scheduling literature, just a limited number of studies include aspects of history-dependent scheduling, where setup times for a job are affected by the aggregate activities of all predecessors (see, e.g., Lee, Lei, and Pinedo [332] and Dayama et al. [146]). However, the information about aggregate activities is still not sufficient to calculate the setup time between two consecutive jobs exactly, since this information does not allow to completely know which are precisely the tools equipped by a machine before processing the current job.

A problem that has been largely studied in the literature and is more closely related to the PMPST is the *Tool Switching Problem with non-Uniform Setup Times* (SSPNU), a particular case of the well-known *Tool Switching Problem* (ToSP) (see Laporte, Salazar-González, and Semet [329], Crama et al. [125], and Calmels [73]). The ToSP consists in optimally sequencing jobs and in assigning tools to a capacitated magazine in order to minimize the number of tool switches. This is equivalent to considering unit-time setups. In the SSPNU, instead, the setup times required to switching the tools are dependent on the specific switched tools, and the objective function is to minimize the total setup time.

In the ToSP, the position of the tools in the magazine is irrelevant (see Laporte, Salazar-González, and Semet [329]) and this information can be thus neglected. On the other hand, this information is extremely necessary in the SSPNU, since the setup time depends on the switched tools at the same magazine slot. Another problem variant of the ToSP with unit-time setups but in which the information about the position of the tools in a magazine cannot be neglected, arises in the case in which tools may require more than one slot. To tackle this problem, Tzur and Altman [467] proposed an ILP formulation and developed a heuristic procedure. Later, Van Hop [469] proposed a construction heuristic, while Crama et al. [126] proved that the problem is $\mathcal{NP}$-hard even when the order in which the jobs are processed is fixed.

To the best of our knowledge, most of the previous research on ToSPs concerned uniform switching time (see Calmels [73]) and the only works that take into consideration non-uniform setup times are those by Privault and Finke [409], Windras Mara et al. [482], and Iori et al. [278]. In [409], a max-flow-min-cost model is developed for solving the SSPNU. In [482], the SSPNU is solved by means of two ILP models, namely, a five-index formulation and a multicommodity flow formulation. Iori et al. [278] introduced two variants of the SSPNU in which tools must be sorted along the slot sequence respecting a given order. For each variant, they discussed its complexity and proposed an ILP model.

The PMPST considers the same setup times of the SSPNU, but it also comprises a number of additional features. Indeed, there are different typologies of tools, each magazine slot of a machine can only hold a specific typology of tool and has a limited capacity. Moreover, the PMPST considers non-identical parallel machines, which differ in tool magazine capacity, processing times, and setup times. A generalization of the ToSP in the context of non-identical parallel machines was recently studied by Calmels [74], who provided a mixed ILP and different iterated local search methods. In addition, incompatibilities among machines and jobs must be taken into account, as well as due dates for the jobs, which, if violated, cause a penalization (weighted tardiness) in the objective function. To this regard, the PMPST is thus a generalization of the SSPNU. Since the SSPNU is a generalization of the ToSP and the ToSP is $\mathcal{NP}$-hard (see, e.g., Crama et al. [125]), we can conclude that the PMPST is $\mathcal{NP}$-hard too. In addition, it is also very challenging to solve it in practice. In the next sections, we formally define the PMPST and then present the heuristic that we developed to solve it.

5.3 Problem description

In the PMPST, a set $J = \{1, \ldots, n\}$ of jobs has to be processed by a set $M = \{1, \ldots, m\}$ of heterogeneous flexographic printer machines along a scheduling period. Each machine $i \in M$ is available from Monday to Friday for 8 working hours per day and it can be simply paused at the end of each work shift and resumed the day after without any penalty cost (even during the printing phase). This implies that the scheduling period can be considered as a unique uninterrupted time interval.

A flexographic printer machine $i \in M$ is a flexible machine which can simultaneously hold m_i distinct tools able to perform three different types of operations: applying a specific color, embossing the printed material, and perforating the printed material. Let R be the set of all available tools. Since the company has a large number of tools, their availability is not an issue and so we assume that R is a countably infinite set which contains an infinite number of each tool. Let $R = C \cup E \cup P$, with C, E, and P be the disjoint sets of all available ink cartridges, embossing rollers, and perforating rollers, respectively.

Each machine has three different typologies of magazine slots. The first one is able to hold just ink cartridges (C), the second one just embossing rollers (E), and the last one just perforating rollers (P). More specifically, the number of ink cartridges that a machine $i \in M$ can hold is limited by the number c_i of color groups of the machine. In addition, a machine can equip at most e_i embossing rollers and p_i perforating rollers. The compatibility between the typology of the magazine slot and the typology of the mounted tool must always be respected. All tools required for processing a job should have been loaded into the magazine slots before starting the processing of the job. If a tool required by a job is already loaded into the magazine, then the setup time required for loading that tool is null. Otherwise, a positive setup time is incurred. The operation of loading and unloading an embossing roller requires a fixed setup time, and the same holds for the operation of loading and unloading a perforating roll. Instead, the setup time required for the operation of unloading and loading an ink cartridge in a color group depends on the two involved colors.

Each machine $i \in M$ is also characterized by the following elements:

- $J_i \subseteq J$: subset of jobs that can be printed by machine i;

- θ_i: printing speed of machine i;
- μ^i: availability date of machine i;
- ℓ_i: washing system type of machine i, either automatic ($\ell_i = 1$) or manual ($\ell_i = 0$).

Each job $j \in J$ requires a subset of ink cartridges $C_j \subseteq C$, a subset of embossing rollers $E_j \subseteq E$, and a subset of perforating rollers $P_j \subseteq P$. A printing job $j \in J$ is also characterized by the following elements:

- $M_j \subseteq M$: subset of machines capable of printing job j;
- v_j: length of job j, measured in linear meters;
- p_{ij}: processing time of job j on machine $i \in M_j$, computed as $p_{ij} = v_j/\theta_i$;
- d_j: due date of job j;
- w_j: tardiness penalty assigned to job j.

A sequence of jobs $S^i = (j^i_1, \ldots, j^i_{n_i})$ is feasible for machine $i \in M$ if $i \in M_j$ for each $j \in S^i$. A feasible sequence S^i of jobs defines a schedule for machine $i \in M$ and can be used to compute start and completion times of all jobs. A set of feasible sequences $S = \{S^i : i \in M\}$ is a feasible schedule for J if it forms a partition of J.

Given a schedule S^i for machine $i \in M$, a *tool switching schedule for machine i* is a function $\xi \colon S^i \times \{1, \ldots, m_i\} \to R$ such that, for each $j \in S^i$:

- $C_j \cup E_j \cup P_j \subseteq \bigcup_{k \in 1}^{m_i} \{\xi(j,k)\}$, i.e., the magazine of machine i, before starting job j, contains all the tools required by job j;
- the compatibility between the typology of the *k-th* magazine slot and the typology of the tool $\xi(j,k)$ is respected, for each magazine slot position $k \in \{1, \ldots, m_i\}$.

In other words, given a schedule S^i for machine $i \in M$, for each magazine slot position $k \in \{1, \ldots, m_i\}$ and job $j \in S^i$, the function $\xi(j,k)$ defines which is the tool in the *k-th* magazine slot during the processing of job j.

In the PMPST, the switching costs are determined by the sequence of job processing and by the tool switching schedule. These factors have to be simultaneously taken into account by determining a feasible schedule S and a tool switching schedule for each machine $i \in M$ with the main goal to minimize a weighted sum of total weighted tardiness and total setup times. More specifically, the PMPST consists in finding a feasible schedule that minimizes

$$z = \beta WT + (1 - \beta)TST, \tag{5.1}$$

where $\beta \in [0,1]$ is an input parameter used to balance the two components of the objective function, namely

- $WT = \sum_{i \in M} \sum_{j \in S^i} w_j T_j$;
- $TST = \sum_{i \in M} \sum_{h=1}^{n_i - 1} s^i_{j_h j_{h+1}}$.

Here, T_j denotes the tardiness of a job j in S^i, while $s^i_{j_h j_{h+1}}$ denotes the setup time needed to switch from job j^i_h to job j^i_{h+1}. It turns out that the problem of calculating such setup time is not trivial and may require a large computational effort. To be able to solve practical industrial instances, in Section 5.4.1, we propose a heuristic evaluation of the setup times. The tardiness is first computed as a number of days from the due date and then switched to minutes, so both WT and TST are expressed in minutes.

With a slight abuse of notation, given a feasible job sequence $S^i = (j^i_1, \ldots, j^i_{n_i})$ for machine $i \in M$, we let $z(S^i)$ denote the value $\beta \cdot \sum_{j \in S^i} w_j T_j + (1-\beta) \cdot \sum_{h=1}^{n_i - 1} s^i_{j_h j_{h+1}}$. Note that $z(S^i)$ can be viewed as the contribution of machine i to the objective function. The aim of the PMPST is to find a job and tool switching schedule that is feasible and minimize (5.1).

Example 5.3.1. We provide a numerical example to better illustrate the addressed problem. We consider an instance composed of two machines $M = \{1, 2\}$, whose characteristics are reported in Table 5.2, and five jobs $J = \{1, 2, 3, 4, 5\}$ requiring tools in $C = \{$Black, Cyan, Magenta, Pink, Red, Yellow, White$\}$, whose characteristics are detailed in Table 5.1. All times are given in minutes. For simplicity, we assume that the setup time required for the operation of unloading and loading an ink cartridge is fixed to 25 for each color involved. Now, we consider the feasible schedule $S = \{S^1 = (3, 1, 5), S^2 = (2, 4)\}$ and the tool switching schedule for machine 1 and 2 given in Tables 5.3 and 5.4, respectively.

Table 5.1: Characteristics of considered jobs

j	C_j	M_j	v_j	p_{1j}	p_{2j}	d_j	w_j
1	{Red, Yellow}	{1,2}	15000	125	150	480	2
2	{Yellow, Cyan, Red}	{1,2}	12000	100	120	480	5
3	{Yellow, Cyan, Red, Black}	{1}	18000	150	180	480	10
4	{Yellow, Cyan, Magenta}	{2}	30000	250	300	480	1
5	{ White, Pink, Magenta, Black}	{1}	18000	150	180	960	7

Table 5.2: Characteristics of considered machines

i	c_i	J_i	θ_i	μ^i	l_i
1	4	{1,2,3,5}	120	300	1
2	3	{1,2,4}	100	350	1

Table 5.3: Tool switching schedule for machine 1

S^1 \ Slot	1	2	3	4
3	$\xi(3,1) =$ Yellow	$\xi(3,2) =$ Cyan	$\xi(3,3) =$ Red	$\xi(3,4) =$ Black
1	$\xi(1,1) =$ Yellow	$\xi(1,2) =$ Cyan	$\xi(1,3) =$ Magenta	$\xi(1,4) =$ Black
5	$\xi(5,1) =$ White	$\xi(5,2) =$ Pink	$\xi(5,3) =$ Magenta	$\xi(5,4) =$ Black

Table 5.4: Tool switching schedule for machine 2

S^2 \ Slot	1	2	3
2	$\xi(2,1) =$ Yellow	$\xi(2,2) =$ Cyan	$\xi(2,3) =$ Red
4	$\xi(4,1) =$ Yellow	$\xi(4,2) =$ Cyan	$\xi(4,3) =$ Magenta

For each job, the time required for the printing phase is reported in columns p_{1j} and p_{2j} of Table 5.1. Now we calculate, for each job, the time required for the machine setup phase. Initially, we assume that the slots of both machines are empty, so there is no switching operation to perform for processing jobs 3 and 2. Thus, the time required for the machine setup phase of jobs 2 and 3 is null. On the other hand, the time required for the machine setup phase of jobs 1 and 4 is 25 (there is a switch operation to perform in both cases) and 50 for the machine setup phase of job 5 (there are two switch operations to perform) and thus $TST = 100$.

Since the working time of the machines is known (480 minutes per day) and the jobs in S^1 and S^2 are performed one after another, start and completion times (and thus also the tardiness) of all jobs can be directly calculated as follows. First, we consider jobs in S^1:

- job 3 starts at time μ^1 and its completion time is at time $\mu^1 + p_{13} = 300 + 150 = 450$;
- job 1 starts at time 450 and its completion time is at $450 + s^1_{\text{Red,Magenta}} + p_{11} = 450 + 25 + 125 = 600$;
- job 5 starts at time 600 and its completion time is at $600 + s^1_{\text{Yellow,White}} + s^1_{\text{Cyan,Pink}} + p_{15} = 600 + 25 + 25 + 150 = 800$.

While, for jobs in S^2:

- job 2 starts at time μ^2 and its completion time is at time $\mu^2 + p_{22} = 350 + 120 = 470$;
- job 4 starts at time 470 and its completion time is at $470 + s^1_{\text{Red,Magenta}} + p_{24} = 470 + 25 + 300 = 795$.

Figure 5.1 reports a graphical representation of schedule S. It turns out that just jobs 1 and 4 have 120 and 315 minutes of tardiness, respectively. So, $WT = 120 \cdot w_1 + 315 \cdot w_4 = 120 \cdot 2 + 315 \cdot 1 = 555$. If we set $\beta = 0.2$ (for more details see Section 5.5.5), we obtain $z = \beta WT + (1 - \beta)TST = 0.2 \cdot 555 + 0.8 \cdot 100 = 191$.

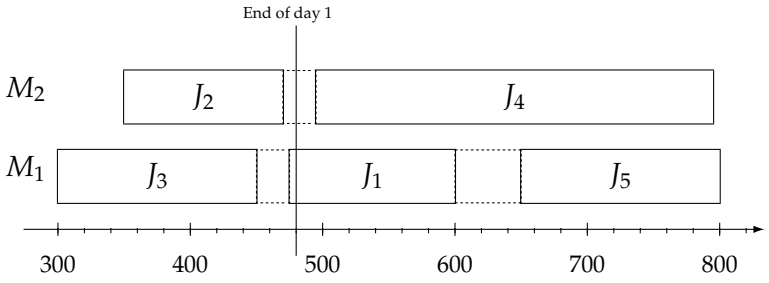


Figure 5.1: Graphical representation of the feasible schedule $S = \{S^1 = (3,1,5), S^2 = (2,4)\}$

5.4 A GRASP Approach for the PMPST

The PMPST is $\mathcal{NP}$-hard because it generalizes the ToSP, which is known to be $\mathcal{NP}$-hard (see, e.g., Crama et al. [125]). Furthermore, the PMPST (which considers multiple non-identical machines and also requires to find an optimal tool switching schedule for different tool typologies) is very challenging to solve in practice. In this section, we present the GRASP that we developed to solve the PMPST. Its pseudocode

is provided in Algorithm 2. In a few words, the GRASP is an iterative process in which each iteration consists of two phases: construction and local search. In our implementation, the GRASP is combined with a preprocessing method, described in Section 5.4.2, that allows to potentially reduce scheduling complexity by grouping specific jobs that are similar in their required tools into super-jobs. The construction phase, described in Section 5.4.3, builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. As detailed in Section 5.4.4, the local search phase is composed of four procedures, namely LS1, LS2, LS3, and LS4. At each iteration (step 4) a feasible solution S is found by means of the construction phase, and then (step 5) S is brought to a local minimum by sequentially invoking LS1, LS2, and LS3 in this order. Super-jobs are then unpacked into jobs right before the LS4 procedure is invoked (step 6). Finally, at step 9, the GRASP returns S_{best}, the best solution found in $I_{\max} \in \mathbb{N}$ iterations.

Algorithm 2: GRASP metaheuristic for the PMPST

1 **input**: M, J, $I_{\max}$, threshold parameter α, availability date μ^i for each $i \in M$;
2 create super-jobs by means of Algorithm 3;
3 **for** $k \in \{1, \ldots, I_{\max}\}$ **do**
4 construct new feasible solution S using Algorithm 4;
5 bring S to a local minimum by sequentially invoking LS1, LS2, and LS3;
6 unpack all super-jobs in S and invoke LS4;
7 compare S with S_{best} and possibly update S_{best};
8 **end**
9 **return** S_{best}

In the PMPST, even just the problem of calculating the setup times is not trivial. In the next section, we describe a simple and effective heuristic method for evaluating the setup times that is repeatedly used within the GRASP.

5.4.1 Heuristic setup time Evaluation

In our work, we decided to adopt a simple but quick and good-enough evaluation of the setup times. This evaluation is used in all the steps of the GRASP.

First, a specific policy is introduced for determining a tool switching schedule. Let us consider a machine with a given loading of the tools in its magazine, and suppose we need to determine the tool switching operations required to process the next job. According to our policy, if a magazine is full and a tool $c \in C$ required for printing the new job is not in the magazine, then the first tool $c' \in C$ in the magazine that is not required by the new job is removed and switched with c. If, instead, a tool $c \in C$ required for printing the new job is not in the magazine and there are one or more empty magazine slots that can hold a tool in C, then c is loaded in the first of these empty slots. The two policies for defining the tool switching operations for tools in E and P are similarly defined.

Given two consecutive jobs j_h and j_{h+1} of a schedule S^i for machine $i \in M$, let us recall that $s^i_{j_h j_{h+1}}$ denotes the setup time needed to switch from job j_h to job j_{h+1}. The setup times (i.e., the setup times required to switch the required tools in the machine magazine) are evaluated as follows. A color washing time is added to $s^i_{j_h j_{h+1}}$ for each $c \in C_{j_{h+1}}$ that is not in the magazine after the end of job j_h (in case $\ell_i = 0$) or if there is at least one tool $c \in C_{j_{h+1}}$ that is not in the magazine after the end of job j_h (in case $\ell_i = 1$). Indeed, if a printer machine equips an automatic washing system, then the

color groups can be washed in parallel, otherwise they have to be washed one by one in sequence. Moreover, if there is a tool $c \in C_{j_{h+1}}$ that is not in the magazine after the end of job j_h and that corresponds to the color white or to a special varnish, then an additional penalty time is considered. Indeed, the process to wash a color group and to refill it with white ink or a special varnish requires the complete cleaning of the corresponding color group, with the effect of significantly raising the setup times. Furthermore, for each tool $e \in E_{j_{h+1}}$ and for each tool $p \in P_{j_{h+1}}$ that are not in the magazine after the end of job j_h, a fixed tool switching time is added to $s^i_{j_h j_{h+1}}$.

5.4.2 Preprocessing method to Group Specific Jobs

For practical instances of the PMPST, where there is a large variety of jobs, it is convenient to group specific jobs, that tend to be similar in some of their required tools, into super-jobs. Indeed, there are some specific tools for which their tool switching operations are significantly time consuming. Therefore, it is convenient to sequentially process all the jobs requiring these particular tools, so as to minimize the resulting tool switching operations. The creation of super-jobs allows to identify a relatively small set of good schedules and avoid searching through a large number of alternatives, thus reducing the scheduling complexity. The grouping of jobs is a desirable strategy and allows to deal with smaller and more tractable problems. This general idea, which was highly appreciated by the company and is confirmed in practice by the computational results showed in Section 5.5, has also been applied in other related problems (see, e.g., Burger et al. [70]). On the other hand, it is worth noting that the grouping process may lead to an increase of total tardiness. To reduce this downside, we introduce a parameter γ that identifies the maximum number of jobs that can compose a super-job. For $\gamma = 1$, the prepossessing method that produces the grouping process, described in Algorithm 3, does not create any super-jobs.

Let us describe the preprocessing method in detail. Given a tool $t \in R$, let J^t be the set of all jobs requiring t. Given a machine $i \in M$, let $J^t_i = \{j \in J^t : i \in M_j\}$ be the subset of jobs in J^t which can be printed by machine i, and let $R' \subseteq R$ be the set of all tools for which switching operations are highly time consuming. For each tool $t \in R'$, at step 3 Algorithm 3 selects a machine $i' \in M$ such that $|J^t_{i'}| = \max_{i \in M} |J^t_i|$. Then, at step 5, the sequence $S^t_{i'}$ composed of all jobs in $J^t_{i'}$, ordered by due date (in order to reduce the effect of raising the total tardiness due to the grouping process), is created. If $S^t_{i'}$ contains more than γ jobs, then, at step 7, the last $|S^t_{i'}| - \gamma$ jobs are eliminated from $S^t_{i'}$ and the set $J^t_{i'}$ is updated accordingly. At step 9, the sequence $S^t_{i'}$ is considered as a unique job j', called a *super-job* and having the following characteristics:

- $C_{j'} = \bigcap_{j \in S^t_{i'}} C_j$, $E_{j'} = \bigcap_{j \in S^t_{i'}} E_j$, and $P_{j'} = \bigcap_{j \in S^t_{i'}} P_j$. In doing so, we are sure that tool t, for which the switching operation is highly time consuming, is a tool of super-job j' and that the magazine capacity of machine i' is not exceeded by the tools of j'. It is worthwhile to remark that the objective function value of a solution has to be calculated in a schedule which does not contain any super-jobs. Before calculating a solution value, all the super-jobs have to be unpacked until there are no more super-jobs in the schedule. In order to unpack a super-job $j' \in S^i$, it is sufficient to replace j' with the sequence S^t_i created at steps 5–7.

- $M_{j'} = \{i \in M : i \in M_j \text{ for all } j \in J^t_{i'}\}$ is the subset of machines which are able to print all the jobs in $J^t_{i'}$;

- $p_{j'i}$ is the sum of the time required by machine i' to print all the jobs in $J^t_{i'}$ (i.e., $\sum_{j \in J^t_{i'}} v_j/\theta_i$) and the sum of the setup times between two consecutive jobs in $j \in J^t_{i'}$ required by machine i';
- $d_{j'} = \min\{d_j : j \in J^t_{i'}\}$;
- $w_{j'} = \max\{w_j : j \in J^t_{i'}\}$.

The operation of creating super-jobs is iteratively repeated until it is not possible to create further super-jobs.

Algorithm 3: Preprocessing method for creating super-jobs

```
1  input: M, J, R' ⊆ R tools for which switching operations are highly time
     consuming;
2  for t ∈ R' do
3      select i' ∈ M such that |J^t_{i'}| = max_{i∈M} |J^t_i|;
4      while |J^t_{i'}| > 1 do
5          create the sequence S^t_{i'} composed of jobs in J^t_{i'}, ordered by due date;
6          if |S^t_{i'}| > γ then
7              eliminate from S^t_{i'} the last |S^t_{i'}| − γ jobs;
8          end
9          from the sequence S^t_{i'}, create the super-job j';
10         remove from J all jobs in S^t_{i'} and add the super-job j';
11     end
12 end
13 return J
```

5.4.3 Constructive Phase

The constructive phase of the proposed GRASP is presented in Algorithm 4. This method assigns, at each iteration, a job to the first available machine, generating, for each machine $i \in M$, a feasible sequence of jobs S^i. From step 5 to step 11, a set L_i of jobs is created by selecting from J_i the jobs behind schedule, i.e., $\{j \in J_i : d_j < \mu^i\}$. If there are no jobs behind schedule, then L_i is created by selecting from J_i the feasible jobs that can only be printed by machine i, i.e., $\{j \in J_i : |M_j| = 1\}$. If L_i is still empty, then we set $L_i = J_i$. At Step 13, the elements in L_i are evaluated by the objective function z defined in (5.1). This leads to the creation of a *Restricted Candidate List* (RCL), formed by the jobs in L_i whose addition at the end of the current sequence of jobs S^i produces the smallest incremental costs. More precisely, let $\delta_{\min}$ and $\delta_{\max}$ be, respectively, the smallest and the largest incremental costs produced by adding a job in L_i at the end of S^i and let $\alpha \in [0,1]$ be a threshold input parameter. The RCL is created by considering all jobs $j \in L_i$ whose incremental costs produced by adding job j at the end of S^i is in the range $[\delta_{\min}; \delta_{\min} + \alpha(\delta_{\max} - \delta_{\min})]$. At step 14, the job to be incorporated into the partial solution is randomly selected from those in the RCL. At step 16, the availability date μ^i of the machine is updated, as well as the magazine state of machine i (according to the policy introduced above in Section 5.4.1 for determining the tool switching schedule). Finally, at step 18, the feasible schedule $S = \{S^i : i \in M\}$ is returned.

Algorithm 4: GRASP Constructive phase

input: M, J, threshold parameter α, availability date μ^i for each $i \in M$;
$S^i = \emptyset$ $(i \in M)$;
while $J \neq \emptyset$ **do**
  select a machine $i \in M$ such that $\mu^i = \min_{i' \in M}\{\mu^{i'} : J_{i'} \neq \emptyset\}$ and $J_i \neq \emptyset$;
  $L_i := \{j \in J_i : d_j < \mu^i\}$;
  if $(L_i = \emptyset)$ **then**
    $L_i = \{j \in J_i : |M_j| = 1\}$;
  end
  if $(L_i = \emptyset)$ **then**
    $L_i = J_i$;
  end
  build RCL (using threshold parameter α) from elements in L_i;
  select a job $j \in$ RCL at random;
  add job j at the end of sequence S^i;
  remove job j from J;
  update μ^i and the magazine state of machine i;
end
return $S = \{S^i : i \in M\}$.

5.4.4 Local Search Phase

To improve the solution S built by Algorithm 4, four local search procedures, namely LS1, LS2, LS3, and LS4, are applied on S.

Procedure LS1 is based on an intra-machine swap neighborhood: it selects a machine $i \in M$, two job positions h and g $(1 \leq h < g \leq n_i)$ in the sequence S^i of jobs, and then generates a new sequence $S^i_{(h,g)} = (j^i_1, \ldots, j^i_g, \ldots, j^i_h, \ldots, j^i_{n_i})$ by switching job j^i_h with job j^i_g (see Figure 5.2). The generated solution is better than the current one if $z(S^i_{(h,g)}) < z(S^i)$.

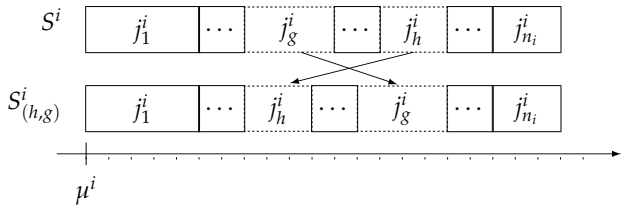


Figure 5.2: Intra-machine swap move used in LS1

Procedure LS2 is based on an inter-machine insertion neighborhood,: it selects two machines i, $i' \in M$ and two job positions h $(1 \leq h \leq n_i)$ and g $(1 \leq g \leq n_{i'})$ in the sequences of jobs S^i and $S^{i'}$, respectively, and, if $i' \in M_{j^i_h}$, removes job j^i_h from S^i (generating $S^i_{(j^i_h,h^-)} = (j^i_1, \ldots, j^i_{h-1}, j^i_{h+1}, \ldots, j^i_{n_i})$) and inserts j^i_h before the g-th position of $S^{i'}$ (generating $S^{i'}_{(j^i_h,g^+)} = (j^{i'}_1, \ldots, j^{i'}_{g-1}, j^i_h, j^{i'}_g, \ldots, j^i_{n_{i'}})$) (see Figure 5.3). The generated solution is better than the current one if $z(S^i_{(j^i_h,h^-)}) + z(S^{i'}_{(j^i_h,g^+)}) < z(S^i) + z(S^{i'})$.

Procedure LS3 is based on an intra-machine sub-sequence insertion neighborhood. Given a feasible sequence of jobs $S^i = (j^i_1, \ldots, j^i_{n_i})$ and two job positions h and g $(1 \leq h < g \leq n_i)$ of S^i, a sub-sequence of consecutive jobs $S^i_{(h-g)} = (j^i_h, \ldots, j^i_g)$

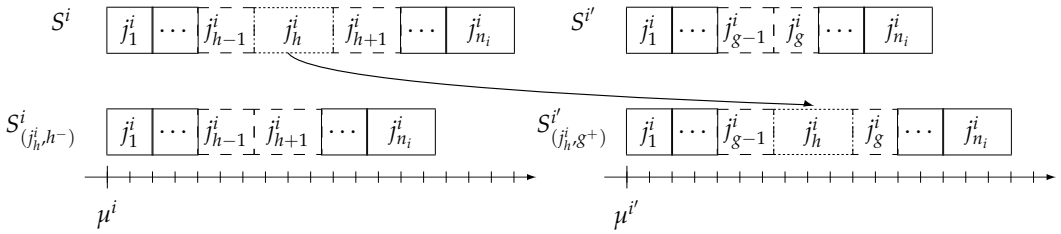


Figure 5.3: Inter-machine insertion neighborhood move used in LS2

is t-maximal if $s^i_{j_{h-1}j_h} > t$ (or $h = 1$), $s^i_{j_g j_{g+1}} > t$ (or $g = n_i$), and $s^i_{j,j+1} \leq t$ (for each $j = h, \ldots, g-1$). Stated in another way, a t-maximal sub-sequence is a maximal sequence of consecutive jobs to be executed on machine i whose setup times do not exceed t. By definition, a t-maximal sub-sequence $S^i_{(h-g)} = (j^i_h, \ldots, j^i_g)$ of S^i is composed of at least two jobs. In LS3, first, for each $i \in M$, a list L_i of all t-maximal sub-sequences of S^i is created. Then, LS3 selects a machine $i \in M$, a t-maximal sub-sequence $S^i_{(h-g)}$ in L_i, a job position k ($1 \leq k < h$ or $g < k \leq n_i$), and then generates a new sequence $S^i_{k,(h-g)} = (j^i_1, \ldots, j^i_{k-1}, j^i_h, \ldots, j^i_g, j^i_k \ldots, j^i_{n_i})$ by inserting the sub-sequence $S^i_{(h-g)}$ before the k-th position of S^i (see Figure 5.4). The generated solution is better than the current one if $z(S^i_{k,(h-g)}) < z(S^i)$.

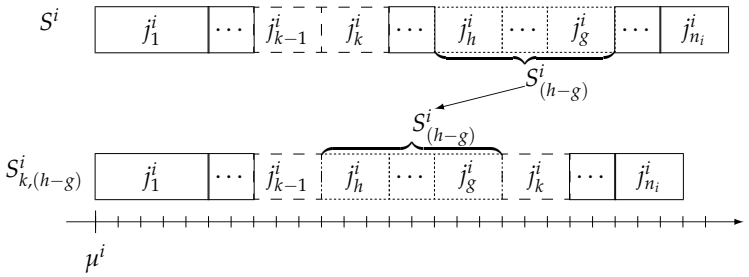


Figure 5.4: Intra-machine sub-sequence insertion move used in LS3

Procedure LS4 is based on an intra-machine swap neighborhood similar to LS1 but involving just jobs composing a super-job. Procedure LS4 selects a machine $i \in M$, a super-job in S^i composed of the sequence $S^t_{i'}$, two job positions h and g ($1 \leq h < g \leq n_i$) in the sequence of jobs $S^t_{i'}$, and then generates a new sequence $S^i_{(h,g)} = (j^i_1, \ldots, j^i_g, \ldots, j^i_h, \ldots, j^i_{n_i})$ by switching job j^i_h with job j^i_g. The generated solution is better than the current one if $z(S^i_{(h,g)}) < z(S^i)$.

All four local search procedures operate in a first-improvement manner. For each $i \in M$, all the possible moves are considered, evaluating all the solutions in the neighborhood until an improving solution (if any) is found. In case an improving solution is detected, the current solution S is updated and the local search is re-executed on S. If, instead, no improving solution is found, then the local search returns the current local optimal solution.

5.5 Computational Results

In this section, we present extensive computational experiments on various instances derived from a real-world application. All the algorithms have been coded in Python version 3.9.2 and run on a computer with Intel(R) Xeon(R) Gold 6130 with CPU 2.10 GHz and RAM 16 GB, using Windows 10 Pro 64-bit.

5.5.1 Test Instances and Parameters Setting

We consider 25 real-world instances, each corresponding to a week of production at the company. The number of jobs spans from 50 to 116, while the number of machines is 10. The parameter β in the objective function has been set to 0.2, as required by the company (an evaluation obtained by attempting different β values is provided in Section 5.5.5 below). The parameter t in the intra-machine sub-sequence insertion moves is set to 125, because it has been noticed that with this value LS3 produces good-quality solutions.

In the GRASP algorithm, the threshold parameter α has been set to 0.1 and the number $I_{\max}$ of iterations to 10. These two values have been chosen according to preliminary experimental trials and guarantee to obtain stable and good results on most of the tested instances within relatively small computing times. More specifically, Table 5.5 shows the average values of the solutions obtained by the GRASP over the 25 instances by varying $\alpha \in \{0, 0.1, 0.2, \ldots, 1\}$. Columns z, TST, and WT give the average value of the objective function, the total setup time, and the total weighted tardiness, respectively. These values are used for drawing, in the form of a diagram, Figure 5.5, which confirms that setting α to 0.1 is the best choice.

Figure 5.6 is instead obtained after running the GRASP with $I_{\max}$ set to 50. First, we calculated the percentage gap between the best current solution value in a given iteration of the GRASP and the solution value returned at the end of the execution. Then, we calculated the average gap over the 25 instances and reported it on the left vertical axis of Figure 5.6. We can observe that after 10 iterations the curve is almost flat. Moreover, in the right vertical axis of Figure 5.6 the average time necessary to complete each iteration is reported. As expected, Figure 5.6 shows that the run time of the GRASP is a linear function of $I_{\max}$. From a managerial point of view, we point out that a disruption may occur during the everyday activity, so the company may be interested in re-running the algorithm to obtain a modified schedule, preferably in a short time. Setting the parameter $I_{\max}$ to 10 allows the GRASP to quickly converge (in less than 200 seconds, on average) to good solution values.

The value of γ has been set to 5, as this value guarantees to deal with tractable problems in practice without increasing the total tardiness. Table 5.6 shows the average values of the solutions obtained by the GRASP over the 25 instances by varying $\gamma \in \{0, 1, 2, \ldots, 10\}$. Columns z, TST, WT, and sec give the average value of the objective function, the total setup time, the total weighted tardiness, and the execution run-time, respectively. The values in the column γ and z are reported as a diagram in Figure 5.7, which confirms that setting γ to 5 is the optimal choice. From a managerial point of view, it is evident that γ has a crucial impact on the quality of the solutions obtained. Large values of gamma allow the company to save setup time, but at the expense of additional weighted tardiness. From a managerial point of view, when facing similar applications we suggest to perform a preliminary analysis with different γ values. Its impact is very relevant also for the applicability of the problem, because time increases in a relevant way when γ decreases.

5.5.2 Evaluation of the GRASP Algorithm

Table 5.7 shows the results of the first round of experiments, which we performed with the aim of comparing the GRASP without preprocessing (obtained by setting $\gamma = 1$) against the algorithms CGHA and ISHA by Iori, Locatelli, and Locatelli

α	z	TST	WT
0.0	9322.7	10726.6	3707.0
0.1	9139.9	10509.4	3661.9
0.2	9183.4	10566.8	3650.0
0.3	9188.1	10551.6	3734.0
0.4	9194.7	10580.8	3650.1
0.5	9253.4	10631.2	3742.0
0.6	9273.4	10679.2	3650.1
0.7	9265.4	10673.2	3634.1
0.8	9307.0	10711.2	3690.3
0.9	9336.1	10773.6	3586.1
1.0	9344.6	10783.2	3590.0

Table 5.5: Average GRASP solution values over the 25 instances, by varying the threshold parameter α

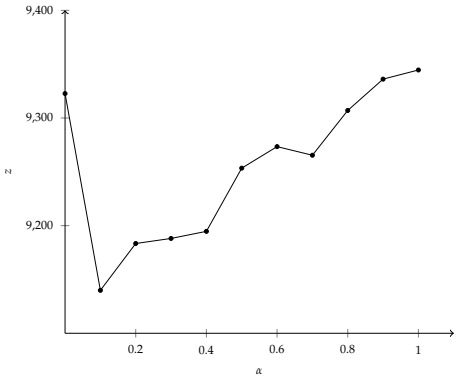


Figure 5.5: Average z value over the 25 instances, by varying the threshold parameter α

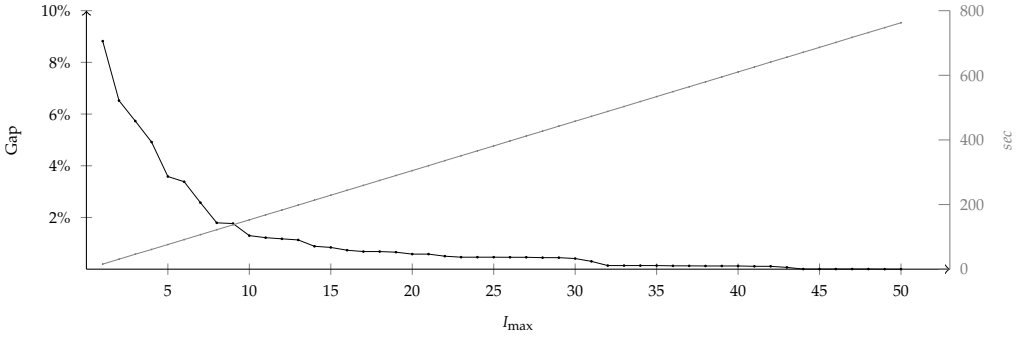


Figure 5.6: Evolution of the percentage gap with respect to the best known solution (on the left vertical axis) and the average time necessary to complete each iteration (reported in gray on the right vertical axis)

γ	z	TST	WT	sec
0	9685.1	11216.0	3561.6	347.2
1	9613.6	11122.6	3577.5	300.1
2	9363.7	10804.0	3602.6	273.2
3	9350.7	10783.4	3620.1	222.8
4	9237.2	10638.2	3633.2	181.9
5	9137.0	10509.8	3645.8	177.9
6	9175.1	10542.6	3705.1	165.9
7	9255.7	10654.2	3861.5	149.1
8	9228.3	10572.4	3851.9	132.1
9	9256.1	10582.4	3950.9	130.8
10	9264.1	10592.4	3950.9	131.3

Table 5.6: Average GRASP solution values over the 25 instances, by varying parameter γ

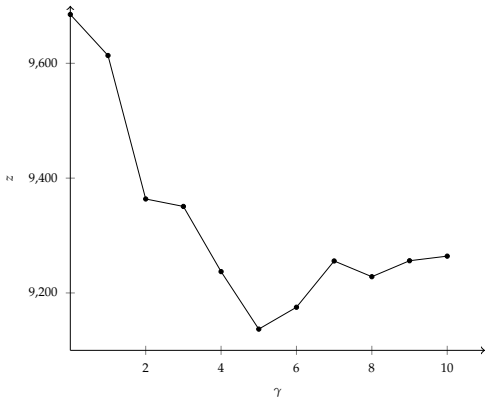


Figure 5.7: Average z value over the 25 instances, by varying parameter γ

[275], which represent the state of the art for the problem. Note that the CGHA corresponds to the constructive phase of the GRASP with $\alpha = 0$ and without preprocessing, while the ISHA corresponds to the GRASP, again with $\alpha = 0$ and without

preprocessing. Entries in the columns $|J|$ and $|M|$ give the number of jobs and of machines, respectively. Columns z, TST, and WT give the value of the objective function, the total setup time, and the total weighted tardiness, respectively. Entries in the columns *sec* exhibit the execution run-time, expressed in seconds. Entries in the row AVG provide the average values of each column, while GAP presents the percentage gaps from the mean of the solution values produced by the CGHA. Table 5.7 shows that the CGHA finds feasible solutions in a very short time (always less than three tenths of a second). the ISHA improves the quality of the solution found by the CGHA on average by 12%, with percentage improvements on TST and WT equal to 9.9% and 32.2%, respectively. The GRASP without preprocessing is always able to produce better solutions than the ISHA (and therefore also of the CGHA), with an improvement in the average function value of 18.4%, and a percentage reduction of TST and WT equal to 16.9% and 33.4%, respectively.

Table 5.7: Computational results of CGHA, ISHA, and GRASP without preprocessing

Inst.	$\|J\|$	$\|M\|$	CGHA [275]				ISHA [275]				GRASP no preproc.			
			z	TST	WT	*sec*	z	TST	WT	*sec*	z	TST	WT	*sec*
I01	50	10	7188.6	8485	2003.0	0.1	6272.7	7390	1803.3	5.0	5460.6	6325	2003.2	50.4
I02	76	10	9938.7	11855	2273.6	0.1	9213.1	11100	1665.5	20.7	8901.0	10710	1664.8	201.6
I03	78	10	9488.4	10725	4542.1	0.1	8044.1	9120	3740.3	18.2	7691.5	8605	4037.6	179.9
I04	82	10	9544.2	11295	2540.8	0.1	8383.3	9970	2036.3	23.8	7422.8	8745	2134.0	219.1
I05	83	10	11682.2	10740	15451.2	0.1	9423.8	9265	10058.8	49.3	8826.7	8470	10253.7	500.4
I06	85	10	11285.4	13375	2927.0	0.1	10417.5	12390	2527.3	21.1	8828.9	10405	2524.5	194.8
I07	85	10	12039.7	13540	6038.7	0.1	10823.3	12320	4836.7	23.5	9903.1	11345	4135.7	240.2
I08	86	10	12069.9	14270	3269.3	0.1	11401.3	13560	2766.3	22.9	10621.2	12610	2666.1	232.0
I09	87	10	9747.4	10850	5337.2	0.1	8455.1	10135	1735.3	28.0	7611.1	9105	1635.6	252.9
I10	89	10	10682.0	12730	2489.9	0.1	9449.7	11490	1288.3	31.3	8865.0	10735	1385.0	293.9
I11	89	10	12617.7	14035	6948.4	0.1	11526.5	13470	3752.4	57.1	10345.8	12045	3549.2	557.6
I12	89	10	13234.6	14130	9653.2	0.1	11179.9	12035	7759.4	28.2	10607.8	11395	7458.9	254.1
I13	91	10	13820.4	15635	6562.0	0.1	12404.7	14290	4863.3	27.7	11455.7	12955	5458.7	254.8
I14	93	10	11237.6	13710	1347.9	0.1	10305.7	12720	648.4	27.9	8744.7	10845	343.4	267.4
I15	97	10	11559.4	13295	4617.1	0.2	10479.7	12320	3118.3	36.3	9991.8	11735	3018.8	329.9
I16	98	10	14232.0	15625	8659.9	0.1	12530.8	13825	7353.8	38.6	10999.1	12060	6755.6	353.7
I17	99	10	11344.5	13850	1322.7	0.1	9598.1	11770	910.6	39.7	9573.2	11740	906.2	370.0
I18	102	10	11002.2	13010	2971.2	0.2	10025.1	11940	2365.6	33.6	9761.5	11685	2067.4	357.6
I19	102	10	12298.6	14285	4352.8	0.1	10974.9	13080	2554.5	42.6	10198.0	12235	2049.9	407.5
I20	105	10	13993.1	16180	5245.6	0.2	11672.7	13705	3543.5	44.1	11184.8	13145	3343.9	413.1
I21	107	10	12235.8	13255	8159.2	0.2	10534.0	11880	5150.2	49.7	9618.4	10685	5351.8	461.1
I22	109	10	14626.1	14545	14950.7	0.2	12054.3	12680	9551.6	46.8	11990.2	12575	9651.2	468.9
I23	110	10	13079.0	15320	4115.1	0.1	11522.7	14225	713.3	68.4	10386.2	12730	1011.2	668.1
I24	113	10	13103.4	14605	7097.0	0.1	11527.1	12835	6295.5	46.8	11086.8	12410	5793.8	447.3
I25	116	10	12757.6	15295	2607.9	0.2	11288.0	13910	799.9	70.4	10343.3	12680	996.7	703.3
AVG	92.8	10	11792.3	13385.6	5419.3	0.1	10380.3	12057	3673.5	36.1	9616.8	11119	3607.9	347.2
GAP							-12.0%	-9.9%	-32.2%		-18.4%	-16.9%	-33.4%	

The second round of experiments, whose results are given in Table 5.8, compares the solutions obtained by means of the CGHA+P, i.e., the CGHA combined with the preprocessing Algorithm 3, the ISHA+P, i.e., the ISHA combined with the preprocessing Algorithm 3, and the GRASP (with preprocessing). Even with the introduction of the preprocessing method, the behavior of the algorithms is similar to the one observed in the first round of experiments. CGHA+P is very fast, as all the solutions are calculated in less than two tenths of a second. ISHA+P improves the quality of the initial solution found by CGHA+P by 10.6% on average, reducing TST and WT by 9.4% and 21.9%, respectively. The GRASP is always able to produce the best solutions, improving by 14.1% the average solution value, and reducing TST and WT by 13.0% and 25.7%, respectively.

By comparing the results in Table 5.7 with those in Table 5.8, we can observe that, with respect to the algorithms under analysis, the introduction of the preprocessing method always allows to obtain better quality solutions within shorter computing times, which are basically halved.

Table 5.8: Computational results of CGHA, ISHA, and GRASP with preprocessing

Inst.	$\|J\|$	$\|M\|$	CGHA+P				ISHA+P				GRASP			
			z	TST	WT	sec	z	TST	WT	sec	z	TST	WT	sec
I01	50	10	5400.6	6275	1903.2	0.1	5084.6	5880	1903.2	3.1	4572.6	5265	1803.2	35.1
I02	76	10	9754.3	11600	2371.5	0.1	8886.1	10840	1070.6	14.7	8162.5	9910	1172.3	142.4
I03	78	10	8856.6	10060	4042.9	0.1	8008.6	9150	3442.9	11.5	7629.0	8775	3044.9	103.0
I04	82	10	9484.3	11170	2741.5	0.1	8339.6	9940	1938.2	15.2	7540.6	8940	1943.0	156.6
I05	83	10	10598.5	10060	12752.7	0.1	9600.0	8985	12059.8	12.6	9295.2	8730	11556.2	114.7
I06	85	10	10269.5	11855	3927.6	0.2	8825.0	10400	2525.1	10.5	8649.7	10230	2328.7	100.5
I07	85	10	11723.6	12820	7338.1	0.1	9771.8	10855	5439.2	10.7	9852.1	10880	5740.7	120.0
I08	86	10	10753.5	12775	2667.6	0.1	9917.0	11780	2465.1	15.4	9272.8	11025	2263.8	138.2
I09	87	10	8054.9	9435	2534.3	0.1	7843.1	9220	2335.3	16.1	7643.0	8870	2735.1	147.1
I10	89	10	9567.8	11485	1899.0	0.1	8134.6	9695	1892.9	16.4	7853.2	9445	1486.2	136.1
I11	89	10	12114.4	13630	6052.2	0.1	10800.7	12690	3243.4	31.4	10200.5	11990	3042.7	314.2
I12	89	10	11518.7	12085	9253.5	0.1	10643.2	11215	8356.0	16.9	9732.2	10300	7460.9	152.8
I13	91	10	12041.2	13635	5665.9	0.1	10885.3	12540	4266.4	14.6	10439.8	12010	4159.1	142.2
I14	93	10	10289.1	12675	745.6	0.1	9492.8	11780	344.2	13.3	8492.6	10530	343.1	138.2
I15	97	10	11151.8	12810	4518.8	0.1	9893.1	11760	2425.6	14.6	9502.0	11295	2330.1	135.6
I16	98	10	12624.0	13565	8859.8	0.1	10919.2	11760	7556.2	18.4	10727.2	11695	6856.2	171.2
I17	99	10	10916.4	13240	1621.8	0.1	9996.3	12165	1321.3	18.4	9466.7	11605	913.7	182.3
I18	102	10	9658.5	11180	3572.6	0.2	8874.1	10500	2370.3	16.4	8565.4	10165	2166.8	169.7
I19	102	10	10495.1	12380	2955.5	0.2	9536.1	11355	2260.3	20.6	9234.8	10980	2253.8	186.6
I20	105	10	12194.1	13930	5250.5	0.1	10753.4	12180	5047.2	24.7	10449.6	12150	3648.0	208.5
I21	107	10	10530.1	11500	6650.7	0.2	9698.6	10685	5753.2	22.8	9551.1	10500	5755.4	218.6
I22	109	10	12648.8	12225	14344.1	0.1	10721.6	11065	9347.8	22.6	10746.1	11045	9550.3	218.6
I23	110	10	11254.6	13615	1813.2	0.1	10104.1	12155	1900.5	42.4	9703.9	11705	1699.5	425.4
I24	113	10	12456.8	13895	6704.2	0.1	10976.4	12295	5702.0	26.5	10699.5	11900	5897.5	258.5
I25	116	10	11697.7	13995	2508.5	0.1	10188.3	12510	901.7	33.6	10442.9	12805	994.7	319.3
AVG	92.8	10.0	10642.2	12075.8	4907.8	0.1	9515.7	10936.0	3834.7	18.5	9137.0	10509.8	3645.8	177.4
GAP							-10.6%	-9.4%	-21.9%		-14.1%	-13.0%	-25.7%	

5.5.3 Comparison with Company Solutions

It is difficult to obtain a consistent comparison between solutions obtained by the proposed approaches and practical industrial solutions executed by the company, because some interruptions may occur during the daily activities. Thus, to be consistent with the evaluation of the solutions returned by the proposed approaches, given a company solution we re-calculate all the times required for the machine setup phases (following the scheme presented in Section 5.4.1) and all the times required for the printing phases (by simply dividing v_j by θ_i). In this way, start and completion times (and thus also the tardiness and the setup times) of all jobs of the practical industrial solutions have been re-calculated and provide a sound means of comparison.

Graphical representations of the results that we obtained are shown in Figures 5.8, 5.9, and 5.10, where we compare company and the GRASP solutions in terms of objective function values, TST, and WT, respectively. To comply with company privacy requirements, we do not report explicit numerical values. Still, it is evident that the GRASP provides better solutions and is always capable to outperform, by a considerable margin, the quality of the company solutions. This result is due to the fact that the GRASP algorithm is able to find solutions with a much shorter total setup time and almost equivalent total weighted tardiness.

5.5.4 Impact of the local search procedures

The next round of experiments, whose results are highlighted in Table 5.9, aims at studying the impact of the local search components (described in Section 5.4.4) on the initial solution computed by CGHA+P. In CGHA+P+LS1, CGHA+P+LS2, CGHA+P+LS3, and CGHA+P+LS4, the constructive phase is completed by means of CGHA+P and then the solution is brought to a local minimum thanks to LS1, LS2, LS3, and LS4, respectively. The results show that, in terms of solution quality, LS1 is able to produce on average the largest improvement. Indeed, the value of the objective function improves by 10.6% with LS1, by 4.8% with LS2, by 0.7% with

Figure 5.8: Comparative evaluation of the objective function value between company and the GRASP solutions

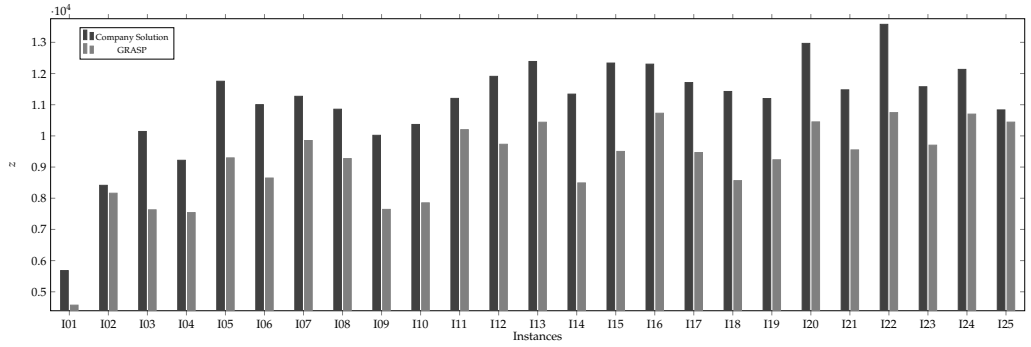


Figure 5.9: Comparative evaluation of the total setup time (TST) between company and the GRASP solution

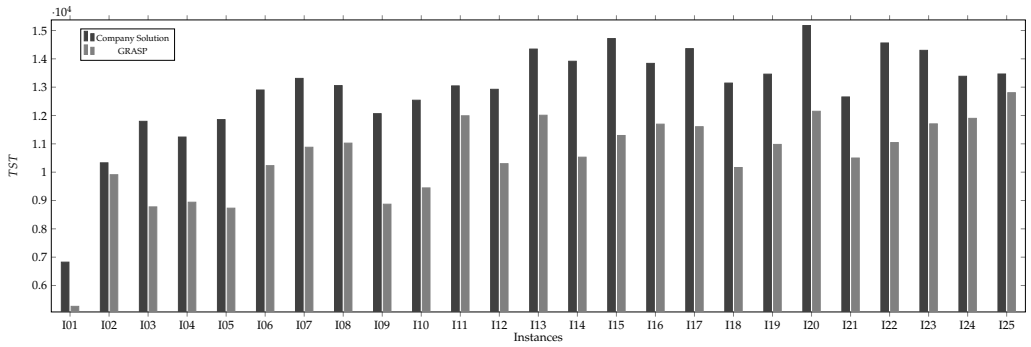
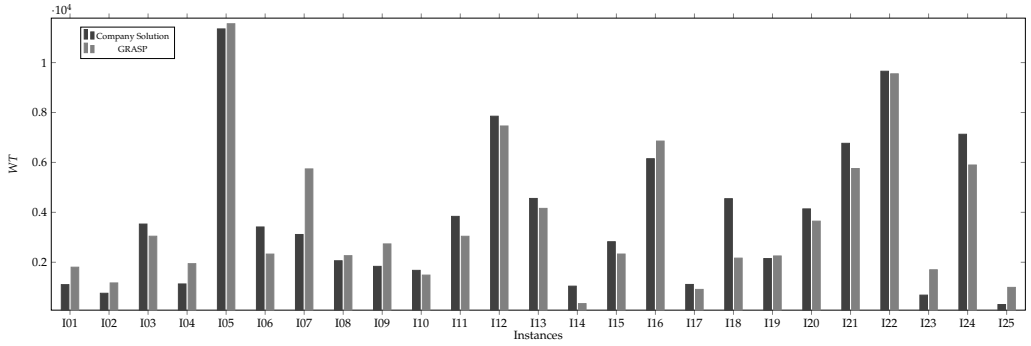


Figure 5.10: Comparative evaluation of the total weighted tardiness (WT) between company and the GRASP solution



LS3, and by 2.4% with LS4. This result highlights the strongly sequence-dependent nature of the problem. Indeed, by re-sorting each sequence of jobs by means of the intra-machine swap moves of LS1 and LS4 it is possible to obtain a large improvement over the initial solution. Even if LS3 brings just slight improvements of the initial solution, they are obtained in a very short time (always less than two tenths of a second), thus LS3 is able to provide a good trade-off between the computational

effort that it requires and its effectiveness. On the other side, LS2 is the most time-consuming local search, reflecting the fact that the size of the inter-machine insertion neighborhood is the largest one.

Table 5.9: Comparative evaluation of the different local search procedures

Inst.	$\|J\|$	$\|M\|$	CGHA+P		CGHA+P+LS1		CGHA+P+LS2		CGHA+P+LS3		CGHA+P+LS4	
			z	*sec*	z	*sec*	z	*sec*	z	*sec*	z	*sec*
I01	50	10	5400.6	0.1	5212.6	0.4	5392.6	1.7	5400.6	0.1	5336.6	1.3
I02	76	10	9754.3	0.1	9219.0	5.8	9729.7	8.0	9726.5	0.1	9622.2	0.9
I03	78	10	8856.6	0.1	8192.9	1.9	8684.2	6.6	8565.1	0.1	8692.6	1.1
I04	82	10	9484.3	0.1	8499.8	2.9	8569.8	10.3	9472.3	0.1	9340.3	3.1
I05	83	10	10598.5	0.1	9951.5	3.8	10190.3	6.5	10484.1	0.1	10598.5	1.4
I06	85	10	10269.5	0.2	9309.6	1.3	9993.4	8.4	10269.5	0.2	10157.3	2.6
I07	85	10	11723.6	0.1	10555.9	2.3	11439.8	6.8	11719.6	0.1	10943.8	2.3
I08	86	10	10753.5	0.1	10045.6	4.2	10124.9	9.8	10733.6	0.1	10601.5	1.4
I09	87	10	8054.9	0.1	7935.4	2.6	8010.9	11.3	7970.8	0.1	8034.9	1.3
I10	89	10	9567.8	0.1	9102.9	3.8	9551.9	10.6	9431.8	0.1	9411.8	1.4
I11	89	10	12114.4	0.1	10838.9	25.8	11913.1	8.0	12047.3	0.1	11835.7	2.7
I12	89	10	11518.7	0.1	10863.1	2.8	11350.8	11.5	11430.7	0.1	11366.7	1.7
I13	91	10	12041.2	0.1	11137.7	3.8	11968.8	7.7	11953.2	0.1	11965.2	2.5
I14	93	10	10289.1	0.1	9616.9	2.3	10172.7	11.5	10185.3	0.1	10044.7	1.8
I15	97	10	11151.8	0.1	9945.6	3.7	11135.6	8.6	11151.8	0.1	11023.7	2.0
I16	98	10	12624.0	0.1	11909.3	5.9	12619.6	9.1	12579.9	0.1	12139.7	2.1
I17	99	10	10916.4	0.1	10276.5	4.8	10851.6	10.1	10868.5	0.1	10482.9	3.6
I18	102	10	9658.5	0.2	9202.8	5.3	9621.8	8.3	9626.5	0.2	9222.5	3.2
I19	102	10	10495.1	0.2	9876.0	4.1	10443.1	13.5	10415.1	0.2	10483.1	2.1
I20	105	10	12194.1	0.1	11394.1	6.2	12058.2	16.7	12062.8	0.1	11710.3	3.0
I21	107	10	10530.1	0.2	9990.8	5.0	10377.2	13.8	10506.2	0.2	10314.0	2.7
I22	109	10	12648.8	0.1	10798.5	8.2	12516.3	10.7	12465.0	0.1	12601.1	2.4
I23	110	10	11254.6	0.1	10674.2	12.2	11209.4	17.1	11238.7	0.1	10866.4	3.6
I24	113	10	12456.8	0.1	11489.3	6.3	12187.9	14.6	12325.1	0.1	12064.8	5.0
I25	116	10	11697.7	0.1	10460.9	11.5	11140.3	17.7	11689.5	0.1	11417.5	6.8
AVG	92.8	10	10642.2	0.1	9860.0	5.5	10161.8	10.4	10572.8	0.1	10411.1	2.5
GAP					-10.6%		-4.8%		-0.7%		-2.4%	

Table 5.10 shows the results of the round of experiments that aims at evaluating the impact of the local search components on the GRASP. In GRASP-LS1, GRASP-LS2, GRASP-LS3, and GRASP-LS4, we do not invoke either LS1, LS2, LS3, or LS4 at steps 5-6 of Algorithm 2, respectively. The results are consistent with those in Table 5.9. Indeed, the quality of the solutions decreases on average by 7.2% with GRASP-LS1, by 1.0% with GRASP-LS2, by 0.6% with GRASP-LS3, and by 2.4% with GRASP-LS4. The results highlight once more the strongly sequence-dependent nature of the problem. Indeed, without invoking the intra-machine swap moves of LS1 and LS4 in the local search phase of the GRASP, we obtain a relative large GAP with the GRASP solution. GRASP-LS2 is the fastest procedure under analysis, reflecting the fact that the LS2 is the most time-consuming local search, as highlighted by the previous round of experiments. On the other hand, GRASP-LS3 is the most time-consuming procedure. This result is consistent with the fact that LS3 is the fastest among the proposed local search procedures.

5.5.5 The Role of β in the Interplay Between WT and TST

In this section, we provide some insights for managing production operations through a detailed analysis of the interplay between WT and TST by varying parameter β. Specifically, we analyse the impact of the two different objective function components in the PMPST by means of a Pareto front analysis. In Figure 5.11, a Pareto front is obtained by means of the weighted sum method (see, e.g., Ehrgott [170]). This method consists in solving different single objective problems obtained by a

Table 5.10: Comparative evaluation of the impact of the different local search procedures on the GRASP

Inst.	$\|J\|$	$\|M\|$	GRASP		GRASP-LS1		GRASP-LS2		GRASP-LS3		GRASP-LS4	
			z	sec	z	sec	z	sec	z	sec	z	sec
I01	50	10	4572.6	35.1	5008.6	22.8	4884.6	14.5	4548.6	26.5	4824.6	19.4
I02	76	10	8162.5	142.4	8833.1	86.2	8175.1	66.4	8514.1	135.7	8546.6	130.8
I03	78	10	7629.0	103.0	8067.7	74.7	7685.2	32.7	7724.1	92.4	7536.6	90.2
I04	82	10	7540.6	156.6	8607.9	128.0	7551.9	52.7	7820.1	143.6	8036.1	126.1
I05	83	10	9295.2	114.7	9622.5	68.3	9755.2	54.8	9499.2	109.6	9315.3	99.7
I06	85	10	8649.7	100.5	9005.2	80.3	8521.5	35.3	8635.6	92.0	8563.5	74.6
I07	85	10	9852.1	120.0	11004.1	93.9	9896.0	44.8	9768.1	109.8	10368.2	92.6
I08	86	10	9272.8	138.2	9920.9	96.5	9265.4	52.9	9140.9	131.6	9512.8	126.0
I09	87	10	7643.0	147.1	7687.2	114.4	7635.1	36.3	7639.5	134.2	7667.0	144.3
I10	89	10	7853.2	136.1	8418.7	110.1	7821.7	40.0	8193.9	130.0	8349.8	119.7
I11	89	10	10200.5	314.2	11154.2	97.6	10346.4	223.2	10612.7	286.0	10832.5	285.3
I12	89	10	9732.2	152.8	10295.1	113.7	9987.4	42.7	9571.5	142.4	9895.0	135.9
I13	91	10	10439.8	142.2	10844.0	92.1	10661.1	62.6	10167.7	131.5	10432.8	119.1
I15	93	10	8492.6	138.2	9165.0	103.5	8662.0	37.0	8643.9	125.4	8684.6	112.7
I16	97	10	9502.0	135.6	10719.6	90.2	9121.2	58.1	9398.6	128.7	9326.5	109.3
I17	98	10	10727.2	171.2	11634.7	105.4	10524.2	109.8	10695.6	163.1	11384.2	145.3
I18	99	10	9466.7	182.3	10131.0	122.2	9471.6	78.4	9480.1	173.9	9558.6	138.7
I19	102	10	8565.4	169.7	9165.8	101.7	8947.4	66.0	8745.7	148.6	9017.6	132.7
I20	102	10	9234.8	186.6	9571.2	142.0	9452.0	57.5	9391.4	175.1	9247.1	159.0
I21	105	10	10449.6	208.5	11502.3	162.1	10849.6	68.0	10941.1	205.2	10845.4	185.8
I22	107	10	9551.1	218.6	9605.4	156.5	9655.3	74.1	9551.1	215.4	9401.5	185.4
I23	109	10	10746.1	218.6	12076.7	118.4	10618.6	117.8	10658.1	201.0	10818.9	186.3
I24	110	10	9703.9	425.4	10549.1	202.2	9846.0	211.7	9645.2	325.8	10232.7	361.6
I25	113	10	10699.5	258.5	11420.0	179.0	10640.6	112.9	10748.2	247.5	11152.0	204.7
I26	116	10	10442.9	319.3	10860.2	204.1	10747.7	158.8	10119.1	299.1	10375.2	296.6
AVG	92.8	10.0	9137.0	177.4	9794.8	114.6	9228.9	76.4	9194.2	163.0	9357.0	151.3
GAP					7.2%		1.0%		0.6%		2.4%	

linear scalarization of the multiple objective function components. In the PMPST, the Pareto optimal solutions are sought one by one by changing the value of parameter β. Table 5.11 reports the average values of the Pareto solutions obtained over the 25 instances described in Section 5.5.1 by varying β in the set $\{0, 0.1, 0.2, \ldots, 1\}$. For each value taken by β, columns z, TST, and WT give the average value of the objective function, of the total setup time, and of the total weighted tardiness, respectively. These values are used for drawing the Pareto front in Figure 5.11.

As expected, the average of the total setup time increases with the increasing of β, while the average of the total weighted tardiness decreases. The value 0.2, chosen by the company as the value to be used in the real-world application, represents indeed a good balance. It guarantees to obtain schedules with a very short total setup time and an acceptable weighted tardiness. Other good values, which would however lead to higher TST values, are those between 0.3 and 0.9. Notably, the solutions of the two mono-objective problems should be disregarded because they are very unbalanced: when optimizing only TST (i.e., $\beta = 0$), we obtain $WT = 4552.9$, which is 45% away from the best value that we obtain by optimizing only WT (3106.1); on the other side, when optimizing only WT ($\beta = 1$), we obtain $TST = 13700.4$, which is 32% away from the best WT value (10407.2). With $\beta = 0.2$, instead, the solution obtained differs only by 20% from the best WT value and by 1% from the best TST value.

β	z	TST	WT
0.0	10407.2	10407.2	4552.9
0.1	9814.6	10445.4	4137.1
0.2	9191.3	10555.6	3733.9
0.3	8513.9	10598.6	3649.5
0.4	7808.2	10726.4	3430.9
0.5	7058.5	10733.0	3384.0
0.6	6319.0	10884.0	3275.7
0.7	5525.0	10959.4	3196.0
0.8	4756.7	11094.4	3172.3
0.9	3943.3	11349.4	3120.4
1.0	3106.1	13700.4	3106.1

Table 5.11: Iterative results of Pareto optimal solutions by changing the parameter β

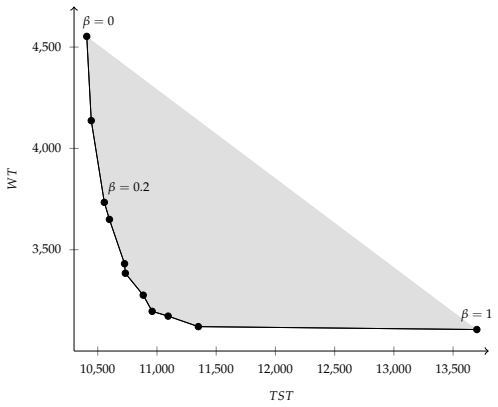


Figure 5.11: Pareto optimal fron

5.5.6 The Impact of the Random Component of the GRASP on the final solution

In this section, we analyze the impact of the random component of the GRASP construction phase on the obtained solutions. For this purpose, we ran the GRASP 10 times. Table 5.12 shows the results of this round of experiments. More specifically, in the column "Best solution found" (resp. "Worst solution found") the values of the best solution (resp. worst solution) are reported, whereas in the column "Average solution values" we report the average values over the 10 solutions. We can observe that the values do not differ much from each other. The percentage gap between the objective function values of the best and worst solutions is on average 2.5%, while the percentage gap between the best and the average solution values is just 0.6%. Furthermore, the average standard deviation of the solution values is 115.5, which is very low compared to the average value (9162.5). This low value confirms the limited impact of the random component of the GRASP and the robustness of the overall algorithm.

5.5.7 Computational Results on an Available Instance Set

Due to the commercial nature of this research, we cannot publicly share the real-world instances used in the previous sections. However, to allow and encourage future research on this interesting problem, we propose a new instance set composed of 25 realistic but randomly created instances. The set is available online at `https://github.com/regor-unimore/Parallel-Print-Machine-Problem-with-Setup-Times/tree/main/Instances` and has been generated by applying some randomization to the original 25 real-world instances. In the new set that we obtained in this way, the number of jobs spans from 50 to 130, and the number of machines varies in the set $\{8, 9, 10, 11, 12\}$. Table 5.13 shows the results obtained by means of our best algorithm, namely the GRASP (with preprocessing). All the solutions that we obtained have been also made publicly available at the web site mentioned above.

Table 5.12: The Impact of the Random Component of the GRASP on the final solution

Inst.	$\|J\|$	$\|M\|$	Best found Solution			Worst found Solution			Average Values			
			z	TST	WT	z	TST	WT	z	TST	WT	sec
I01	50	10	4572.6	5265.0	1803.2	4820.6	5575.0	1803.2	4654.5	5377.5	1823.2	40.4
I02	76	10	8382.2	10185.0	1170.8	8562.4	10410.0	1172.1	8392.5	10231.0	1151.8	111.4
I03	78	10	7455.9	8460.0	3439.7	7756.3	8860.0	3341.7	7614.2	8657.0	3522.0	134.7
I04	82	10	7723.7	9145.0	2038.3	7531.8	8930.0	1939.2	7503.4	8908.0	1949.0	155.9
I05	83	10	9462.4	8865.0	11852.0	9531.0	8900.0	12054.9	9473.9	8829.5	12074.8	168.6
I06	85	10	8477.1	9940.0	2625.6	8617.3	10085.0	2746.6	8530.7	10026.5	2600.6	126.1
I07	85	10	9763.7	10870.0	5338.6	10180.2	11415.0	5241.1	9933.1	11038.5	5620.1	160.6
I08	86	10	9292.8	11125.0	1964.0	9616.6	11430.0	2363.0	9275.1	11105.0	2094.5	150.3
I09	87	10	7535.1	8910.0	2035.5	7639.5	9025.0	2097.3	7631.0	8931.0	2447.0	192.9
I10	89	10	8714.2	10570.0	1290.9	8193.9	9920.0	1289.4	8193.9	9920.0	1289.4	188.4
I11	89	10	10256.9	11935.0	3544.4	10300.6	12115.0	3043.1	10298.1	12112.0	3043.1	316.1
I12	89	10	9771.5	10300.0	7657.5	9843.2	10315.0	7955.9	9749.1	10257.0	7906.2	129.0
I13	91	10	10335.7	11930.0	3958.6	10716.8	12380.0	4064.1	10393.0	11984.5	4071.3	138.4
I14	93	10	8397.3	10410.0	346.7	8701.8	10765.0	448.9	8484.1	10534.0	414.1	318.0
I15	97	10	8976.5	10515.0	2822.6	9418.3	11065.0	2831.7	9239.3	10853.5	2877.9	151.7
I16	98	10	10572.0	11325.0	7559.8	10883.4	11715.0	7557.2	10627.3	11506.5	7188.0	165.3
I17	99	10	9267.0	11330.0	1015.1	9975.3	12165.0	1216.6	9585.0	11747.5	1116.1	161.4
I18	102	10	8657.3	10255.0	2266.7	9129.6	10845.0	2267.8	8847.3	10494.0	2337.4	227.5
I19	102	10	9155.0	10930.0	2054.8	9351.5	11120.0	2277.6	9214.8	10957.0	2327.6	430.7
I20	105	10	10781.7	12490.0	3948.3	11093.7	12990.0	3508.7	10751.0	12513.5	3858.0	213.2
I21	107	10	9346.1	10295.0	5550.4	9546.2	10595.0	5351.1	9364.7	10301.0	5682.6	112.5
I22	109	10	10646.3	10995.0	9251.6	10778.2	11125.0	9391.0	10707.1	11015.0	9502.1	325.7
I23	110	10	9481.1	11300.0	2205.6	9940.2	11925.0	2405.7	9700.9	11638.0	2074.4	301.0
I24	113	10	10612.5	11665.0	6402.4	10903.6	12175.0	6499.7	10742.6	11966.5	5899.7	113.0
I25	116	10	10054.3	12295.0	1091.7	10274.5	12555.0	1398.6	10155.2	12407.5	1214.4	273.4
AVG	92.8	10.0	9107.6	10452.2	3729.4	9332.3	10736.0	3770.6	9162.5	10532.5	3763.4	192.3
GAP						2.5%	2.7%	1.1%	0.6%	0.8%	0.9%	

Table 5.13: Computational results of the GRASP on the publicly-available instance set

Inst.	$\|J\|$	$\|M\|$	GRASP			
			z	TST	WT	sec
BI01	50	8	4025.8	4920.0	404.0	31.5
BI02	75	8	6335.9	7885.0	103.5	126.4
BI03	80	8	8197.6	10190.0	105.2	139.5
BI04	80	8	6380.8	7530.0	938.2	154.8
BI05	80	8	6849.3	8195.0	1403.5	119.5
BI06	80	9	6545.1	7625.0	2012.6	116.8
BI07	85	9	8002.9	9415.0	1823.4	121.8
BI08	85	9	7601.1	8615.0	2454.5	131.7
BI09	90	9	6997.9	8615.0	408.3	205.7
BI10	95	9	8323.6	10070.0	1208.8	164.2
BI11	90	10	6671.6	8230.0	402.0	96.8
BI12	90	10	7926.5	9195.0	1094.4	136.7
BI13	95	10	8915.3	10110.0	3529.7	166.8
BI14	100	10	9791.3	11555.0	2129.6	174.6
BI15	100	10	8371.1	10065.0	1309.6	174.7
BI16	100	11	7379.6	9140.0	113.0	158.3
BI17	100	11	7435.3	9105.0	414.5	179.1
BI18	105	11	9209.0	11415.0	19.2	123.6
BI19	105	11	8789.6	10730.0	808.8	209.7
BI20	110	11	9941.9	11795.0	1370.6	181.6
BI21	115	12	9569.8	11480.0	843.9	232.1
BI22	120	12	10071.5	12395.0	324.3	247.7
BI23	120	12	9823.1	11575.0	2512.7	163.4
BI24	120	12	10100.3	11295.0	4543.4	213.9
BI25	130	12	11268.1	13605.0	858.3	295.9
AVG	96	10	8181.0	9790.0	1245.4	162.7

5.6 Conclusions

In this chapter, we tackled an interesting real-world industrial problem arising in a food packaging company located in the city of Reggio Emilia (Italy). In particular, we addressed a scheduling problem that consists in assigning printing jobs to a heterogeneous set of parallel flexographic printer machines, with the aim of minimizing a weighted sum of total weighted tardiness and total setup time.

To face the problem, we proposed a GRASP algorithm that makes use of multiple local search procedures and of a preprocessing method aimed at producing smaller and more tractable instances. Computational results on real-world instances showed that the solution quality obtained by the GRASP is far superior than the one

obtained by a constructive heuristic, even when the latter is further improved by the use of local search procedures. This improved performance comes at the cost of an increase in the total computing time. However, this increase is considerably reduced by the introduction of the preprocessing method and, in practical terms, is not an issue. Indeed, since the planned activities of the company cover about one week, the computational times required by the GRASP (three minutes on average) are still largely acceptable for practical use.

A comparative evaluation with company solutions showed that the GRASP is able to produce significantly better quality solutions than the ones proposed by the decision makers of the company. Indeed, it finds solutions with much shorter total setup time and almost equivalent total weighted tardiness. These good results were confirmed also in practice, as the company is currently using the algorithm for their weekly production scheduling.

Several relevant extensions to this work may be taken into account as future research directions. The FSS metaheuristic has been recently applied with success to different combinatorial optimization problems (see, e.g., Jovanovic, Sanfilippo, and Voß [286]), including a scheduling problem on unrelated parallel machines with sequence-dependent setup times (see Jovanovic and Voß [287]). It would be interesting to extend the proposed GRASP by combining it with the concept adopted in the FSS metaheuristic.

In addition, an ILP formulation or a constraint programming model for solving exactly the PMPST could be introduced. A constraint programming approach has been recently applied for solving the multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times (see Yunusoglu and Topaloglu Yildiz [502]). Due to the size of real-world instances, we point out that an ILP approach could be not suitable for solving the PMPST. Indeed, Iori et al. [278] proposed an ILP formulation for solving a simplified version of the PMPST on a single machine and, even for very small instances (10 jobs), the proposed method is not able to find the optimal solution.

A rolling horizon framework could also be considered to allow the decision makers to use the algorithm even on a daily basis, thus looking for further improvements. Furthermore, since in the real-world application processing and setup times are subject to significant uncertainties (which are not captured by the heuristic evaluation procedure we proposed in Section 5.4.1), another interesting future research project consists in estimating these times through the application of machine learning methods. Finally, as a practical extension of our work, it would be interesting to make the GRASP directly interact with the incoming product warehouse, through the introduction of the release dates of jobs.

Acknowledgments

We acknowledge financial support from Istituto Stampa s.r.l. (Italy). The work originates from the daily activity of Istituto Stampa s.r.l., a company whose headquarters is located in Reggio Emilia (Italy). The company operates in the field of packaging industry by producing and printing packaging materials for food products since 1933.

Data Availability Statement

The data that support the findings of this study are openly available in *Instances* at https://github.com/regor-unimore/Parallel-Print-Machine-Problem-with-Setup-Times.

.

Chapter 6

Setup time prediction using machine learning algorithms: A real-world case study*

In this chapter, we explore the use of machine learning regression algorithms for setup time prediction and we apply them to a real-world scheduling application arising in the color printing industry. To enhance the quality of setup time evaluations and narrow the gap between scheduling theory and practice, we aim at exploiting a data-driven approach based on machine learning algorithms. Using a real-world industrial dataset, we train three different machine learning models: linear regression, random forests, and gradient boosting machines. The experimental results demonstrate that the gradient boosting machine approach obtains the best performance overall, immediately followed by random forests. For both such models, the mean squared error on the predicted setup times is less than half of that of the heuristic evaluation method proposed in Chapter 5.

6.1 Introduction

According to Allahverdi and Soroush [21], *setup time* (ST) can be defined as the time required to prepare the necessary resource (e.g., machines, people) to perform a task (e.g., a job, an operation). Setup operations may include, for instance, switching tools, cleaning up, changing material, adjusting tools, etc. As all these operations are often strongly time-consuming and may take a considerable part of the entire production time, the reduction of STs plays a crucial role in scheduling. The deep connection between scheduling activities and STs was analyzed in detail by Abd-Alsabour [1], who provided a comprehensive survey on scheduling problems involving STs.

Although, in many real-world manufacturing environments, STs are influenced by various and random factors (e.g., crew experience, breakdowns of a tool or a machine, lack of personnel, complex and non-fixed procedures, etc.), in the extant literature, the vast majority of scheduling studies consider STs as a given input taking a deterministic value, calculated by means of simplistic average-based methods that are not capable to catch the complexity of reality. As already highlighted by Kim and Bobrowski [304], assuming stochastic STs as fixed and constant values may lead to develop inefficient schedules. This is also confirmed by practitioners, who

*The results of this chapter were accepted to international conference: M. Iori, M. Lippi, A. Locatelli, and M. Locatelli. "Setup time prediction using machine learning algorithms: A real-world case study". In: *24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF 2023*.

often view the lack of uncertainty and dynamic elements in the modeling of scheduling processes as the major source of gap between scheduling theory and practice (see Sabuncuoglu and Goren [428]). Only a few of studies, recently reviewed by Allahverdi [19], took into account the issues related to the STs uncertainty. These studies mainly focused on robust optimization approaches (modeling STs as a probability distribution, or a fuzzy number, or a random variable within some interval), but did not provide any methods to estimate these STs.

In this work, we seek to fill this gap of the literature by provide models that can be used to provide good estimations of the STs starting from a set of empirical data. In detail, we use *machine learning* (ML) regression algorithms to predict the STs, and we apply them to a real-world scheduling application arising in the color printing industry, where a finite set of jobs must be sequentially performed by a heterogeneous set of parallel flexographic printer machines. Specifically, we deal with *uncertain machine-dependent and job sequence-dependent setup times* (UMJSSTs) with an additional issue: the UMJSST between two jobs not only depends on the two jobs and the involved machine, but also on all jobs previously scheduled on that specific machine, owing to tool configurations (see, e.g., Soares and Carvalho [457]). Indeed, jobs have different tooling requirements and it is often beneficial to leave a certain tool unused in a machine magazine only to use it again a few jobs later (see Iori, Locatelli, and Locatelli [275, 276] and Iori et al. [278]). We use the acronym UMJESTs to denote UMJSSTs with such job entire-sequence-dependent characteristic.

In the addressed real-world industrial application, UMJESTs are strongly time-consuming (around 65% of the total production time) and subject to significant uncertainties, because depending on many factors: characteristics of the specific machine, status of the machine (which directly depends on the jobs already processed), characteristics of the current job, operators' experience, etc. It turns out that the problem of predicting such times is not trivial and may require a large computational effort (see Iori, Locatelli, and Locatelli [275]). Iori, Locatelli, and Locatelli [276] introduced a *heuristic evaluation method* (HEM), which expresses UMJESTs in terms of some specific and pre-fixed features (i.e., characteristics of jobs and machines). This analytic approach suffers from intrinsic limitations and cannot account for the whole complexity of the problem (see [276]).

To improve the accuracy of UMJESTs prediction, we aim at exploiting a data-driven approach based on ML algorithms to be exploratory both in selecting the significant features and expressing the UMJESTs in terms of these features. Using a real-world industrial database, we train three different ML models: *linear regression* (LR), *random forest* (RF), and *gradient boosting machine* (GBM). For each model, we take into account a wide set of features and several possible inter-dependencies among them, in order to identify a parsimonious but comprehensive subset of these features. We compare the three models with HEM by Iori, Locatelli, and Locatelli [276] on a real-world industrial test set. The experimental results demonstrate that the GBM approach obtains the best performance overall, immediately followed by RF. For both models, the mean squared error on the predicted UMJESTs is less than half of that of HEM, proving their effectiveness in modeling the application. The results are of interest because the ML models can be easily adapted to deal with ST evaluation in many other scheduling problems.

6.2 Brief Literature Review

Almost the totality of scheduling studies that tacked into account issues related to the STs uncertainty mainly focused on providing robust optimization methods to best manage the shortcomings in the STs prediction (see González-Neira, Montoya-Torres, and Barrera [210]). In this context, the uncertainty of STs is generally represented by interval data or fuzzy description. Aydilek, Aydilek, and Allahverdi [34, 35], Allahverdi and Allahverdi [20], and Allahverdi [23] modeled sequence-independent STs by using interval data, while Yimer and Demirli [497] and Lu, Lin, and Ying [345] represented sequence-independent and sequence-dependent family STs, respectively, by fuzzy sets. To represent the shortcomings associated with estimation of sequence-dependent STs, Lu, Lin, and Ying [345] and Naderi-Beni et al. [382] used fuzzy description, while Behnamian and Fatemi Ghomi [44] used interval description.

As sequence-dependent STs are subject to many complexities, the problem of predicting such times is not trivial and, as already highlighted in Section 6.1, standard analytical approaches cannot account for the whole complexity of the problem. Surprisingly, advanced analytic methods received limited attention in this context. To the best of our knowledge, the only study that dealt with STs estimation complexities is the one by Cheng et al. [107], who proposed a random-forest-based metaheuristic to minimize the makespan in an unrelated parallel machines scheduling problem with UMJSSTs.

In contrast to the traditional sequence-dependent STs, the UMJESTs between two jobs depends also on all jobs previously scheduled on the involved machine, rather than only on the two jobs. This characteristic of STs typically arises in the context of flexible manufacturing systems (see, e.g., Chryssolouris [115]), where a flexible manufacturing machine (see, e.g., [124]) is capable to handle different types of operations performed by the available tools installed in its limited capacity magazine. As each job requires a subset of tools and the magazine cannot hold all tools at once, some tool switches may be necessary when performing two jobs in succession (see, e.g., Laporte, Salazar-González, and Semet [329]). If a tool switch is required, then a positive setup time is incurred. Thus, it is often beneficial to leave a certain tool unused in the magazine to use it again a few jobs later. For instance, a tool in the magazine not required by the job currently being processed may be kept in the magazine if required to process the next jobs (see, e.g., Soares and Carvalho [457]). This feature of UMJESTs adds further complexities and uncertainties to STs and makes the problem of predicting UMJESTs even more challenging to solve in practice. To be able to solve practical industrial instances, Iori, Locatelli, and Locatelli [276] introduced HEM, a quick evaluation method to predict UMJESTs based on a specific policy for determining the required setup operations. This simplistic approach is not capable to incorporate the whole complexity of reality and, in addition, does not take into account any issues related to the STs uncertainty. In this chapter, to improve the accuracy of UMJESTs prediction, we employ three different ML techniques: LR, RF, and GBM. There is a vast literature on these three ML algorithms and we refer the interested reader to the comprehensive books by Montgomery, Peck, and Vining [377], Breiman et al. [65], and to the article by Natekin and Knoll [383], respectively.

6.3 Methodology

Firstly, we formulate the task of forecasting UMJESTs as a regression problem, where the aim is to predict a UMJEST, represented by a real positive number $y \in \mathbb{R}$, as a function f of a n-dimensional set A of attributes/features. We recall that the UMJEST between two jobs depends on many factors: the involved machine, the two jobs, and on all jobs previously scheduled on that specific machine, thus the attributes in A must express all these factors. On the other hand, a too wide set of attributes may lead to overfitting issues and uncertainty predictions, reducing the effectiveness of the ML models (see Kuhn and Johnson [319]). Thus, we take into account a wide set of features and several possible inter-dependencies among them, and then, using a feature selection method based on feature importance scores generated by RF, we identify a parsimonious but comprehensive subset of these features.

As already stated in Section 6.1, we use supervised ML algorithms to build mathematical models for predicting UMJESTs. In this phase, a finite dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^{m}$ of pairs, each containing both the n-dimensional input vector of features $x_i = (x_i^1, \ldots, x_i^n) \in A$ and the desired output $y_i \in \mathbb{R}$, is provided. The learning task is to find a function $\widetilde{f}$ belonging to a certain family F of machine learning models (in our case LR, RF, and GBM) that, for each input x_i, has a predicted output $\widetilde{y}_i = \widetilde{f}(x_i)$ as close as possible to the provided target y_i.

As first basic approach, we compute f as a linear combination of input features $f(x) = \beta_n x_n + \ldots, \beta_1 x_1 + \beta_0$ with the vector of parameters $(\beta_0, \ldots, \beta_n)$ that we wish to learn. LR is a widely employed technique that achieves good results when a linear function can represent a reasonable approximation of the relation holding between input and output variables (see Lewis-Beck and Lewis-Beck [336]).

In our problem there is a complex non-linear relation between target and heterogeneous variables (nominal, ordinal, and scalar), thus we also consider two non-linear models largely used in ML applications, namely RF and GBM, and able to manage heterogeneous data. More specifically, a RF is an ensemble learning method made up of a number of decision trees, called estimators, which each produce their own predictions. The RF model combines the predictions of the estimators to produce a more accurate prediction (see Breiman [64]). Also a GBM is an ensemble of weak prediction models (in our case decision trees) combined into a single strong learner in an iterative fashion. As each weak learner is added, a new model is fitted to provide a more accurate estimate of the response variable. The new weak learners are maximally correlated with the negative gradient of the loss function, associated with the whole ensemble (see Hastie, Tibshirani, and Friedman [234]).

The aim of this work is to provide a general-purpose ML framework for predicting sequence-dependent STs. This is reflected in the choice of using general-purpose ML models, without resorting to more sophisticated and case-specific approaches difficult to replicate and adapt to other cases. More specifically, the choice fell on RM and GBM because, among the classic ML algorithms, their characteristics fit well with those required by the problem. Indeed, RM and GBM perform well with a high number of heterogeneous features and can be applied also when the data contain missing values, typical characteristics of a real-world industrial database. An additional important feature of RF and GBM is the evaluation of the relevance of a variable by means of measures, which provides a greater readability of the models and allows the identification of possible redundancy among features.

Finally, we compare the three obtained models, with the aim of finding the one that performs the best on our real-world dataset. The final unbiased generalization error of the selected model is computed on a provided test set $\mathcal{D}_{test}$.

6.4 Case and Data Description

We apply the general-purpose ML framework introduced in Section 6.3 to a real-world scheduling application arising in the color printing industry. The problem consists in evaluating UMJESTs between jobs that are processed by a set of heterogeneous flexographic printer machines along a scheduling period from January 2019 to October 2022.

6.4.1 Case Description

Flexography is a printing technology, generally used in packaging industry, that allows for printing on almost any type of material such as plastic, paper, cellophane, and aluminum foil [305]. A flexographic printer machine is a flexible manufacturing machine [124] able to handle different types of operations (i.e., coloring, embossing, and perforating) which are performed by the available tools installed in its magazine. The number of tools that a machine can equip is limited by its magazine capacity. For instance, the number of different ink cartridges that a machine can simultaneously load is bounded by its number of color groups.

The setup phase consists in preparing the machine for printing a new job and may require performing some tool switching operations. Indeed, the limited magazine capacity of the machines and the demand to process various types of jobs often induce to replace the currently installed tools with other tools required to process the next jobs. Since the variety of colors required by the jobs is enormous, changing the ink cartridges in the color groups of a machine is definitely the most frequent switching operation performed between two jobs.

This operation requires a washing process to remove the previous color residue from the inner surface of the color group, because such residual could affect the quality of the next job. The process to wash a color group takes a long time and depends directly on the specific colors involved. For instance, if the process requires to switch a black ink cartridge with a white one, then a complete cleaning of the corresponding color group is necessary to preserve the white purity, with the effect of significantly raising the ST. If, instead, the process requires to change a color group from a pale green ink to a dark green ink, then the setup takes less time because the dark green ink is not easily altered by the residuals of the pale green ink and therefore the relative color group needs just a quick washing. Moreover, if a printer machine equips an automatic washing system, the color groups can be washed in parallel, otherwise they have to be washed one by one. Hence, the time required for the setup phase is machine-dependent, job-dependent, and job-sequence-dependent. Indeed, it depends on the washing system of the machine, on the tools (i.e., ink cartridges, embossing rollers, and perforating rolls) available in the magazine, and on the tools required by the next job.

6.4.2 Data Description

The dataset $\mathcal{D}$ is provided by the company as an extraction of orders from the database of the Company Management System. It is composed of 16 595 samples collected between January 2019 and October 2022. In this format, each sample, corresponding to a processed job, has 74 attributes (e.g., job *id*, machine *id*, tools required by the job, list of colours required by the job, *id* of the job processed right before by

the same machine, etc.), many of which are removed because irrelevant for the prediction problem (e.g., order quantity, due date, release date, etc.). Table 6.1 reports the set of 26 attributes that may impact the UMJESTs.

Table 6.1: Features of dataset $\mathcal{D}$

Feature	Description	Type
F_1	Identification code of the job	Int
F_2	Identification code of the job processed right before	Int
F_3	Identification code of the set of required colours	Int
F_4	Identification code of the type of printed material	Int
F_5	The number of required colors	Int
F_6	1^{st} colour required	String
⋮	⋮	⋮
F_{15}	10^{th} colour required	String
F_{16}	Print mode, i.e, mirror or normal print	String
F_{17}	If perforating roller is required	Boolean
F_{18}	Identification code of the required special varnish	Int
F_{19}	Identification code of the required embossing roller	Int
F_{20}	Identification code of the required corona treatment	Int
F_{21}	If job requires compostable inks	Boolean
F_{22}	Identification code of the machine used to process the job	Int
F_{23}	The starting date of setup phase	Date
F_{24}	The starting time of setup phase	Time
F_{25}	The end date of setup phase	Date
F_{26}	The end time of setup phase	Time

6.4.3 Pre-processing and Cleaning Phase

In this section, we outline the pre-processing and manipulation procedures applied to the dataset $\mathcal{D}$ to enhance its quality, structure, and information content, with the aim of improving the performance of machine learning models. The features of the refined dataset are listed in Table 6.2.

As UMJESTs depend on many factors, the features reported in Table 6.1 are not capable to provide a complete representation of the information necessary for an accurate prediction of the UMJESTs. Thus, as a first step, we enriched the dataset by introducing additional features.

The results of a preliminary statistical analysis showed that UMJESTs are influenced not only by the specific machine used to process a job but also by some specific machine characteristics, i.e., the number of color groups on the machine and whether the machine has an automatic washing system. More precisely, when the machine has an automatic washing system, the color groups can be washed simultaneously, otherwise, they must be washed sequentially. Thus, feature F_{22} (see Table 6.1) was divided into three separate features, i.e., f_1, f_2, and f_3 (see Table 6.2), reporting these additional information.

Because of the sequence-dependent nature of the UMJESTs, we replaced the feature F_2 reporting the identification code of the job processed right before with new features f_{60}-f_{116} (see Table 6.2) describing all its characteristics.

After statistical analysis, we identified four tools for which the involving switching operations are most time-consuming, namely, white cartridge, varnish cartridge, and embossing roller. Because of the entire sequence-dependent nature of the UMJESTs, for each of these tools, we added an additional feature (future f_{117}-f_{119}, respectively, see Table 6.2), which indicates whether a switching operation involving such a tool is required or whether that tool is already held in the magazine, considering also all the jobs previously processed by the involved machine. On the other hand,

Table 6.2: Features of the dataset $\mathcal{D}$ after the pre-processing and manipulation phase.

Feature	Description	Type
f_1	Identification code of the machine used to process the job	Int
f_2	Whether the machine has an automatic washing system	Boolean
f_3	The number of color groups of the machine	Int
f_4	The number of colors required by the job	Int
f_5	Identification code of the required embossing roller	Int
f_6	Whether perforating roller is required	Boolean
f_7	Identification code of the required special varnish	Int
f_8	Whether job requires compostable inks	Boolean
f_9	Identification code of the required corona treatment	Int
f_{10}	Print mode, i.e, mirror or normal print	String
f_{11}	Identification code of the type of printed material	Int
f_{12}	Identification code for the whole set of required colours.	Int
f_{13}	Whether colour red is required.	Boolean
⋮	⋮	⋮
f_{59}	Whether colour yellow is required.	Boolean
f_{60}	Print mode of previous job (i.e, mirror or normal)	String
f_{61}	Identification code of the type of printed material of previous job	Int
f_{62}	Identification code of the set of required colours of previous job	Int
f_{63}	Whether colour red is required by previous job	Boolean
⋮	⋮	⋮
f_{116}	Whether colour yellow is required by previous job	Boolean
f_{117}	Whether a switch involving white cartridge is necessary.	Boolean
f_{118}	Whether a switch involving varnish cartridge is necessary.	Boolean
f_{119}	Whether a switch involving embossing roller is necessary.	Boolean
f_{120}	Whether f_{12} is equal to f_{62}.	Boolean
f_{121}	UMJEST (in minutes) between current and previous jobs	Int

future f_{120} indicates if a the current job and the job processed right before requiring exactly the same colours.

The transformation of categorical features F_5-F_{16} into boolean features f_{13}-f_{59} was implemented to simplify the representation of the data and improve its suitability for use by machine learning algorithms. Indeed, this conversion process not only simplifies the representation of the data, but also enhances the performance of the algorithms by reducing the dimensionality of the dataset.

For each sample, we only know the starting and ending time of the setup phase, thus UMJEST was calculated as the difference between these times. As we do not know the exact working timetable of the machines (indeed, machines may work overtime), we removed all samples for which the setup phase takes place on two different days. Indeed, in this case, we cannot accurately calculate UMJEST. After this cleaning procedure, around 40% of data were removed. We detected and removed outliers by using a boxplot-based method Schwertman and Silva [443] and all corrupted data. Finally, we replaced features F_{23} to F_{26} with feature f_{121} reporting the value of UMJESTs measured in minutes.

6.4.4 Feature Selection

We used the feature importance scores generated by RF to identify the most important features of the dataset. More precisely, feature importance is calculated based on the average decrease in impurity over all trees in the forest. The higher the importance score of a feature, the more it was used to split the data across decision trees. We use a threshold parameter $\alpha \in [0, 1]$, to select only the features with importance scores above α. Thus, we train the models using only the features with importance scores above α.

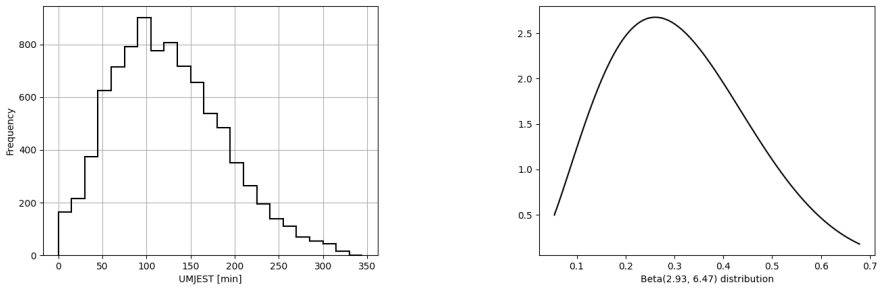


Figure 6.1: The distribution of UMJESTs from $\mathcal{D}$ and the resulting probability distribution.

6.5 Experiments

This section illustrates the computational experiments conducted to assess the effectiveness of the proposed models and compare them with EEM. Data preparation, pre-processing, and statistical analysis (see Section 6.4.3) were performed in Python using `pandas` and `numpy` libraries. The three models introduced in Section 6.3 were developed in Python by using the open-source `H2O` library [7]. The tests were executed on a 2.3GHz Intel Xeon Gold 6252N with 16 GB of memory.

6.5.1 Experimental Setup

After the pre-processing and cleaning phase described in Section 6.4.3), we obtained a dataset $\mathcal{D}$ composed of 9005 samples. Figure 6.1 shows, on the left, the distribution of UMJESTs observed in $\mathcal{D}$ and, on the right, the probability distribution that best fits the data, namely, beta distribution (found using Distfit library) with parameters α and β equal to 2.93 and 6.47, respectively.

The models were fit on $\mathcal{D}_{train}$ composed of 7890 samples and collected between January 2019 and April 2022. As the number of entries in the available real-world industrial dataset is limited, to avoid reducing the number of samples by removing a held-out dataset and running into any bias problem, the evaluation of ML models was performed via 10-fold cross-validation on $\mathcal{D}_{train}$, a procedure normally recommended for small real-world datasets [418]. More precisely, we randomly partitioned data into 10 equally-sized parts, named folds, and we trained 10 different models on 9 parts, using for testing purposes the part that was not used for training. Finally, the models were tested on a new corpus, named $\mathcal{D}_{test}$, composed of 1937 samples, which was collected later on, between May 2022 and October 2022.

6.5.2 Parameter Tuning

According to `H2O` library, LR corresponds to the Gaussian family model in which the link function is the identity, and the density corresponds to a normal distribution. LR model was fit using the least squares approach.

We trained RF by means of the random grid search, provided by `H2O`, over a specific hyper-parameter space defined according to preliminary experimental trials. Specifically, the number of trees takes values in {20, 50, 80, 110, 140, 170, 200}, the sampling rate in {0.63, 0.70, 0.80, 1.00}, the maximum depth of each tree in {10, 15, 20,25, 30, 35}, the number of columns to randomly select at each level in {10, 20, 40, 60, 80}, the number of bins to be included in the histogram and then split at the best point in {8, 16, 32, 64, 128, 256, 512, 1024}.

To train GBM, we used AutoML, an automated hyper-parameter tuning method provided by `H2O`.

Feature importance scores generated by RF are reported in Figure 6.2. We trained LR, RF, and GBM with $\alpha \in \{0, 0.05, 0.10, 0.15, 0.20, 0.25\}$, using the first 120, 52, 47, 40, 26, and 18 features, respectively.

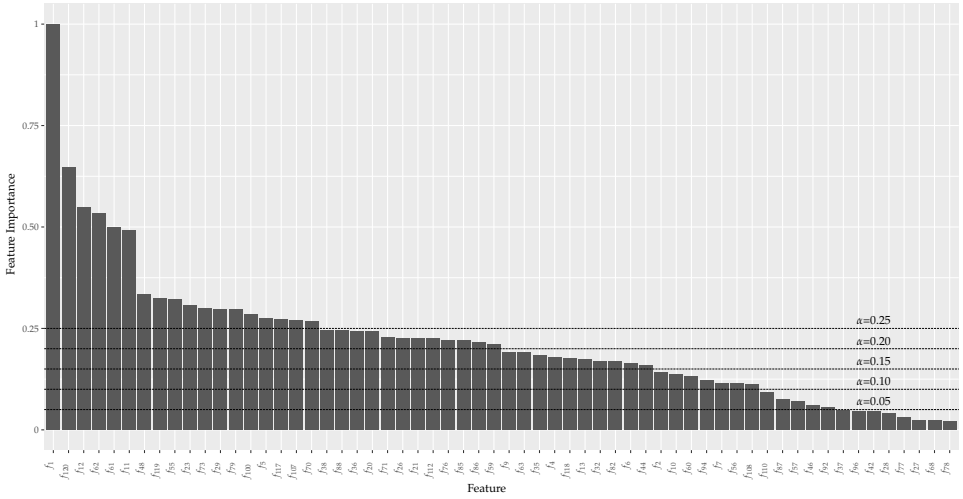


Figure 6.2: Feature importance scores of the first 60 features generated by RF.

6.5.3 Forecasting Results

Performance evaluations of the obtained models were done by measuring their quality using four statistical indicators: *mean square error* (MSE), *root mean square error* (RMSE),*mean absolute error* (MAE), and *root mean squared log error* (RMSLE). More precisely, MSE=$\frac{1}{m}\sum_{i=1}^{m}(y_i - \widetilde{y_i})^2$, RMSE=$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \widetilde{y_i})^2}$, MAE= $\frac{1}{m}\sum_{i=1}^{m}|y_i - \widetilde{y_i}|$, and RMSLE=$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(\log y_i - \log \widetilde{y_i})^2}$.

Table 6.3 shows the computational results of EEM, LR, RF, and GBM for different values of α. Thus, models were trained using different numbers of features, depending on the value of α, and their performances were evaluated in terms of MSE, RMSE, MAE, and RMSLE.

RF and GBM models consistently outperform the LR and EEM, with lower MSE, RMSE, and MAE values for all considered threshold parameter values. In particular, their performances increase as α increases (up to $\alpha = 0.10$), indicating that feature selection helps to simplify the models and reduce overfitting. More specifically, Figure 6.3 highlights that setting α to 0.10 produces the best performance for both models by striking a balance between having too many features, which may lead to overfitting issues, and too few features, which may lead to an oversimplification of models. On the other hand, the bad outcomes concerning LR are directly attributable to the failure of the model to describe the complex and non-linear relation between UMJESTs and variables.

Table 6.3: Computational results of EEM, LR, RF, and GBM.

Model	α	#features	MSE	RMSE	MAE	RMSLE
EEM	-	-	4080.60	61.28	46.58	-
LR	0.00	120	6422.50	80.14	58.53	-
RF	0.00	120	1995.00	44.66	36.67	0.42
GBM	0.00	120	1827.35	42.74	34.81	0.60
LR	0.05	52	4355.59	66.00	50.63	-
RF	0.05	52	1900.12	43.59	34.89	0.41
GBM	0.05	52	1816.73	42.62	34.65	0.40
LR	0.10	47	4353.31	65.98	50.61	-
RF	0.10	47	1853.33	43.05	34.73	0.40
GBM	0.10	47	1803.27	42.46	34.37	0.44
LR	0.15	40	4312.58	65.67	50.57	-
RF	0.15	40	1875.88	43.31	35.17	0.41
GBM	0.15	40	1821.83	42.68	34.73	0.40
LR	0.20	26	5065.27	71.17	55.12	-
RF	0.20	26	1903.73	43.63	35.69	0.41
GBM	0.20	26	1890.49	43.47	35.26	0.44
LR	0.25	18	5143.02	71.71	55.76	-
RF	0.25	18	2017.39	44.91	36.70	0.42
GBM	0.25	18	2058.74	45.37	37.48	0.43

6.6 Conclusions

In this paper, we faced the problem of predicting ST in the context of real-world manufacturing environments. Although shortcomings in the ST prediction may lead to develop inefficient schedules and represent a relevant source of gap between scheduling theory and practice, ST estimation received limited attention in the literature. To fill this gap, we explored the use of ML regression algorithms to provide good estimations of the ST. We applied these ML algorithms to a real-world scheduling application arising in the color printing industry, where a finite set of jobs must be sequentially performed by a heterogeneous set of parallel flexographic printer machines. Specifically, we dealt with UMJSSTs which are subject to many complexities and typically arise in the context of flexible manufacturing systems.

After data preparation, pre-processing, and statistical analysis of a real-world industrial dataset, we trained three different ML models, namely, LR, RF, and GBM. Computational experiments assessed the effectiveness of GBM approach, which obtained the best performance overall, immediately followed by RF. For both models, the mean squared error on the predicted UMJESTs is less than a third of LR and less than half of that of EEM, a heuristic approach tailored for the same application.

The obtained results are of interest because the ML models can be easily adapted to deal with ST evaluation in many other scheduling problems. In particular, as a possible topic for future research, we are interested in embedding this approach in scheduling algorithms to improve their accuracy and narrow the gap between scheduling theory and practice.

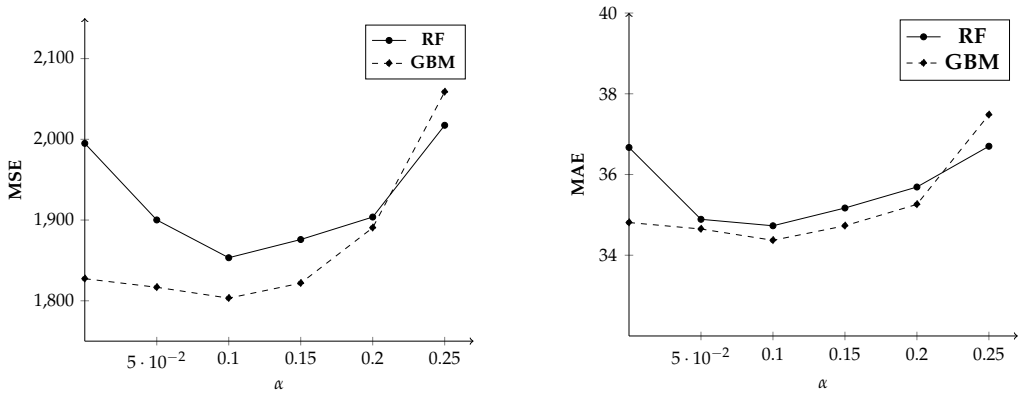


Figure 6.3: MSE, on the left, and MAE, on the right, of RF and GBM, by varying the threshold parameter α.

Acknowledgments

We acknowledge financial support from Istituto Stampa s.r.l. (Italy). The work originates from the daily activity of Istituto Stampa s.r.l., a company whose headquarters is located in Reggio Emilia (Italy). The company operates in the field of packaging industry by producing and printing packaging materials for food products since 1933.

Appendices

Appendix A

List of Acronyms

A.1 Acronyms, definitions, pages

ΓRKP Γ-Robust Knapsack Problem. 19

2D-UKPGC Two-Dimensional Unbounded Knapsack Problem with Guillotine Constraints. 37

2D-KPGC Two-Dimensional Knapsack Problem with Guillotine Constraints. 37

2D-KP Two-Dimensional Knapsack Problem. 36

3D-KP Three-Dimensional Geometric Orthogonal Knapsack Problem. 38

B&B Branch-and-Bound. 9, 29

B&C Branch-and-Cut. 16, 33

B&P Branch-and-Price. 42

BCMKP Budget-Constrained Multiple Knapsack Problem. 31

BCMP Bounded Change-Making Problem. 12

BKP Bounded Knapsack Problem. 11

BLKP Bilevel Knapsack Problem. 20

BLKPI Bilevel Knapsack Problem with Interdiction Constraints. 21

BOKP Biobjective Knapsack Problem. 45

BPP Bin Packing Problem. 7

C&P Cutting and Packing. 36

CCKP Constrained Compartmentalized Knapsack Problem. 20

CKP Compartmentalized Knapsack Problem. 20

CUF-ToSP ToSP with Constant setup times, Unordered tools, and Fixed sequence of jobs. 2

D-KP d-Dimensional Orthogonal Knapsack Problem. 38

DP Dynamic Programming. 7, 29

EPTAS Efficient Polynomial Time Approximation Scheme. 29

FCMKP Fixed-Charge Multiple Knapsack Problem. 31

FPTAS Fully Polynomial Time Approximation Scheme. 9, 29

FSS Fixed Set Search. 70

GAP Generalized Assignment Problem. 7

GBM Gradient Boosting Machine. 92

GKSP Generalized Knapsack Sharing Problem. 15

GOF-ToSP ToSP with General setup times, Ordered tools, and Fixed sequence of jobs. 2

GOV-ToSP ToSP with General setup times, Ordered tools, and Variable sequence of jobs. 2

GQMKP Generalized Quadratic Multiple Knapsack Problem. 43

GRASP Greedy Randomized Adaptive Search Procedure. 38

GUF-ToSP ToSP with General setup times, Unordered tools, and Fixed sequence of jobs. 2

HEM Heuristic Evaluation Method. 92

ILP Integer Linear Programming. 2, 6, 32

KP01 0-1 Knapsack Problem. 6, 27

KPCG Knapsack Problem with Conflict Graph. 15

KPS Knapsack Problem with Setup. 12, 31

KSP Knapsack Sharing Problem. 15

LP Linear Programming. 6, 33

LR Linear Regression. 92

MCKP Multiple-Choice Knapsack Problem. 14

MdKP Multidimensional Knapsack Problem. 32

MdKP-GUB Multidimensional Knapsack Problem with Generalized Upper Bound Constraints. 36

MdMCKP Multidimensional Multiple-Choice Knapsack Problem. 34

MdQKP Quadratic Knapsack Problem with Multiple Knapsack Constraints. 42

MKAP Multiple Knapsack Assignment Problem. 30

MKP Multiple Knapsack Problem. 29

MKPC Multiple Knapsack Problem with Conflicts. 31

MKPCC Multiple Knapsack Problem with Color Constraints. 31

MKPS Multiple Knapsack Problem with Setup. 31

ML Machine Learning. 92

MMdKP Multiple Multidimensional Knapsack Problem. 34

MMdKPF Multiple Multidimensional Knapsack with Family-Split Penalties. 34

MMKP Max-Min Knapsack Problem. 17

MMKP Multidemand Multidimensional Knapsack Problem. 34

MMRKP Min-Max Regret Knapsack Problem. 18

MOKP Multiobjective Knapsack Problem. 45

MSSP Multiple Subset Sum Problem. 30

NLKP Non-Linear Knapsack Problem. 39

NLMKP Non-Linear Multiple Knapsack Problem. 39

OKP Online Knapsack Problem. 43

ORKP Online Removable Knapsack Problem. 44

PCKP Precedence Constrained Knapsack Problem. 17

PMPST Parallel Print Machine Problem with Setup Times. 69

PTAS Polynomial Time Approximation Scheme. 22, 29

QKP Quadratic Knapsack Problem. 39

QKPCG Quadratic Knapsack Problem with Conflict Graphs. 41

QMKP Quadratic Multiple Knapsack Problem. 42

QMPCBC Quadratic Multiknapsack Problem with Conflicts and Balance Constraints. 43

RCL Restricted Candidate List. 77

RF Random Forest. 92

RKP Rectangular Knapsack Problem. 41

SQKP Symmetric Quadratic Knapsack Problem. 41

SSP Subset Sum Problem. 9

SSPNU Tool Switching Problem with non-Uniform Setup Times. 70

ST Setup Time. 91

SVP Shortest Vector Problem. 30

ToSP Tool Switching Problem. 1, 70

TU Totally Unimodular. 51

UCMP Unbounded Change-Making Problem. 12

UKP Unbounded Knapsack Problem. 11

UMdKP Unbounded Multidimensional Knapsack Problem. 34

UMJSST Uncertain Machine-dependent and Job Sequence-dependent Setup Time. 92

Bibliography

[1] N. Abd-Alsabour. "Binary ant colony optimization for subset problems". In: *Studies in Computational Intelligence* 592 (2015), pp. 105–121.

[2] M. Abdel-Basset et al. "A binary multi-verse optimizer for 0-1 multidimensional knapsack problems with application in interactive multimedia systems". In: *Computers & Industrial Engineering* 132 (2019), pp. 187–206.

[3] A. Adamaszek and M. Adamaszek. "Combinatorics of the change-making problem". In: *European Journal of Combinatorics* 31 (2010), pp. 47–63.

[4] Y. Adouani, B. Jarboui, and M. Masmoudi. "A matheuristic for the 0-1 generalized quadratic multiple knapsack problem". In: *Annals of Operations Research* (2019 (forthcoming)), pp. 1–23.

[5] A. Agra and C. Requejo. "The linking set problem: A polynomial special case of the multiple-choice knapsack problem". In: *Journal of Mathematical Sciences* 161 (2009), pp. 919–929.

[6] R. Ahuja and C. Cunha. "Very large-scale neighborhood search for the *K*-constraint multiple knapsack problem". In: *Journal of Heuristics* 11 (2005), pp. 465–481.

[7] S. Aiello et al. "Machine learning with python and h2o". In: *H2O. ai Inc* (2016).

[8] H. Aissi, C. Bazgan, and D. Vanderpooten. "Min-max and min-max regret versions of combinatorial optimization problems: A survey". In: *European Journal of Operational Research* 197 (2009), pp. 427–438.

[9] M. Akbar et al. "Heuristic Solutions for the Multiple-Choice Multi-dimension Knapsack Problem". In: *Computational Science - ICCS 2001 (LNCS, volume 2074)*. Springer, 2001, pp. 659–668.

[10] M. Akbar et al. "Solving the multidimensional multiple-choice knapsack Problem by constructing convex hulls". In: *Computers & Operations Research* 33 (2006), pp. 1259–1273.

[11] H. Akeb, M. Hifi, and M. Ould Ahmed Mounir. "Local branching-based algorithms for the disjunctively constrained knapsack problem". In: *Computers & Industrial Engineering* 60 (2011), pp. 811–820.

[12] U. Akinc. "Approximate and exact algorithms for the fixed-charge knapsack problem". In: *European Journal of Operational Research* 170 (2006), pp. 363–375.

[13] T. Al-douri, M. Hifi, and V. Zissimopoulos. "An iterative algorithm for the Max-Min knapsack problem with multiple scenarios". In: *Operational Research* 21 (2021), 1355–1392.

[14] F. Al-Maliky, M. Hifi, and H. Mhalla. "Sensitivity analysis of the setup knapsack problem to perturbation of arbitrary profits or weights". In: *International Transactions in Operational Research* 25 (2018), pp. 637–666.

[15] S. Al-Shihabi and S. Ólafsson. "A hybrid of Nested Partition, Binary Ant System, and Linear Programming for the multidimensional knapsack problem". In: *Computers & Operations Research* 37 (2010), pp. 247–255.

[16] T. Aldouri and M. Hifi. "A hybrid reactive search for solving the max-min knapsack problem with multi-scenarios". In: *International Journal of Computers & Applications* 40 (2018), pp. 1–13.

[17] A. Allahverdi. "The third comprehensive survey on scheduling problems with setup times/costs". In: *European Journal of Operational Research* 246.2 (2015), pp. 345–378.

[18] A. Allahverdi. "A survey of scheduling problems with no-wait in process". In: *European Journal of Operational Research* 255.3 (2016), pp. 665–686.

[19] A Allahverdi. "A survey of scheduling problems with uncertain interval/bounded processing/setup times". In: *Journal of Project Management* 7 (2022), pp. 255–264.

[20] A. Allahverdi and M. Allahverdi. "Two-machine no-wait flowshop scheduling problem with uncertain setup times to minimize maximum lateness". In: *Computational and Applied Mathematics* 37 (2018), 6774 – 6794.

[21] A. Allahverdi and H. Soroush. "The significance of reducing setup times/setup costs". In: *European Journal of Operational Research* 187 (2008), 978 – 984.

[22] A. Allahverdi et al. "A survey of scheduling problems with setup times or costs". In: *European Journal of Operational Research* 187.3 (2008), pp. 985–1032.

[23] M. Allahverdi. "An improved algorithm to minimize the total completion time in a two-machine no-wait flow-shop with uncertain setup times". In: *Journal of Project Management* 7 (2022), pp. 1–12.

[24] N. Altay, P. Robinson Jr., and K. Bretthauer. "Exact and heuristic solution approaches for the mixed integer setup knapsack problem". In: *European Journal of Operational Research* 190 (2008), pp. 598–609.

[25] R. Alvarez-Valdés, A. Parajón, and J. Tamarit. "A Tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems". In: *Computers & Operations Research* 29 (2002), pp. 925–947.

[26] R. Alvarez-Valdés, F. Parreño, and J. Tamarit. "A Tabu search algorithm for a two-dimensional non-guillotine cutting problem". In: *European Journal of Operational Research* 183 (2007), pp. 1167–1182.

[27] A. Amiri. "A Lagrangean based solution algorithm for the knapsack problem with setups". In: *Expert Systems with Applications* 143 (2020), p. 113077.

[28] A. Amiri and R. Barkhi. "A Lagrangean based solution algorithm for the multiple knapsack problem with setups". In: *Computers & Industrial Engineering* 153 (2021), p. 107089.

[29] J. Ang, C. Cao, and H.-Q. Ye. "Model and algorithms for multi-period sea cargo mix problem". In: *European Journal of Operational Research* 180 (2007), pp. 1381–1393.

[30] E. Angelelli, R. Mansini, and M. Speranza. "Kernel search: A general heuristic for the multi-dimensional knapsack problem". In: *Computers & Operations Research* 37 (2010), pp. 2017–2026.

[31] D. Applegate et al. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2011.

[32] H. Arntzen, L. Hvattum, and A. Løkketangen. "Adaptive memory search for multidemand multidimensional knapsack problems". In: *Computers & Operations Research* 33 (2006), pp. 2508–2525.

[33] M. Avci and S. Topaloglu. "A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem". In: *Computers & Operations Research* 83 (2017), pp. 54–65.

[34] A. Aydilek, H. Aydilek, and A. Allahverdi. "Increasing the profitability and competitiveness in a production environment with random and bounded setup times". In: *International Journal of Production Research* 51 (2013), 106 – 117.

[35] A. Aydilek, H. Aydilek, and A. Allahverdi. "Production in a two-machine flowshop scheduling environment with uncertain processing and setup times to minimize makespan". In: *International Journal of Production Research* 53 (2015), 2803 – 2819.

[36] M. Azad, A. Rocha, and E. Fernandes. "A simplified binary artificial fish swarm algorithm for 0-1 quadratic knapsack problems". In: *Journal of Computational and Applied Mathematics* 259 (2014), pp. 897–904.

[37] M. Baldi, G. Perboli, and R. Tadei. "The three-dimensional knapsack problem with balancing constraints". In: *Applied Mathematics and Computation* 218 (2012), pp. 9802–9818.

[38] S. Balev et al. "A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem". In: *European Journal of Operational Research* 186 (2008), pp. 63–76.

[39] C. Basnet. "Heuristics for the multiple knapsack problem with conflicts". In: *International Journal of Operational Research* 32 (2018), pp. 514–525.

[40] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[41] J. Beasley. "A population heuristic for constrained two-dimensional non-guillotine cutting". In: *European Journal of Operational Research* 156 (2004), pp. 601–627.

[42] H. Becker and L. Buriol. "An empirical analysis of exact algorithms for the unbounded knapsack problem". In: *European Journal of Operational Research* 277 (2019), pp. 84–99.

[43] E. Bednarczuk, J. Miroforidis, and P. Pyzel. "A multi-criteria approach to approximate solution of multiple-choice knapsack problem". In: *Computational Optimization and Applications* 70 (2018), pp. 889–910.

[44] J. Behnamian and S. Fatemi Ghomi. "Multi-objective fuzzy multiprocessor flowshop scheduling". In: *Applied Soft Computing Journal* 21 (2014), 139 – 148.

[45] T. Bektas and O. Oğuz. "On separating cover inequalities for the multidimensional knapsack problem". In: *Computers & Operations Research* 34 (2007), pp. 1771–1776.

[46] T. Belgacem and M. Hifi. "Sensitivity analysis of the knapsack sharing problem: Perturbation of the weight of an item". In: *Computers & Operations Research* 35 (2008), pp. 295–308.

[47] T. Belgacem and M. Hifi. "Sensitivity analysis of the optimum to perturbation of the profit of a subset of items in the binary knapsack problem". In: *Discrete Optimization* 5 (2008), pp. 755–761.

[48] R. Bellman. "Comment on Dantzig's Paper on Discrete Variable Extremum Problems". In: *Operations Research* 5 (1957), pp. 723–724.

[49] D. Bergman. "An exact algorithm for the quadratic multiknapsack problem with an application to event seating". In: *INFORMS Journal on Computing* 31 (2019), pp. 477–492.

[50] A. Bettinelli, V. Cacchiani, and E. Malaguti. "A Branch-and-Bound Algorithm for the Knapsack Problem with Conflict Graph". In: *INFORMS Journal on Computing* 29 (2017), pp. 457–473.

[51] D. Bienstock et al. "On inequalities with bounded coefficients and pitch for the min knapsack polytope". In: *Discrete Optimization* (2020 (forthcoming)).

[52] A. Billionnet, A. Faye, and E. Soutif. "A new upper bound for the 0-1 quadratic knapsack problem". In: *European Journal of Operational Research* 112 (1999), pp. 664–672.

[53] A. Billionnet and E. Soutif. "An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem". In: *European Journal of Operational Research* 157 (2004), pp. 565–575.

[54] A. Billionnet and E. Soutif. "Using a mixed integer programming tool for solving the 0-1 quadratic knapsack problem". In: *INFORMS Journal on Computing* 16 (2004), pp. 188–197.

[55] H.-J. Böckenhauer et al. "The online knapsack problem: Advice and randomization". In: *Theoretical Computer Science* 527 (2014), pp. 61–72.

[56] N. Boland et al. "Clique-based facets for the precedence constrained knapsack problem". In: *Mathematical Programming* 133 (2012), pp. 481–511.

[57] S. Boora and S. Verma. "To Study Print Contrast Variation on Art and Map Litho Paper Due to Effect of Ink Sequence". In: *International Journal of Science, Engineering and Computer Technology* 7 (2017), pp. 125–127.

[58] I. Borgulya. "An EDA for the 2D knapsack problem with guillotine constraint". In: *Central European Journal of Operations Research* 27 (2019), pp. 329–356.

[59] G. Borradaile, B. Heeringa, and G. Wilfong. "The knapsack problem with neighbour constraints". In: *Journal of Discrete Algorithms* 16 (2012), pp. 224–235.

[60] A. Bortfeldt and T. Winter. "A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces". In: *International Transactions in Operational Research* 16 (2009), pp. 685–713.

[61] S. Boussier et al. "A multi-level search strategy for the 0-1 Multidimensional Knapsack Problem". In: *Discrete Applied Mathematics* 158 (2010), pp. 97–109.

[62] V. Boyer, D. El Baz, and M. Elkihel. "Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound". In: *European Journal of Industrial Engineering* 4 (2010), pp. 434–449.

[63] V. Boyer, M. Elkihel, and D. El Baz. "Heuristics for the 0-1 multidimensional knapsack problem". In: *European Journal of Operational Research* 199 (2009), pp. 658–664.

[64] L. Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[65] L. Breiman et al. *Classification and regression trees*. Routledge, 2017.

[66] K. Bretthauer and B. Shetty. "The nonlinear knapsack problem - algorithms and applications". In: *European Journal of Operational Research* 138 (2002), pp. 459–472.

[67] K. Bringmann. "A near-linear pseudopolynomial time algorithm for subset sum". In: *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*. 2017, pp. 1073–1084.

[68] L. Brotcorne, S. Hanafi, and R. Mansi. "A dynamic programming algorithm for the bilevel knapsack problem". In: *Operations Research Letters* 37 (2009), pp. 215–218.

[69] L. Brotcorne, S. Hanafi, and R. Mansi. "One-level reformulation of the bilevel Knapsack problem using dynamic programming". In: *Discrete Optimization* 10 (2013), pp. 1–10.

[70] A. Burger et al. "Scheduling multi-colour print jobs with sequence-dependent setup times". In: *Journal of Scheduling* 18.2 (2015), pp. 131–145.

[71] C. Büsing, A. Koster, and M. Kutschka. "Recoverable robust knapsacks: The discrete scenario case". In: *Optimization Letters* 5 (2011), pp. 379–392.

[72] C. Büsing et al. "Formulations and algorithms for the recoverable Γ-robust knapsack problem". In: *EURO Journal on Computational Optimization* 7 (2019), pp. 15–45.

[73] D. Calmels. "The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends". In: *International Journal of Production Research* 57.15-16 (2019), pp. 5005–5025.

[74] D. Calmels. "An iterated local search procedure for the job sequencing and tool switching problem with non-identical parallel machines". In: *European Journal of Operational Research* 297 (2022), 66 – 85.

[75] A. Candia-Véjar, E. Álvarez-Miranda, and N. MacUlan. "Minmax regret combinatorial optimization problems: An Algorithmic Perspective". In: *RAIRO - Operations Research* 45 (2011), pp. 101–129.

[76] P. Cappanera and M. Trubian. "A local-search-based heuristic for the demand-constrained multidimensional knapsack problem". In: *INFORMS Journal on Computing* 17 (2005), pp. 82–98.

[77] A. Caprara, F. Furini, and E. Malaguti. "Uncommon Dantzig-Wolfe reformulation for the temporal knapsack problem". In: *INFORMS Journal on Computing* 25 (2013), pp. 560–571.

[78] A. Caprara, H. Kellerer, and U. Pferschy. "A PTAS for the Multiple Subset Sum Problem with different knapsack capacities". In: *Information Processing Letters* 73 (2000), pp. 111–118.

[79] A. Caprara, H. Kellerer, and U. Pferschy. "The Multiple Subset Sum Problem". In: *SIAM Journal on Optimization* 11 (2000), pp. 308–319.

[80] A. Caprara, H. Kellerer, and U. Pferschy. "A 3/4-approximation algorithm for multiple subset sum". In: *Journal of Heuristics* 9 (2003), pp. 99–111.

[81] A. Caprara, A. Lodi, and M. Monaci. "An approximation scheme for the two-stage, two-dimensional knapsack problem". In: *Discrete Optimization* 7 (2010), pp. 114–124.

[82] A. Caprara and M. Monaci. "On the two-dimensional Knapsack Problem". In: *Operations Research Letters* 32 (2004), pp. 5–14.

[83] A. Caprara, D. Pisinger, and P. Toth. "Exact Solution of the Quadratic Knapsack Problem". In: *INFORMS Journal on Computing* 11 (1999), pp. 125–137.

[84] A. Caprara et al. "Packing into the smallest square: Worst-case analysis of lower bounds". In: *Discrete Optimization* 3 (2006), pp. 317–326.

[85] A. Caprara et al. "A study on the computational complexity of the bilevel knapsack problem". In: *SIAM Journal on Optimization* 24 (2014), pp. 823–838.

[86] A. Caprara et al. "Bilevel Knapsack with interdiction constraints". In: *INFORMS Journal on Computing* 28 (2016), pp. 319–333.

[87] A. Caprara et al. "Solving the Temporal Knapsack Problem via Recursive Dantzig-Wolfe Reformulation". In: *Information Processing Letters* 116 (2016), pp. 379–386.

[88] J. Valério de Carvalho. "Exact solution of bin-packing problems using column generation and branch-and-bound". In: *Annals of Operations Research* 86 (1999), pp. 629–659.

[89] M. Carvalho, A. Lodi, and P. Marcotte. "A polynomial algorithm for a continuous bilevel knapsack problem". In: *Operations Research Letters* 46 (2018), pp. 185–188.

[90] M. Caserta, E. Quiñonez Rico, and A. Márquez Uribe. "A cross entropy algorithm for the Knapsack problem with setups". In: *Computers & Operations Research* 35 (2008), pp. 241–252.

[91] M. Caserta and S. Voß. "The robust multiple-choice multidimensional knapsack problem". In: *Omega* 86 (2019), pp. 16–27.

[92] F. Castillo-Zunino and P. Keskinocak. "Bi-criteria multiple knapsack problem with grouped items". In: *Journal of Heuristics* 27 (2021), pp. 747–789.

[93] A. Cerqueus, A. Przybylski, and X. Gandibleux. "Surrogate upper bound sets for bi-objective bi-dimensional binary knapsack problems". In: *European Journal of Operational Research* 244 (2015), pp. 417–433.

[94] A. Cerqueus et al. "On branching heuristics for the bi-objective 0/1 unidimensional knapsack problem". In: *Journal of Heuristics* 23 (2017), pp. 285–319.

[95] A. Ceselli and G. Righini. "An optimization algorithm for a penalized knapsack problem". In: *Operations Research Letters* 34 (2006), pp. 394–404.

[96] E. Chajakis and M. Guignard. "Exact algorithms for the setup knapsack problem". In: *INFOR: Information Systems and Operational Research* 32 (1994), pp. 124–142.

[97] T. M. Chan. "Approximation schemes for 0-1 knapsack". In: *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. 2018, 5:1–5:12.

[98] K. Chebil and M. Khemakhem. "A dynamic programming algorithm for the Knapsack Problem with Setup". In: *Computers & Operations Research* 64 (2015), pp. 40–50.

[99] C. Chekuri and S. Khanna. "A polynomial time approximation scheme for the multiple knapsack problem". In: *SIAM Journal on Computing* 35 (2006), pp. 713–728.

[100] L. Chen and G. Zhang. "Approximation algorithms for a bi-level knapsack problem". In: *Theoretical Computer Science* 497 (2013), pp. 1–12.

[101] L. Chen and G. Zhang. "Packing groups of items into multiple knapsacks". In: *ACM Transactions on Algorithms* (2018).

[102] Y. Chen and J.-K. Hao. "A "reduce and solve" approach for the multiple-choice multidimensional knapsack problem". In: *European Journal of Operational Research* 239 (2014), pp. 313–322.

[103] Y. Chen and J.-K. Hao. "Iterated responsive threshold search for the quadratic multiple knapsack problem". In: *Annals of Operations Research* 226 (2014), pp. 101–131.

[104] Y. Chen and J.-K. Hao. "Memetic search for the generalized quadratic multiple knapsack problem". In: *IEEE Transactions on Evolutionary Computation* 20 (2016), pp. 908–923.

[105] Y. Chen and J.-K. Hao. "An iterated "hyperplane exploration" approach for the quadratic knapsack problem". In: *Computers & Operations Research* 77 (2017), pp. 226–239.

[106] Y. Chen, J.-K. Hao, and F. Glover. "An evolutionary path relinking approach for the quadratic multiple knapsack problem". In: *Knowledge-Based Systems* 92 (2016), pp. 23–34.

[107] C.-Y. Cheng et al. "Learning-based metaheuristic for scheduling unrelated parallel machines with uncertain setup times". In: *IEEE Access* 8 (2020), pp. 74065–74082.

[108] N. Cherfi and M. Hifi. "Hybrid algorithms for the Multiple-choice Multidimensional Knapsack Problem". In: *International Journal of Operational Research* 5 (2009), pp. 89–109.

[109] N. Cherfi and M. Hifi. "A column generation method for the multiple-choice multi-dimensional knapsack problem". In: *Computational Optimization and Applications* 46 (2010), pp. 51–73.

[110] M. Chih. "Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem". In: *Applied Soft Computing* 26 (2015), pp. 378–389.

[111] M. Chih. "Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem". In: *Swarm and Evolutionary Computation* 39 (2018), pp. 279–296.

[112] M. Chih et al. "Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem". In: *Applied Mathematical Modelling* 38 (2014), pp. 1338–1350.

[113] H. Choe and C. Choe. "The k-subset sum problem over finite fields of characteristic 2". In: *Finite Fields and their Applications* 59 (2019), pp. 175–184.

[114] H. Christensen et al. "Approximation and online algorithms for multidimensional bin packing: A survey". In: *Computer Science Review* 24 (2017), pp. 63–79.

[115] G. Chryssolouris. *Manufacturing systems: theory and practice*. Springer Science & Business Media, 2013.

[116] P. Chu and J. Beasley. "A Genetic Algorithm for the Multidimensional Knapsack Problem". In: *Journal of Heuristics* 4 (1998), pp. 63–86.

[117] G. Claßen, A. Koster, and A. Schmeink. "The multi-band robust knapsack problem - A dynamic programming approach". In: *Discrete Optimization* 18 (2015), pp. 123–149.

[118] F. Clautiaux, B. Detienne, and G. Guillot. "An iterative dynamic programming approach for the temporal knapsack problem". In: *European Journal of Operational Research* 293 (2021), pp. 442–456.

[119] E. Coffman et al. "Bin packing approximation algorithms: survey and classification". In: *Handbook of combinatorial optimization, 2nd edition*. Ed. by D.-Z. Du, P. Pardalos, and R. Graham. Springer, 2013, pp. 455–531.

[120] E. Conde. "On the complexity of the continuous unbounded knapsack problem with uncertain coefficients". In: *Operations Research Letters* 33 (2005), pp. 481–485.

[121] S. Coniglio, F. Furini, and P. San Segundo. "A new combinatorial branch-and-bound algorithm for the Knapsack Problem with Conflicts". In: *European Journal of Operational Research* 289 (2021), pp. 435–455.

[122] P. Correia, L. Paquete, and J. Figueira. "Compressed data structures for bi-objective {0,1}-knapsack problems". In: *Computers & Operations Research* 89 (2018), pp. 82–93.

[123] L. Cowen, R. Cowen, and A. Steinberg. "Totally greedy coin sets and greedy obstructions". In: *The Electronic Journal of Combinatorics* 15 (2008), R90.

[124] Y. Crama. "Combinatorial optimization models for production scheduling in automated manufacturing systems". In: *European Journal of Operational Research* 99 (1997), pp. 136–153.

[125] Y. Crama et al. "Minimizing the number of tool switches on a flexible machine". In: *International Journal of Flexible Manufacturing Systems* 6.1 (1994), pp. 33–54.

[126] Y. Crama et al. "The tool switching problem revisited". In: *European Journal of Operational Research* 182 (2007), 952 – 957.

[127] Y. Crama et al. "The tool switching problem revisited". In: *European Journal of Operational Research* 182 (2007), 952 – 957.

[128] I. Crévits et al. "Iterative semi-continuous relaxation heuristics for the multiple-choice multidimensional knapsack problem". In: *Computers & Operations Research* 39 (2012), pp. 32–41.

[129] J. Cunha, V. L. de Lima, and T. Queiroz. "Grids for cutting and packing problems: a study in the 2D knapsack problem". In: *4OR* 18 (2020), pp. 293–339.

[130] J. Cunha, L. Simonetti, and A. Lucena. "Lagrangian heuristics for the Quadratic Knapsack Problem". In: *Computational Optimization and Applications* 63 (2016), pp. 97–120.

[131] V. Curtis and C. Sanches. "A low-space algorithm for the subset-sum problem on GPU". In: *Computers & Operations Research* 83 (2017), pp. 120–124.

[132] V. Curtis and C. Sanches. "An improved balanced algorithm for the subset-sum problem". In: *European Journal of Operational Research* 275 (2019), pp. 460–466.

[133] M. Cygan, Ł. Jeż, and J. Sgall. "Online Knapsack Revisited". In: *Theory of Computing Systems* 58 (2016), pp. 153–190.

[134] I. Dahmani and M. Hifi. "A modified descent method-based heuristic for binary quadratic knapsack problems with conflict graphs". In: *Annals of Operations Research* 298 (2021), 125–147.

[135] I. Dahmani, M. Hifi, and L. Wu. "An exact decomposition algorithm for the generalized knapsack sharing problem". In: *European Journal of Operational Research* 252 (2016), pp. 761–774.

[136] I. Dahmani et al. "A swarm optimization-based search algorithm for the quadratic knapsack problem with conflict Graphs". In: *Expert Systems with Applications* 148 (2020), p. 113224.

[137] C. D'Ambrosio and S. Martello. "Heuristic algorithms for the general non-linear separable knapsack problem". In: *Computers & Operations Research* 38 (2011), pp. 505–513.

[138] C. D'Ambrosio, S. Martello, and L. Mencarelli. "Relaxations and heuristics for the multiple non-linear separable knapsack problem". In: *Computers & Operations Research* 93 (2018), pp. 79–89.

[139] C. D'Ambrosio, S. Martello, and M. Monaci. "Lower and upper bounds for the non-linear generalized assignment problem". In: *Computers & Operations Research* 120 (2020), p. 104933.

[140] C. D'Ambrosio et al. "On the Product Knapsack Problem". In: *Optimization Letters* 12 (2018), pp. 691–712.

[141] G. Dantzig. "Discrete-Variable Extremum Problems". In: *Operations Research* 5 (1957), pp. 266–277.

[142] T. Dantzig. *Number: The language of science*. Penguin, 2007.

[143] M. Daoud and D. Chaabane. "New reduction strategy in the biobjective knapsack problem". In: *International Transactions in Operational Research* 25 (2018), pp. 1739–1762.

[144] A. Darmann, U. Pferschy, and J. Schauer. "Resource allocation with time intervals". In: *Theoretical Computer Science* 411 (2010), pp. 4217–4234.

[145] A. Darmann et al. "The Subset Sum game". In: *European Journal of Operational Research* 233 (2014), pp. 539–549.

[146] N. Dayama et al. "History-dependent scheduling: Models and algorithms for scheduling with general precedence and sequence dependence". In: *Computers and Operations Research* 64 (2015), pp. 245–261.

[147] V. Deineko and G. Woeginger. "Pinpointing the complexity of the interval min-max regret knapsack problem". In: *Discrete Optimization* 7 (2010), pp. 191–196.

[148] V. Deineko and G. Woeginger. "A well-solvable special case of the bounded knapsack problem". In: *Operations Research Letters* 39 (2011), pp. 118–120.

[149] V. Deineko and G. Woeginger. "Unbounded knapsack problems with arithmetic weight sequences". In: *European Journal of Operational Research* 213 (2011), pp. 384–387.

[150] F. Della Croce and A. Grosso. "Computational experience with a core-based reduction procedure for the 2-knapsack problem". In: *Computers & Operations Research* 38 (2011), pp. 514–516.

[151] F. Della Croce and A. Grosso. "Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem". In: *Computers & Operations Research* 39 (2012), pp. 27–31.

[152] F. Della Croce, U. Pferschy, and R. Scatamacchia. "Approximating the 3-period Incremental Knapsack Problem". In: *Journal of Discrete Algorithms* 52 (2018), pp. 55–69.

[153] F. Della Croce, U. Pferschy, and R. Scatamacchia. "New exact approaches and approximation results for the Penalized Knapsack Problem". In: *Discrete Applied Mathematics* 253 (2019), pp. 122–135.

[154] F. Della Croce, U. Pferschy, and R. Scatamacchia. "On approximating the Incremental Knapsack Problem". In: *Discrete Applied Mathematics* 264 (2019), pp. 26–42.

[155] F. Della Croce, F. Salassa, and R. Scatamacchia. "A new exact approach for the 0-1 Collapsing Knapsack Problem". In: *European Journal of Operational Research* 260 (2017), pp. 56–69.

[156] F. Della Croce, F. Salassa, and R. Scatamacchia. "An exact approach for the 0-1 knapsack problem with setups". In: *Computers & Operations Research* 80 (2017), pp. 61–67.

[157] F. Della Croce and R. Scatamacchia. "An exact approach for the bilevel knapsack problem with interdiction constraints and extensions". In: *Mathematical Programming* 183 (2020), pp. 249–281.

[158] M. Dell'Amico et al. "Mathematical models and decomposition methods for the multiple knapsack problem". In: *European Journal of Operational Research* 274 (2019), pp. 886–899.

[159] M. Delorme and M. Iori. "Enhanced Pseudo-polynomial Formulations for Bin Packing and Cutting Stock Problems". In: *INFORMS Journal on Computing* 32 (2020), pp. 101–119.

[160] M. Delorme, M. Iori, and S. Martello. "Bin packing and cutting stock problems: Mathematical models and exact algorithms". In: *European Journal of Operational Research* 255 (2016), pp. 1–20.

[161] S. Dempe and K. Richter. "Bilevel programming with knapsack constraints." In: *Central European Journal of Operations Research* 8 (2000), pp. 93–107.

[162] S. DeNegre. "Interdiction and discrete bilevel linear programming". PhD thesis. Bethlehem, PA: Lehigh University, 2011.

[163] P. Detti. "A new upper bound for the multiple knapsack problem". In: *Computers & Operations Research* 129 (2021), p. 105210.

[164] R. Diao, Y.-F. Liu, and Y.-H. Dai. "A new fully polynomial time approximation scheme for the interval subset sum problem". In: *Journal of Global Optimization* 68 (2017), pp. 749–775.

[165] F. Diedrich et al. "Approximation algorithms for 3D orthogonal knapsack". In: *Journal of Computer Science and Technology* 23 (2008), pp. 749–762.

[166] N. Dimitrov et al. "Emergency relocation of items using single trips: Special cases of the Multiple Knapsack Assignment Problem". In: *European Journal of Operational Research* 258 (2017), pp. 938–942.

[167] M. Durasević and D. Jakobović. *Heuristic and Metaheuristic Methods for the Unrelated Machines Scheduling Problem: A Survey*. arXiv:2107.13106. July 2021.

[168] J. Egeblad and D. Pisinger. "Heuristic approaches for the two- and three-dimensional knapsack packing problem". In: *Computers & Operations Research* 36 (2009), pp. 1026–1049.

[169] M. Ehrgott. *Multicriteria optimization*. Vol. 491. Springer Science & Business Media, 2005.

[170] M. Ehrgott. *Multicriteria optimization*. Vol. 491. Springer, 2005.

[171] K. Elbassioni, A. Karapetyan, and T. Nguyen. "Approximation schemes for r-weighted Minimization Knapsack problems". In: *Annals of Operations Research* 279 (2019), pp. 367–386.

[172] D. Espinoza, M. Goycoolea, and E. Moreno. "The precedence constrained knapsack problem: Separating maximally violated inequalities". In: *Discrete Applied Mathematics* 194 (2015), pp. 65–80.

[173] Y. Faenza and I. Malinovic. "A PTAS for the time-invariant incremental knapsack problem". In: *International Symposium on Combinatorial Optimization*. 2018, pp. 157–169.

[174] M. Fampa et al. "Parametric convex quadratic relaxation of the quadratic knapsack problem". In: *European Journal of Operational Research* 281 (2020), pp. 36–49.

[175] Z. Feng, Z Ren, and A. Zhan. "Fusing ant colony optimization with Lagrangian relaxation for the multiple-choice multidimensional knapsack problem". In: *Information Sciences* 182 (2012), pp. 15–29.

[176] J. Figueira et al. "Algorithmic improvements on dynamic programming for the bi-objective {0,1} knapsack problem". In: *Computational Optimization and Applications* 56 (2013), pp. 97–111.

[177] D. Fischer and G. Woeginger. "A faster algorithm for the continuous bilevel knapsack problem". In: *Operations Research Letters* 48 (2020), pp. 784–786.

[178] M. Fischetti et al. "Interdiction games and monotonicity, with application to knapsack problems". In: *INFORMS Journal on Computing* 31 (2019), pp. 390–410.

[179] K. Fleszar. "A Branch-and-Bound Algorithm for the Quadratic Multiple Knapsack Problem". In: *European Journal of Operational Research* (2021 (forthcoming)).

[180] F. Fomeni, K. Kaparis, and A. Letchford. "A cut-and-branch algorithm for the Quadratic Knapsack Problem". In: *Discrete Optimization* (2020 (forthcoming)).

[181] F. Fomeni and A. Letchford. "A dynamic programming heuristic for the quadratic knapsack problem". In: *INFORMS Journal on Computing* 26 (2014), pp. 173–182.

[182] D. G. Fon-Der-Flaass. "Arrays of distinct representatives–a very simple NP-complete problem". In: *Discrete Mathematics* 1 (1997), pp. 295–298.

[183] J. Forrest, J. Kalagnanam, and L. Ladanyi. "A column-generation approach to the multiple knapsack problem with color constraints". In: *INFORMS Journal on Computing* 18 (2006), pp. 129–134.

[184] A. Fréville. "The multidimensional 0-1 knapsack problem: An overview". In: *European Journal of Operational Research* 155 (2004), pp. 1–21.

[185] A. Fréville and S. Hanafi. "The multidimensional 0-1 knapsack problem-bounds and computational aspects". In: *Annals of Operations Research* 139 (2005), pp. 195–227.

[186] A. M. Frieze. "Complexity of a 3-dimensional assignment problem". In: *European Journal of Operational Research* 13 (1983), pp. 161–164.

[187] M. Fujimoto and T. Yamada. "An exact algorithm for the knapsack sharing problem with common items". In: *European Journal of Operational Research* 171 (2006), pp. 693–707.

[188] A. Fukunaga. "A branch-and-bound algorithm for hard multiple knapsack problems". In: *Annals of Operations Research* 184 (2011), pp. 97–119.

[189] A. Fukunaga and R. Korf. "Bin completion algorithms for multicontainer packing, knapsack, and covering problems". In: *Journal of Artificial Intelligence Research* 28 (2007), pp. 393–429.

[190] F. Furini, I. Ljubić, and M. Sinnl. "An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem". In: *European Journal of Operational Research* 262 (2017), pp. 438–448.

[191] F. Furini, M. Monaci, and E. Traversi. "Exact approaches for the knapsack problem with setups". In: *Computers & Operations Research* 90 (2018), pp. 208–220.

[192] F. Furini et al. "Heuristic and exact algorithms for the interval min-max regret knapsack problem". In: *INFORMS Journal on Computing* 27 (2015), pp. 392–405.

[193] A. Gál et al. "Space-efficient approximations for subset sum". In: *ACM Transactions on Computation Theory* 8 (2016).

[194] L. Galli et al. "Polynomial-Size Formulations and Relaxations for the Quadratic Multiple Knapsack Problem". In: *European Journal of Operational Research* 291 (2021), pp. 871–882.

[195] C. Gao et al. "An iterative pseudo-gap enumeration approach for the Multidimensional Multiple-choice Knapsack Problem". In: *European Journal of Operational Research* 260 (2017), pp. 1–11.

[196] J. García et al. "Enhancing a machine learning binarization framework by perturbation operators: analysis on the multidimensional knapsack problem". In: *International Journal of Machine Learning and Cybernetics* 11 (2020), pp. 1951–1970.

[197] C. García-Martínez, F. Rodriguez, and M. Lozano. "Tabu-enhanced iterated greedy algorithm: A case study in the quadratic multiple knapsack problem". In: *European Journal of Operational Research* 232 (2014), pp. 454–463.

[198] C. García-Martínez et al. "Strategic oscillation for the quadratic multiple knapsack problem". In: *Computational Optimization and Applications* 58 (2014), pp. 161–185.

[199] M. Garey and D. Johnson. *Computers and intractability*. Freeman, San Francisco, 1979.

[200] T. Ghasemi and M. Razzazi. "Development of core to solve the multidimensional multiple-choice knapsack problem". In: *Computers & Industrial Engineering* 60 (2011), pp. 349–360.

[201] D. Ghosh, N. Chakravarti, and G. Sierksma. "Sensitivity analysis of a greedy heuristic for knapsack problems". In: *European Journal of Operational Research* 169 (2006), pp. 340–350.

[202] A. Ghouila-Houri. "Caractérisation des matrices totalement unimodulaires". In: *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences (Paris)* 254 (1962), pp. 1192–1194.

[203] P. C. Gilmore and R. E. Gomory. "The theory and computation of knapsack functions". In: *Operations Research* 14 (1966), pp. 1045–1074.

[204] A. Giudici et al. "Approximation schemes for the parametric knapsack problem". In: *Information Processing Letters* 120 (2017), pp. 11–15.

[205] S. Goebbels, F. Gurski, and D. Komander. "The knapsack problem with special neighbor constraints". In: *Mathematical Methods of Operations Research* (2021 (forthcoming)).

[206] S. Goebbels et al. "Change-making problems revisited: a parameterized point of view". In: *Journal of Combinatorial Optimization* 34 (2017), pp. 1218–1236.

[207] M. Goerigk. "A note on upper bounds to the robust knapsack problem with discrete scenarios". In: *Annals of Operations Research* 223 (2014), pp. 461–469.

[208] M. Goerigk et al. "The robust knapsack problem with queries". In: *Computers & Operations Research* 55 (2015), pp. 12–22.

[209] E. Gokce and W. Wilhelm. "Valid inequalities for the multi-dimensional multiple-choice 0-1 knapsack problem". In: *Discrete Optimization* 17 (2015), pp. 25–54.

[210] E. González-Neira, J. Montoya-Torres, and D. Barrera. "Flow-shop scheduling problem under uncertainties: Review and trends". In: *International Journal of Industrial Engineering Computations* 8 (2017), pp. 399–426.

[211] P. Goos et al. "A nonlinear multidimensional knapsack problem in the optimal design of mixture experiments". In: *European Journal of Operational Research* 281 (2020), pp. 201–221.

[212] L. Gourvès, J. Monnot, and L. Tlilane. "Subset sum problems with digraph constraints". In: *Journal of Combinatorial Optimization* 36 (2018), pp. 937–964.

[213] R. Graham et al. "Optimization and approximation in deterministic sequencing and scheduling: A survey". In: *Annals of Discrete Mathematics* 5.C (1979), pp. 287–326.

[214] T. Gschwind and S. Irnich. "Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities". In: *OR Spectrum* 39 (2017), pp. 541–556.

[215] H. Gu. "Improving problem reduction for 0-1 Multidimensional Knapsack Problems with valid inequalities". In: *Computers & Operations Research* 71 (2016), pp. 82–89.

[216] F. Gurski and C. Rehs. "Solutions for the knapsack problem with conflict and forcing graphs of bounded clique-width". In: *Mathematical Methods of Operations Research* 89 (2019), pp. 411–432.

[217] B. Haddar et al. "A hybrid heuristic for the 0-1 Knapsack Sharing Problem". In: *Expert Systems with Applications* 42 (2015), pp. 4653–4666.

[218] B. Haddar et al. "A hybrid quantum particle swarm optimization for the Multidimensional Knapsack Problem". In: *Engineering Applications of Artificial Intelligence* 55 (2016), pp. 1–13.

[219] B. Haddar et al. "A quantum particle swarm optimization for the 0-1 generalized knapsack sharing problem". In: *Natural Computing* 15 (2016), pp. 153–164.

[220] E. Hadjiconstantinou and M. Iori. "A hybrid genetic algorithm for the two-dimensional single large object placement problem". In: *European Journal of Operational Research* 183 (2007), pp. 1150–1166.

[221] N. Halman, M. Holzhauser, and S. Krumke. "An FPTAS for the knapsack problem with parametric weights". In: *Operations Research Letters* 46 (2018), pp. 487–491.

[222] N. Halman et al. "Bi-criteria path problem with minimum length and maximum survival probability". In: *OR Spectrum* 41.2 (2019), pp. 469–489.

[223] B. Han, J. Leblet, and G. Simon. "Hard multidimensional multiple choice knapsack problems, an empirical study". In: *Computers & Operations Research* 37 (2010), pp. 172–181.

[224] X. Han, Q. Chen, and K. Makino. "Online knapsack problem under concave functions". In: *Theoretical Computer Science* 786 (2019), pp. 88–95.

[225] X. Han, Y. Kawase, and K. Makino. "Online unweighted knapsack problem with removal cost". In: *Algorithmica* 70 (2014), pp. 76–91.

[226] X. Han, Y. Kawase, and K. Makino. "Randomized algorithms for online knapsack problems". In: *Theoretical Computer Science* 562 (2015), pp. 395–405.

[227] X. Han and K. Makino. "Online removable knapsack with limited cuts". In: *Theoretical Computer Science* 411 (2010), pp. 3956–3964.

[228] X. Han and K. Makino. "Online minimization knapsack problem". In: *Theoretical Computer Science* 609 (2016), pp. 185–196.

[229] X. Han et al. "Online removable knapsack problem under convex function". In: *Theoretical Computer Science* 540-541 (2014), pp. 62–69.

[230] S. Hanafi and C. Wilbaut. "Scatter search for the 0-1 multidimensional knapsack problem". In: *Journal of Mathematical Modelling and Algorithms* 7 (2008), pp. 143–159.

[231] S. Hanafi and C. Wilbaut. "Improved convergent heuristics for the 0-1 multidimensional knapsack problem". In: *Annals of Operations Research* 183 (2011), pp. 125–142.

[232] S. Hanafi et al. "Hybrid approaches for the two-scenario max-min knapsack problem". In: *International Transactions in Operational Research* 19 (2012), pp. 353–378.

[233] R. Harren. "Approximation algorithms for orthogonal packing problems for hypercubes". In: *Theoretical Computer Science* 410 (2009), pp. 4504–4532.

[234] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[235] C. He et al. "An improved binary search algorithm for the Multiple-Choice Knapsack Problem". In: *RAIRO - Operations Research* 50 (2016), pp. 995–1001.

[236] X. He, J. Hartman, and P. Pardalos. "Dynamic-Programming-Based Inequalities for the Unbounded Integer Knapsack Problem". In: *Informatica* 27 (2016), pp. 433–450.

[237] Y.-C. He et al. "Exact and approximate algorithms for discounted 0-1 knapsack problem". In: *Information Sciences* 369 (2016), pp. 634–647.

[238] S. Held, W. Cook, and E. Sewell. "Maximum-weight stable sets and safe lower bounds for graph coloring". In: *Mathematical Programming Computation* 4 (2012), pp. 363–381.

[239] S. Heydrich and A. Wiese. "Faster approximation schemes for the two-dimensional knapsack problem". In: *ACM Transactions on Algorithms* 15 (2019), pp. 1–28.

[240] R. Hickman and T. Easton. "Merging valid inequalities over the multiple knapsack polyhedron". In: *International Journal of Operational Research* 24 (2015), pp. 214–227.

[241] M. Hifi. "Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems". In: *Journal of Combinatorial Optimization* 8 (2004), pp. 65–84.

[242] M. Hifi. "An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem". In: *Engineering Optimization* 46 (2014), pp. 1109–1122.

[243] M. Hifi and H. Mhalla. "Sensitivity analysis to perturbations of the weight of a subset of items: The knapsack case study". In: *Discrete Optimization* 10 (2013), pp. 320–330.

[244] M. Hifi, H. M'Halla, and S. Sadfi. "An exact algorithm for the knapsack sharing problem". In: *Computers & Operations Research* 32 (2005), pp. 1311–1324.

[245] M. Hifi, H. Mhalla, and S. Sadfi. "Sensitivity of the optimum to perturbations of the profit or weight of an item in the binary knapsack problem". In: *Journal of Combinatorial Optimization* 10 (2005), pp. 239–260.

[246] M. Hifi, H. Mhalla, and S. Sadfi. "An adaptive algorithm for the knapsack problem: perturbation of the profit or weight of an arbitrary item". In: *European Journal of Industrial Engineering* 2 (2008), pp. 134–152.

[247] M. Hifi and R. M'Hallah. "Special issue on knapsack problems and applications". In: *Computers & Operations Research* 39 (2012), pp. 1–2.

[248] M. Hifi and M. Michrafy. "A reactive local search-based algorithm for the disjunctively constrained knapsack problem". In: *Journal of the Operational Research Society* 57 (2006), pp. 718–726.

[249] M. Hifi and M. Michrafy. "Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem". In: *Computers & Operations Research* 34 (2007), pp. 2657–2673.

[250] M. Hifi, M. Michrafy, and A. Sbihi. "Heuristic algorithms for the multiple-choice multidimensional knapsack problem". In: *Journal of the Operational Research Society* 55 (2004), pp. 1323–1332.

[251] M. Hifi, M. Michrafy, and A. Sbihi. "A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem". In: *Computational Optimization and Applications* 33 (2006), pp. 271–285.

[252] M. Hifi and N. Otmani. "An algorithm for the disjunctively constrained knapsack problem". In: *International Journal of Operational Research* 13 (2012), pp. 22–43.

[253] M. Hifi and S. Sadfi. "The knapsack sharing problem: An exact algorithm". In: *Journal of Combinatorial Optimization* 6 (2002), pp. 35–54.

[254] M. Hifi, S. Saleh, and L. Wu. "A hybrid guided neighborhood search for the disjunctively constrained knapsack problem". In: *Cogent Engineering* 2 (2015), p. 1068969.

[255] M. Hifi and L. Wu. "New upper bounds and exact methods for the knapsack sharing problem". In: *Applied Mathematics and Computation* 227 (2014), pp. 518–530.

[256] M. Hifi and L. Wu. "Lagrangian heuristic-based neighbourhood search for the multiple-choice multi-dimensional knapsack problem". In: *Engineering Optimization* 47 (2015), pp. 1619–1636.

[257] M. Hifi and L. Wu. "An equivalent model for exactly solving the multiple-choice multidimensional knapsack problem". In: *International Journal of Combinatorial Optimization Problems and Informatics* 3 (2012), pp. 43–58.

[258] R. Hill, Y. Cho, and J. Moore. "Problem reduction heuristic for the 0-1 multidimensional knapsack problem". In: *Computers & Operations Research* 39 (2012), pp. 19–26.

[259] C. Hiremath and R. Hill. "New greedy heuristics for the Multiple-choice Multi-dimensional Knapsack Problem". In: *International Journal of Operational Research* 2 (2007), pp. 495–512.

[260] C. Hojny. "Polynomial size IP formulations of knapsack may require exponentially large coefficients". In: *Operations Research Letters* 48 (2020), pp. 612–618.

[261] C. Hojny et al. "Knapsack polytopes: a survey". In: *Annals of Operations Research* 292 (2019), pp. 469–517.

[262] M. Holzhauser and S. Krumke. "An FPTAS for the parametric knapsack problem". In: *Information Processing Letters* 126 (2017), pp. 43–47.

[263] G. Homsi et al. "The assignment and loading transportation problem". In: *European Journal of Operational Research* 289 (2021), pp. 999–1007.

[264] R. Hoto, M. Arenales, and N. Maculan. "The one dimensional Compartmentalised Knapsack Problem: A case study". In: *European Journal of Operational Research* 183 (2007), pp. 1183–1195.

[265] R. Hoto and G. Bressan. "New solutions to the constrained compartmentalised knapsack problem". In: *International Journal of Operational Research* 28 (2017), pp. 472–487.

[266] S. Htiouech, S. Bouamama, and R. Attia. "OSC: solving the multidimensional multi-choice knapsack problem with tight strategic Oscillation using Surrogate Constraints". In: *International Journal of Computer Applications* 73 (2013), pp. 1–7.

[267] T. C. Hu, L. Landa, and M.-T. Shing. "The Unbounded Knapsack Problem". In: *Research Trends in Combinatorial Optimization: Bonn 2008*. Ed. by W. Cook, L. Lovász, and J. Vygen. Springer, 2009, pp. 201–217.

[268] P. Huang, M. Lawley, and T. Morin. "Tight bounds for periodicity theorems on the unbounded Knapsack problem". In: *European Journal of Operational Research* 215 (2011), pp. 319–324.

[269] P. Huang and K. Tang. "A constructive periodicity bound for the unbounded knapsack problem". In: *Operations Research Letters* 40 (2012), pp. 329–331.

[270] X. Huang et al. "Improved firefly algorithm with courtship learning for unrelated parallel machine scheduling problem with sequence-dependent setup times". In: *Journal of Cloud Computing* 11.1 (2022).

[271] L. Hvattum and A. Løkketangen. "Experiments using scatter search for the multidemand multidimensional knapsack problem". In: *Operations Research/ Computer Science Interfaces Series* 39 (2007), pp. 3–24.

[272] T. Ibaraki and N. Katoh. *Resource allocation problems*. Cambridge, MA: MIT Press, 1988.

[273] T. Ichimura, R. Yokoyama, and H. Magori. "A faster exact method for large-scale knapsack problems with setup costs and times". In: *International Journal of Operational Research* 14 (2012), pp. 485–504.

[274] O. Inarejos, R. Hoto, and N. Maculan. "An integer linear optimization model to the compartmentalized knapsack problem". In: *International Transactions in Operational Research* 26 (2019), pp. 1698–1717.

[275] M. Iori, A. Locatelli, and M. Locatelli. "Scheduling of Parallel Print Machines with Sequence-Dependent Setup Costs: A Real-World Case Study". In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2021, pp. 637–645.

[276] M. Iori, A. Locatelli, and M. Locatelli. "A GRASP for a real-world scheduling problem with unrelated parallel print machines and sequence-dependent setup times". In: *International Journal of Production Research* (2022), pp. 1–19.

[277] M. Iori et al. "Exact solution techniques for two-dimensional cutting and packing". In: *European Journal of Operational Research* 289 (2021), pp. 399–415.

[278] M. Iori et al. "Tool Switching Problems in the Context of Overlay Printing with Multiple Colours". In: *Combinatorial Optimization*. Springer. 2022, pp. 260–271.

[279] K. Iwama and S. Taketomi. "Removable online knapsack problems". In: *International Colloquium on Automata, Languages, and Programming*. 2002, pp. 293–305.

[280] K. Iwarna and G. Zhang. "Optimal resource augmentations for online knapsack". In: *Information Processing Letters* 110 (2010), pp. 1016–1020.

[281] M. Jalali Varnamkhasti and L. Lee. "A fuzzy genetic algorithm based on binary encoding for solving multidimensional knapsack problems". In: *Journal of Applied Mathematics* 2012 (2012), p. 703601.

[282] K. Jansen. "Parameterized approximation scheme for the multiple knapsack problem". In: *SIAM Journal on Computing* 39 (2009), pp. 1392–1412.

[283] K. Jansen and S. Kraft. "A faster FPTAS for the Unbounded Knapsack Problem". In: *European Journal of Combinatorics* 68 (2018), pp. 148–174.

[284] K. Jansen and G. Zhang. "Maximizing the total profit of rectangles packed into a rectangle". In: *Algorithmica* 47 (2007), pp. 323–342.

[285] C. Jin. "An improved FPTAS for 0-1 knapsack". In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Vol. 132. 2019, 76:1–76:14.

[286] R. Jovanovic, A. P. Sanfilippo, and S. Voß. "Fixed set search applied to the multi-objective minimum weighted vertex cover problem". In: *Journal of Heuristics* (2022).

[287] R. Jovanovic and S. Voß. "Fixed set search application for minimizing the makespan on unrelated parallel machines with sequence-dependent setup times". In: *Applied Soft Computing* 110 (2021).

[288] R. Kalaï and D. Vanderpooten. "The lexicographic α-Robust Knapsack problem". In: *International Transactions in Operational Research* 18 (2011), pp. 103–113.

[289] K. Kaparis and A. Letchford. "Local and global lifted cover inequalities for the 0-1 multidimensional knapsack problem". In: *European Journal of Operational Research* 186 (2008), pp. 91–103.

[290] K. Kaparis and A. Letchford. "Separation algorithms for 0-1 knapsack polytopes". In: *Mathematical Programming* 124 (2010), pp. 69–91.

[291] R. Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[292] A. Kasperski. *Discrete optimization with interval data*. Springer, 2008.

[293] S. Kataoka and T. Yamada. "Upper and lower bounding procedures for the multiple knapsack assignment problem". In: *European Journal of Operational Research* 237 (2014), pp. 440–447.

[294] A. Kate and I. Goldberg. "Generalizing cryptosystems based on the subset sum problem". In: *International Journal of Information Security* 10 (2011), pp. 189–199.

[295] L. Ke et al. "An ant colony optimization approach for the multidimensional knapsack problem". In: *Journal of Heuristics* 16 (2010), pp. 65–83.

[296] H. Kellerer, J.-T. Leung, and C.-L. Li. "Multiple subset sum with inclusive assignment set restrictions". In: *Naval Research Logistics* 58 (2011), pp. 546–563.

[297] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Nature Book Archives Millennium. Springer, 2004.

[298] H. Kellerer and V. Strusevich. "Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications". In: *Algorithmica* 57 (2010), pp. 769–795.

[299] H. Kellerer and V. Strusevich. "The symmetric quadratic knapsack problem: Approximation and scheduling applications". In: *4OR* 10 (2012), pp. 111–161.

[300] H. Kellerer and V. Strusevich. "Optimizing the half-product and related quadratic Boolean functions: approximation and scheduling applications". In: *Annals of Operations Research* 240 (2016), pp. 39–94.

[301] M. Khemakhem and K. Chebil. "A tree search based combination heuristic for the knapsack problem with setup". In: *Computers & Industrial Engineering* 99 (2016), pp. 280–286.

[302] A. Khutoretskii, S. Bredikhin, and A. Zamyatin. "A Lexicographic 0.5-Approximation Algorithm for the Multiple Knapsack Problem". In: *Journal of Applied and Industrial Mathematics* 12 (2018), pp. 264–277.

[303] I. Kierkosz and M. Luczak. "A hybrid evolutionary algorithm for the two-dimensional packing problem". In: *Central European Journal of Operations Research* 22 (2014), pp. 729–753.

[304] S. Kim and P. Bobrowski. "Scheduling jobs with uncertain setup times and sequence dependency". In: *Omega* 25 (1997), 437 – 447.

[305] H. Kipphan. *Handbook of print media: technologies and production methods*. Springer Science & Business Media, 2001.

[306] K. Koiliaris and C. Xu. "A faster pseudopolynomial time algorithm for subset sum". In: *ACM Transactions on Algorithms*. Vol. 15. 2019, pp. 1–20.

[307] R. Kolpakov and M. Posypkin. "On the best choice of a branching variable in the subset sum problem". In: *Discrete Mathematics and Applications* 28 (2018), pp. 29–34.

[308] R. Kolpakov, M. Posypkin, and S. Sin. "Complexity of solving the Subset Sum problem with the branch-and-bound method with domination and cardinality filtering". In: *Automation and Remote Control* 78 (2017), pp. 463–474.

[309] M. Kong, P. Tian, and Y. Kao. "A new ant colony optimization algorithm for the multidimensional Knapsack problem". In: *Computers & Operations Research* 35 (2008), pp. 2672–2683.

[310] X. Kong et al. "Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm". In: *Computers & Operations Research* 63 (2015), pp. 7–22.

[311] A. Kothari, S. Suri, and Y. Zhou. "Interval subset sum and uniform-price auction clearing". In: *International Computing and Combinatorics Conference*. Vol. 3595. 2005, pp. 608–620.

[312] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Springer Science & Business Media, 2013.

[313] S. Kovalev. "Approximation issues of fractional knapsack with penalties: a note". In: *4OR* (2021 (forthcoming)), pp. 1–8.

[314] G. Kozanidis. "Solving the linear multiple choice knapsack problem with two objectives: Profit and equity". In: *Computational Optimization and Applications* 43 (2009), pp. 261–294.

[315] G. Kozanidis and E. Melachrinoudis. "A branch and bound algorithm for the 0-1 mixed integer knapsack problem with linear multiple choice constraints". In: *Computers & Operations Research* 31 (2004), pp. 695–711.

[316] G. Kozanidis, E. Melachrinoudis, and M. Solomon. "The linear multiple choice knapsack problem with equity constraints". In: *International Journal of Operational Research* 1 (2005), pp. 52–73.

[317] A. Kramer, M. Dell'Amico, and M. Iori. "Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines". In: *European Journal of Operational Research* 275 (2019), 67 – 79.

[318] A. Kramer, M. Iori, and P. Lacomme. "Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization". In: *European Journal of Operational Research* 289 (2021), 825 – 840.

[319] M. Kuhn and K. Johnson. *Applied predictive modeling*. Springer, 2013.

[320] S. Laabadi et al. "The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches". In: *American Journal of Operations Research* 08 (2018), pp. 395–439.

[321] Y. Laalaoui and R. M'Hallah. "A binary multiple knapsack model for single machine scheduling with machine unavailability". In: *Computers & Operations Research* 72 (2016), pp. 71–82.

[322] M. Labbé and A. Violin. "Bilevel programming and price setting problems". In: *Annals of Operations Research* 240 (2016), pp. 141–169.

[323] R. Lahyani et al. "Matheuristics for solving the Multiple Knapsack Problem with Setup". In: *Computers & Industrial Engineering* 129 (2019), pp. 76–89.

[324] G. Lai, D. Yuan, and S. Yang. "A new hybrid combinatorial genetic algorithm for multidimensional knapsack problems". In: *Journal of Supercomputing* 70 (2014), pp. 930–945.

[325] X. Lai, J.-K. Hao, and D. Yue. "Two-stage solution-based Tabu search for the multidemand multidimensional knapsack problem". In: *European Journal of Operational Research* 274 (2019), pp. 35–48.

[326] M. Lalami et al. "A procedure-based heuristic for 0-1 multiple knapsack problems". In: *International Journal of Mathematics in Operational Research* 4 (2012), pp. 214–224.

[327] L. Lamanna, R. Mansini, and R. Zanotti. "A two-phase kernel search variant for the multidimensional multiple-choice knapsack problem". In: *European Journal of Operational Research* 297 (2022), pp. 53–65.

[328] Y. Lan et al. "2D knapsack: Packing squares". In: *Theoretical Computer Science* 508 (2013), pp. 35–40.

[329] G. Laporte, J. Salazar-González, and F. Semet. "Exact algorithms for the job sequencing and tool switching problem". In: *IIE Transactions* 36 (2004), pp. 37–45.

[330] A. Leão et al. "The constrained compartmentalized knapsack problem: Mathematical models and solution methods". In: *European Journal of Operational Research* 212 (2011), pp. 455–463.

[331] A. Leao et al. "Irregular packing problems: A review of mathematical models". In: *European Journal of Operational Research* 282 (2020), pp. 803–822.

[332] K. Lee, L. Lei, and M. Pinedo. "Production scheduling with history-dependent setup times". In: *Naval Research Logistics* 59.1 (2012), pp. 58–68.

[333] A. Letchford and G. Souli. "Lifting the knapsack cover inequalities for the knapsack polytope". In: *Operations Research Letters* 48 (2020), pp. 607–611.

[334] L. Létocart, A. Nagih, and G. Plateau. "Reoptimization in Lagrangian methods for the 0-1 quadratic knapsack problem". In: *Computers & Operations Research* 39 (2012), pp. 12–18.

[335] S. Leung et al. "A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem". In: *Computers & Operations Research* 39 (2012), pp. 64–73.

[336] C. Lewis-Beck and M. Lewis-Beck. *Applied regression: An introduction*. Vol. 22. Sage publications, 2015.

[337] D. Li and X. Sun. *Nonlinear Integer Programming*. New York, NY: Springer, 2006.

[338] J. Li and D. Wan. "On the subset sum problem over finite fields". In: *Finite Fields and their Applications* 14 (2008), pp. 911–929.

[339] V. Li. "Tight oscillations Tabu search for multidimensional knapsack problems with generalized upper bound constraints". In: *Computers & Operations Research* 32 (2005), pp. 2843–2852.

[340] V. Li and G. Curry. "Solving multidimensional knapsack problems with generalized upper bound constraints using critical event Tabu search". In: *Computers & Operations Research* 32 (2005), pp. 825–848.

[341] V. Li, Y.-C. Liang, and H.-F. Chang. "Solving the multidimensional knapsack problems with generalized upper bound constraints by the adaptive memory projection method". In: *Computers & Operations Research* 39 (2012), pp. 2111–2121.

[342] V. L. de Lima et al. "Arc flow formulations based on dynamic programming: Theoretical foundations and applications". In: *European Journal of Operational Research* 296 (2022), pp. 3–21.

[343] E.-H. Lin. "A Bibliographical Survey On Some Well-Known Non-Standard Knapsack Problems". In: *INFOR* 36 (1998), pp. 274–317.

[344] Q. Liu et al. "A new artificial fish swarm algorithm for the multiple knapsack problem". In: *IEICE Transactions on Information and Systems* E97-D (2014), pp. 455–468.

[345] C.-C. Lu, S.-W. Lin, and K.-C. Ying. "Robust scheduling on a single machine to minimize total flow time". In: *Computers and Operations Research* 39 (2012), 1682 – 1691.

[346] T.-C. Lu and G.-R. Yu. "An adaptive population multi-objective quantum-inspired evolutionary algorithm for multi-objective 0/1 knapsack problems". In: *Information Sciences* 243 (2013), pp. 39–56.

[347] T. Luiz, H. Santos, and E. Uchoa. "Cover by Disjoint Cliques Cuts for the Knapsack Problem with Conflicting Items". In: *Operations Research Letters* 49 (2021), pp. 844–850.

[348] R.-J. Luo, S.-F. Ji, and B.-L. Zhu. "A Pareto evolutionary algorithm based on incremental learning for a kind of multi-objective multidimensional knapsack problem". In: *Computers & Industrial Engineering* 135 (2019), pp. 537–559.

[349] T. Lust and J. Teghem. "The multiobjective multidimensional knapsack problem: a survey and a new approach". In: *International Transactions in Operational Research* 19 (2012), pp. 495–520.

[350] E. Malaguti et al. "Integer optimization with penalized fractional values: The Knapsack case". In: *European Journal of Operational Research* 273 (2019), pp. 874–888.

[351] S. Mancini, M. Ciavotta, and C. Meloni. "The Multiple Multidimensional Knapsack with Family-Split Penalties". In: *European Journal of Operational Research* 289 (2021), pp. 987–998.

[352] R. Mansi et al. "An exact algorithm for bilevel 0-1 knapsack problems". In: *Mathematical Problems in Engineering* 2012 (2012), p. 504713.

[353] R. Mansi et al. "A hybrid heuristic for the multiple choice multidimensional knapsack problem". In: *Engineering Optimization* 45 (2013), pp. 983–1004.

[354] R. Mansini and M. Speranza. "CORAL: An exact algorithm for the multidimensional knapsack problem". In: *INFORMS Journal on Computing* 24 (2012), pp. 399–415.

[355] R. Mansini and R. Zanotti. "A core-based exact algorithm for the multidimensional multiple choice knapsack problem". In: *INFORMS Journal on Computing* 32 (2020), pp. 1061–1079.

[356] F. Marques and M. Arenales. "The constrained compartmentalised knapsack problem". In: *Computers & Operations Research* 34 (2007), pp. 2109–2129.

[357] S. Martello and M. Monaci. "Models and algorithms for packing rectangles into the smallest square". In: *Computers & Operations Research* 63 (2015), pp. 161–171.

[358] S. Martello and M. Monaci. "Algorithmic approaches to the multiple knapsack assignment problem". In: *Omega* 90 (2020), p. 102004.

[359] S. Martello, D. Pisinger, and P. Toth. "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem". In: *Management Science* 45 (1999), pp. 414–424.

[360] S. Martello and P. Toth. "A Bound and Bound algorithm for the zero-one multiple knapsack problem". In: *Discrete Applied Mathematics* 3 (1981), pp. 275–288.

[361] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[362] S. Martello and P. Toth. "An exact algorithm for the two-constraint 0-1 knapsack problem". In: *Operations Research* 51 (2003), pp. 826–835.

[363] J. Martinovic and G. Scheithauer. "Integer linear programming models for the skiving stock problem". In: *European Journal of Operational Research* 251 (2016), 356 – 368.

[364] J. Martinovic, G. Scheithauer, and J. Valério de Carvalho. "A comparative study of the arcflow model and the one-cut model for one-dimensional cutting stock problems". In: *European Journal of Operational Research* 266 (2018), 458 – 471.

[365] J. Martinovic et al. "Improved flow-based formulations for the skiving stock problem". In: *Computers and Operations Research* Volume 113 (2020), p. 104770.

[366] J. Martins and B. Ribas. "A randomized heuristic repair for the multidimensional knapsack problem". In: *Optimization Letters* 15 (2021), pp. 337–355.

[367] M. Martins et al. "Hybrid multi-objective Bayesian estimation of distribution algorithm: a comparative analysis for the multi-objective knapsack problem". In: *Journal of Heuristics* 24 (2018), pp. 25–47.

[368] G. B. Mathews. "On the Partition of Numbers". In: *Proceedings of the London Mathematical Society* s1-28 (1896), pp. 486–490.

[369] G. Mavrotas, K. Florios, and J. Figueira. "An improved version of a core based algorithm for the multi-objective multi-dimensional knapsack problem: A computational study and comparison with meta-heuristics". In: *Applied Mathematics and Computation* 270 (2015), pp. 25–43.

[370] S. McCormick et al. "Primal-Dual Algorithms for Precedence Constrained Covering Problems". In: *Algorithmica* 78 (2017), pp. 771–787.

[371] L. McLay and S. Jacobson. "Integer knapsack problems with set-up weights". In: *Computational Optimization and Applications* 37 (2007), pp. 35–47.

[372] L. McLay and S. Jacobson. "Algorithms for the bounded set-up knapsack problem". In: *Discrete Optimization* 4 (2007), pp. 206–212.

[373] S. Michel, N. Perrot, and F. Vanderbeck. "Knapsack problems with setups". In: *European Journal of Operational Research* 196 (2009), pp. 909–918.

[374] L. Mingo López, N. Gómez Blas, and A. Arteta Albert. "Multidimensional knapsack problem optimization using a binary particle swarm model with genetic operations". In: *Soft Computing* 22 (2018), pp. 2567–2582.

[375] M. Monaci and U. Pferschy. "On the Robust Knapsack Problem". In: *SIAM Journal on Optimization* 23 (2013), pp. 1956–1982.

[376] M. Monaci, U. Pferschy, and P. Serafini. "Exact solution of the robust knapsack problem". In: *Computers & Operations Research* 40 (2013), pp. 2625–2631.

[377] D. Montgomery, E. Peck, and G. Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[378] R. Moraga, G. Depuy, and G. Whitehouse. "Meta-RaPS approach for the 0-1 multidimensional knapsack problem". In: *Computers & Industrial Engineering* 48 (2005), pp. 83–96.

[379] F. Morales and J. Martínez. "Analysis of Divide-and-Conquer strategies for the 0-1 minimization knapsack problem". In: *Journal of Combinatorial Optimization* 40 (2020), pp. 234–278.

[380] H. E. Morales D. R.and Romeijn. "The Generalized Assignment Problem and Extensions". In: *Handbook of Combinatorial Optimization: Supplement Volume B*. Ed. by D.-Z. Du and P. M. Pardalos. Boston: Springer, 2005, pp. 259–311.

[381] H. Murata et al. "VLSI module placement based on rectangle-packing by the sequence-pair". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15 (1996), pp. 1518–1524.

[382] M. Naderi-Beni et al. "Fuzzy bi-objective formulation for a parallel machine scheduling problem with machine eligibility restrictions and sequence-dependent setup times". In: *International Journal of Production Research* 52 (2014), 5799 – 5822.

[383] A. Natekin and A. Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in Neurorobotics* 7 (2013).

[384] A. Navarra and C. Pinotti. "Online knapsack of unknown capacity: How to optimize energy consumption in smartphones". In: *Theoretical Computer Science* 697 (2017), pp. 98–109.

[385] G. L. Nemhauser and L. A. Wolsey. *Integer programming and combinatorial optimization*. Vol. 191. Springer, 1988.

[386] G. Nicosia, A. Pacifici, and U. Pferschy. "Price of Fairness for allocating a bounded resource". In: *European Journal of Operational Research* 257 (2017), pp. 933–943.

[387] K. Nip and Z. Wang. "On the approximability of the two-phase knapsack problem". In: *Journal of Combinatorial Optimization* 38 (2019), pp. 1155–1179.

[388] P. Olivier, A. Lodi, and G. Pesant. "The Quadratic Multiknapsack Problem with Conflicts and Balance Constraints". In: *INFORMS Journal on Computing* 33 (2021), pp. 949–962.

[389] T. Öncan. "A Survey of the Generalized Assignment Problem and Its Applications". In: *INFOR: Information Systems and Operational Research* 45 (2007), pp. 123–141.

[390] F. A. Paine and H. Y. Paine. *A handbook of food packaging*. Boston, MA, USA: Springer, 2012.

[391] Y. Pan and F. Zhang. "Solving low-density multiple subset sum problems with SVP oracle". In: *Journal of Systems Science and Complexity* 29 (2016), pp. 228–242.

[392] C. Patvardhan, S. Bansal, and A. Srivastav. "Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems". In: *Swarm and Evolutionary Computation* 26 (2016), pp. 175–190.

[393] C. Patvardhan, V. Prakash, and A. Srivastav. "Novel quantum-inspired evolutionary algorithms for the quadratic knapsack problem". In: *International Journal of Mathematics in Operational Research* 4 (2012), pp. 114–127.

[394] D. Pearson. "A polynomial-time algorithm for the change-making problem". In: *Operations Research Letters* 33 (2005), pp. 231–234.

[395] B. Peng et al. "An ejection chain approach for the quadratic multiple knapsack problem". In: *European Journal of Operational Research* 253 (2016), pp. 328–336.

[396] U. Pferschy, G. Nicosia, and A. Pacifici. "A Stackelberg knapsack game with weight control". In: *Theoretical Computer Science* 799 (2019), pp. 149–159.

[397] U. Pferschy and R. Scatamacchia. "Improved dynamic programming and approximation results for the knapsack problem with setups". In: *International Transactions in Operational Research* 25 (2018), pp. 667–682.

[398] U. Pferschy and J. Schauer. "The Knapsack Problem with Conflict Graphs". In: *Journal of Graph Algorithms and Applications* 13 (2009), pp. 233–249.

[399] U. Pferschy and J. Schauer. "Approximation of the quadratic knapsack problem". In: *INFORMS Journal on Computing* 28 (2016), pp. 308–318.

[400] U. Pferschy and J. Schauer. "Approximation of knapsack problems with conflict and forcing graphs". In: *Journal of Combinatorial Optimization* 33 (2017), pp. 1300–1323.

[401] U. Pferschy, J. Schauer, and C. Thielen. "Approximating the product knapsack problem". In: *Optimization Letters* 15 (2021), pp. 2529–2540.

[402] U. Pferschy et al. "On the Stackelberg knapsack game". In: *European Journal of Operational Research* 291 (2021), pp. 18–31.

[403] T. Pinto et al. "Solving the multiscenario max-min knapsack problem exactly with column generation and branch-and-bound". In: *Mathematical Problems in Engineering* 2015 (2015), p. 439609.

[404] D. Pisinger. "Where are the hard knapsack problems?" In: *Computers & Operations Research* 32 (2005), pp. 2271–2284.

[405] D. Pisinger. "The quadratic knapsack problem-a survey". In: *Discrete Applied Mathematics* 155 (2007), pp. 623–648.

[406] D. Pisinger and A. Saidi. "Tolerance analysis for 0-1 knapsack problems". In: *European Journal of Operational Research* 258 (2017), pp. 866–876.

[407] W. Pisinger, A. Rasmussen, and R. Sandvik. "Solution of large quadratic knapsack problems through aggressive reduction". In: *INFORMS Journal on Computing* 19 (2007), pp. 280–290.

[408] V. Poirriez, N. Yanev, and R. Andonov. "A hybrid algorithm for the unbounded knapsack problem". In: *Discrete Optimization* 6 (2009), pp. 110–124.

[409] C. Privault and G. Finke. "Modelling a tool switching problem on a single NC-machine". In: *Journal of Intelligent Manufacturing* 6.2 (1995), pp. 87–94.

[410] J. Puchinger, G. Raidl, and U. Pferschy. "The multidimensional knapsack problem: Structure and algorithms". In: *INFORMS Journal on Computing* 22 (2010), pp. 250–265.

[411] J. Qin et al. "Hybridization of Tabu search with feasible and infeasible local searches for the quadratic multiple knapsack problem". In: *Computers & Operations Research* 66 (2016), pp. 199–214.

[412] X. Qiu and W. Kern. "Improved approximation algorithms for a bilevel knapsack problem". In: *Theoretical Computer Science* 595 (2015), pp. 120–129.

[413] D. Quadri, E. Soutif, and P. Tolla. "Exact solution method to solve large scale integer quadratic multidimensional knapsack problems". In: *Journal of Combinatorial Optimization* 17 (2009), pp. 157–167.

[414] Z. Quan and L. Wu. "Cooperative parallel adaptive neighbourhood search for the disjunctively constrained knapsack problem". In: *Engineering Optimization* 49 (2017), pp. 1541–1557.

[415] T. de Queiroz et al. "Algorithms for 3D guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing". In: *Computers & Operations Research* 39 (2012), pp. 200–212.

[416] T. de Queiroz et al. "Two-dimensional Disjunctively Constrained Knapsack Problem: Heuristic and exact approaches". In: *Computers & Industrial Engineering* 105 (2017), pp. 313–328.

[417] J. Quiroga-Orozco, J. Valério de Carvalho, and R. V. Hoto. "A strong integer linear optimization model to the compartmentalized knapsack problem". In: *International Transactions in Operational Research* 26 (2019), pp. 1633–1654.

[418] P. Refaeilzadeh, L. Tang, and H. Liu. "Cross-Validation". In: *Encyclopedia of Database Systems*. Boston: Springer, 2009, pp. 532–538.

[419] M. G. C. Resende and C. C. Ribeiro. "Greedy randomized adaptive search procedures: Advances and extensions". In: *Handbook of metaheuristics*. Springer, 2019, pp. 169–220.

[420] C. Rodrigues et al. "0-1 quadratic knapsack problems: An exact approach based on A t-Linearization". In: *SIAM Journal on Optimization* 22 (2012), pp. 1449–1468.

[421] P. Rohlfshagen and X. Yao. "Dynamic combinatorial optimisation problems: An analysis of the subset sum problem". In: *Soft Computing* 15 (2011), pp. 1723–1734.

[422] A. Rong and J. Figueira. "Computational performance of basic state reduction based dynamic programming algorithms for bi-objective 0-1 knapsack problems". In: *Computers and Mathematics with Applications* 63 (2012), pp. 1462–1480.

[423] A. Rong and J. Figueira. "Dynamic programming algorithms for the bi-objective integer knapsack problem". In: *European Journal of Operational Research* 236 (2014), pp. 85–99.

[424] A. Rong, J. Figueira, and K. Klamroth. "Dynamic programming based algorithms for the discounted 0-1 knapsack problem". In: *Applied Mathematics and Computation* 218 (2012), pp. 6921–6933.

[425] A. Rong, J. Figueira, and M. Pato. "A two state reduction based dynamic programming algorithm for the bi-objective 01 knapsack problem". In: *Computers and Mathematics with Applications* 62 (2011), pp. 2913–2930.

[426] A. Rong, K. Klamroth, and J. Figueira. "Multicriteria 0-1 knapsack problems with *k*-min objectives". In: *Computers & Operations Research* 40 (2013), pp. 1481–1496.

[427] S. Sabet, M. Shokouhifar, and F. Farokhi. "A discrete artificial bee colony for multiple knapsack problem". In: *International Journal of Reasoning-based Intelligent Systems* 5 (2013), pp. 88–95.

[428] I. Sabuncuoglu and S. Goren. "Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research". In: *International Journal of Computer Integrated Manufacturing* 22 (2009), 138 – 157.

[429] R. Sadykov and F. Vanderbeck. "Bin packing with conflicts: a generic branch-and-price algorithm". In: *INFORMS Journal on Computing* 25 (2013), pp. 244–255.

[430] M. Salem et al. "Probabilistic Tabu search with multiple neighborhoods for the Disjunctively Constrained Knapsack Problem". In: *RAIRO - Operations Research* 51 (2017), pp. 627–637.

[431] M. Salem et al. "Optimization algorithms for the disjunctively constrained knapsack problem". In: *Soft Computing* 22 (2018), pp. 2025–2043.

[432] M. Samavati et al. "A methodology for the large-scale multi-period precedence-constrained knapsack problem: an application in the mining industry". In: *International Journal of Production Economics* 193 (2017), pp. 12–20.

[433] C. Sanches, N. Soma, and H. Yanasse. "An optimal and scalable parallelization of the two-list algorithm for the subset-sum problem". In: *European Journal of Operational Research* 176 (2007), pp. 870–879.

[434] C. Sanches, N. Soma, and H. Yanasse. "Parallel time and space upper-bounds for the subset-sum problem". In: *Theoretical Computer Science* 407 (2008), pp. 342–348.

[435] T. Saraç and A. Sipahioglu. "Generalized quadratic multiple knapsack problem and two solution approaches". In: *Computers & Operations Research* 43 (2014), pp. 78–89.

[436] A. Sbihi. "A best first search exact algorithm for the Multiple-choice Multidimensional Knapsack Problem". In: *Journal of Combinatorial Optimization* 13 (2007), pp. 337–351.

[437] A. Sbihi. "A cooperative local search-based algorithm for the Multiple-Scenario Max-Min Knapsack Problem". In: *European Journal of Operational Research* 202 (2010), pp. 339–346.

[438] A. Sbihi. "Adaptive perturbed neighbourhood search for the expanding capacity multiple-choice knapsack problem". In: *Journal of the Operational Research Society* 64 (2013), pp. 1461–1473.

[439] L. Schäfer et al. "The binary knapsack problem with qualitative levels". In: *European Journal of Operational Research* 289 (2021), pp. 508–514.

[440] J. Schauer. "Asymptotic behavior of the quadratic knapsack problem". In: *European Journal of Operational Research* 255 (2016), pp. 357–363.

[441] B. Schulze et al. "On the rectangular knapsack problem: approximation of a specific quadratic knapsack problem". In: *Mathematical Methods of Operations Research* 92 (2020), pp. 107–132.

[442] J. Schuurman and J. Van Vuuren. "Scheduling sequence-dependent colour printing jobs". In: *South African Journal of Industrial Engineering* 27.2 (2016), pp. 43–59.

[443] N. Schwertman and R. de Silva. "Identifying outliers with sequential fences". In: *Computational Statistics and Data Analysis* 51 (2007), 3800 – 3810.

[444] T. Setzer and S. Blanc. "Empirical orthogonal constraint generation for Multidimensional 0/1 Knapsack Problems". In: *European Journal of Operational Research* 282 (2020), pp. 58–70.

[445] R. Shah and P. Reed. "Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems". In: *European Journal of Operational Research* 211 (2011), pp. 466–479.

[446] H. Shah-Hosseini. "Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem". In: *International Journal of Intelligent Computing and Cybernetics* 1 (2008), pp. 193–212.

[447] A. Shahriar et al. "A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem". In: *Journal of Supercomputing* 43 (2008), pp. 257–280.

[448] X. Shi, L. Wu, and X. Meng. "A new optimization model for the sustainable development: Quadratic knapsack problem with conflict graphs". In: *Sustainability* 9 (2017), pp. 1–10.

[449] K. Shiangjen et al. "An iterative bidirectional heuristic placement algorithm for solving the two-dimensional knapsack packing problem". In: *Engineering Optimization* 50 (2018), pp. 347–365.

[450] H. Shojaei et al. "A fast and scalable multidimensional multiple-choice knapsack heuristic". In: *ACM Transactions on Design Automation of Electronic Systems* 18 (2013), pp. 1–32.

[451] E. Silva, T. Toffolo, and T. Wauters. "Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems". In: *Computers & Operations Research* 109 (2019), pp. 12–27.

[452] J. da Silveira, E. Xavier, and F. Miyazawa. "A note on a two dimensional knapsack problem with unloading constraints". In: *RAIRO - Theoretical Informatics and Applications* 47 (2013), pp. 315–324.

[453] J. Simon, A. Apte, and E. Regnier. "An application of the multiple knapsack problem: The self-sufficient marine". In: *European Journal of Operational Research* 256 (2017), pp. 868–876.

[454] S. Sitarz. "Multiple criteria dynamic programming and multiple knapsack problem". In: *Applied Mathematics and Computation* 228 (2014), pp. 598–605.

[455] S. Skiena. "Who is interested in algorithms and why?: lessons from the stony brook algorithms repository". In: *ACM SIGACT News* 30 (1999), pp. 65–74.

[456] K. Smith-Miles, J. Christiansen, and M. Muñoz. "Revisiting where are the hard knapsack problems? via Instance Space Analysis". In: *Computers & Operations Research* 128 (2021), p. 105184.

[457] L. Soares and M. Carvalho. "Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints". In: *European Journal of Operational Research* 285 (2020), 955 – 964.

[458] S. Stefanov. "On the solution of multidimensional convex separable continuous knapsack problem with bounded variables". In: *European Journal of Operational Research* 247 (2015), pp. 366–369.

[459] A. Tamir. "New pseudopolynomial complexity bounds for the bounded and other integer Knapsack related problems". In: *Operations Research Letters* 37 (2009), pp. 303–306.

[460] C. S. Tang and E. V. Denardo. "Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches". In: *Operations Research* 36.5 (1988), pp. 767–777.

[461] F. Taniguchi, T. Yamada, and S. Kataoka. "Heuristic and exact algorithms for the max-min optimization of the multi-scenario knapsack problem". In: *Computers & Operations Research* 35 (2008), pp. 2034–2048.

[462] F. Taniguchi, T. Yamada, and S. Kataoka. "A virtual pegging approach to the max-min optimization of the bi-criteria knapsack problem". In: *International Journal of Computer Mathematics* 86 (2009), pp. 779–793.

[463] R. Taylor. "Approximation of the Quadratic Knapsack Problem". In: *Operations Research Letters* 44 (2016), pp. 495–497.

[464] C. Thielen, M. Tiedemann, and S. Westphal. "The online knapsack problem with incremental capacity". In: *Mathematical Methods of Operations Research* 83 (2016), pp. 207–242.

[465] B. Thiongane, A. Nagih, and G. Plateau. "Lagrangean heuristics combined with reoptimization for the 0-1 bidimensional knapsack problem". In: *Discrete Applied Mathematics* 154 (2006), pp. 2200–2211.

[466] S. Toumi, M. Cheikh, and B. Jarboui. "0-1 quadratic knapsack problem solved with VNS algorithm". In: *Electronic Notes in Discrete Mathematics* 47 (2015), pp. 269–276.

[467] M. Tzur and A. Altman. "Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes". In: *IIE Transactions* 36 (2004), 95 – 110.

[468] A. Ünal and G. Kayakutlu. "A Partheno-Genetic Algorithm for Dynamic 0-1 Multidimensional Knapsack Problem". In: *RAIRO - Operations Research* 50 (2016), pp. 47–66.

[469] N. Van Hop. "The tool-switching problem with magazine capacity and tool size constraints". In: *IEEE Transactions on Systems, Man, and Cybernetics* 35 (2005), 617 – 628.

[470] M. Vasquez and Y. Vimont. "Improved results on the 0-1 multidimensional knapsack problem". In: *European Journal of Operational Research* 165 (2005), pp. 70–81.

[471] Y. Vimont, S. Boussier, and M. Vasquez. "Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem". In: *Journal of Combinatorial Optimization* 15 (2008), pp. 165–178.

[472] S. Voß and E. Lalla-Ruiz. "A set partitioning reformulation for the multiple-choice multidimensional knapsack problem". In: *Engineering Optimization* 48 (2016), pp. 831–850.

[473] H. Wang, G. Kochenberger, and F. Glover. "A computational study on the quadratic knapsack problem with multiple constraints". In: *Computers & Operations Research* 39 (2012), pp. 3–11.

[474] H. Wang, G. Kochenberger, and Y. Xu. "A note on optimal solutions to quadratic knapsack problems". In: *International Journal of Mathematical Modelling and Numerical Optimisation* 1 (2010), pp. 344–351.

[475] L. Wang, S.-Y. Wang, and Y. Xu. "An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem". In: *Expert Systems with Applications* 39 (2012), pp. 5593–5599.

[476] S. Wang et al. "The interval min-max regret knapsack packing-delivery problem". In: *International Journal of Production Research* 59 (2021), pp. 5661–5677.

[477] W. Wang and J. Nguyen. "The *k*-subset sum problem over finite fields". In: *Finite Fields and their Applications* 51 (2018), pp. 204–217.

[478] Z. Wang and W. Xing. "A successive approximation algorithm for the multiple knapsack problem". In: *Journal of Combinatorial Optimization* 17 (2009), pp. 347–366.

[479] X.-H. Wei and K. Zhang. "Discrete artificial bee colony algorithm for multiple knapsack problems". In: *International Journal of Advancements in Computing Technology* 4 (2012), pp. 484–490.

[480] C. Wilbaut, S. Hanafi, and S. Salhi. "A survey of effective heuristics and their application to a variety of knapsack problems". In: *IMA Journal of Management Mathematics* 19 (2008), pp. 227–244.

[481] C. Wilbaut, S. Salhi, and S. Hanafi. "An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem". In: *European Journal of Operational Research* 199 (2009), pp. 339–348.

[482] S. Windras Mara et al. "The job sequencing and tool switching problem with sequence-dependent setup time". In: *Journal of King Saud University - Engineering Sciences* (2021).

[483] J. Wu and T. Srikanthan. "An efficient algorithm for the collapsing knapsack problem". In: *Information Sciences* 176 (2006), pp. 1739–1751.

[484] W. Wu, Y. Mutsunori, and I. Toshihide. "Generalized Assignment Problem". In: *Handbook of Approximation Algorithms and Metaheuristics, Second Edition*. Ed. by F. G. Teofilo. CRC Press, 2018, pp. 713–736.

[485] W. Wu et al. "An iterated dual substitution approach for the min-max regret multidimensional knapsack problem". In: *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE. 2016, pp. 726–730.

[486] W. Wu et al. "Exact and heuristic algorithms for the interval min-max regret generalized assignment problem". In: *Computers & Industrial Engineering* 125 (2018), pp. 98–110.

[487] Z. Wu, B. Jiang, and H. Karimi. "A logarithmic descent direction algorithm for the quadratic knapsack problem". In: *Applied Mathematics and Computation* 369 (2020), p. 124854.

[488] E. Xavier and F. Miyazawa. "Approximation schemes for knapsack problems with shelf divisions". In: *Theoretical Computer Science* 352 (2006), pp. 71–84.

[489] X. F. Xie and J. Liu. "A mini-swarm for the quadratic knapsack problem". In: *2007 IEEE Swarm Intelligence Symposium*. IEEE. Honolulu, HI, 2007, pp. 190–197.

[490] Z. Xu. "A strongly polynomial FPTAS for the symmetric quadratic knapsack problem". In: *European Journal of Operational Research* 218 (2012), pp. 377–381.

[491] Z. Xu and X. Lai. "A fully polynomial approximation scheme for a knapsack problem with a minimum filling constraint". In: *Workshop on Algorithms and Data Structures*. 2011, pp. 704–715.

[492] T. Yamada and T. Takeoka. "An exact algorithm for the fixed-charge multiple knapsack problem". In: *European Journal of Operational Research* 192 (2009), pp. 700–705.

[493] Y. Yang, N. Boland, and M. Savelsbergh. "Multivariable branching: A 0-1 knapsack problem case study". In: *INFORMS Journal on Computing* (2021 (forthcoming)).

[494] Y. Yang and R. Bulfin. "An exact algorithm for the knapsack problem with setup". In: *International Journal of Operational Research* 5 (2009), pp. 280–291.

[495] Z. Yang, G. Wang, and F. Chu. "An effective GRASP and Tabu search for the 0-1 quadratic knapsack problem". In: *Computers & Operations Research* 40 (2013), pp. 1176–1185.

[496] Y. Ye and A. Borodin. "Priority algorithms for the subset-sum problem". In: *Journal of Combinatorial Optimization* 16 (2008), pp. 198–228.

[497] A. D. Yimer and K. Demirli. "Fuzzy scheduling of job orders in a two-stage flowshop with batch-processing machines". In: *International Journal of Approximate Reasoning* 50 (2009), 117 – 137.

[498] Y. Yoon, Y.-H. Kim, and B.-R. Moon. "A theoretical and empirical investigation on the Lagrangian capacities of the 0-1 multidimensional knapsack problem". In: *European Journal of Operational Research* 218 (2012), pp. 366–376.

[499] B. You and T. Yamada. "A pegging approach to the precedence-constrained knapsack problem". In: *European Journal of Operational Research* 183 (2007), pp. 618–632.

[500] B. You and T. Yamada. "An exact algorithm for the budget-constrained multiple knapsack problem". In: *International Journal of Computer Mathematics* 88 (2011), pp. 3380–3393.

[501] G. Yu. "On the max-min 0-1 knapsack problem with robust optimization applications". In: *Operations Research* 44 (1996), pp. 407–415.

[502] P. Yunusoglu and S. Topaloglu Yildiz. "Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times". In: *International Journal of Production Research* 60 (2022), 2212 – 2229.

[503] M. Zennaki. "A New Hybrid Algorithm for the Multiple-Choice Multi-Dimensional Knapsack Problem". In: *WSEAS Transactions on Information Science and Applications* 10 (2013), pp. 219–229.

[504] T. Zhong and R. Young. "Multiple Choice Knapsack Problem: Example of planning choice in transportation". In: *Evaluation and Program Planning* 33 (2010), pp. 128–137.

[505] Q. Zhou, J.-K. Hao, and Q. Wu. "A hybrid evolutionary search for the generalized quadratic multiple knapsack problem". In: *European Journal of Operational Research* 296 (2022), pp. 788–803.

[506] S. Zhou et al. "Two-dimensional knapsack-block packing problem". In: *Applied Mathematical Modelling* 73 (2019), pp. 1–18.

[507] D. Zouache, A. Moussaoui, and F. Ben Abdelaziz. "A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem". In: *European Journal of Operational Research* 264 (2018), pp. 74–88.