Alma Mater Studiorum - Università di Bologna

Dottorato di Ricerca in
Automatica e Ricerca Operativa

Ciclo XXIX
Settore concorsuale di afferenza: 01/A6 - RICERCA OPERATIVA
Settore scientifico disciplinare: MAT/09 - RICERCA OPERATIVA

Mathematical Models and Decomposition Algorithms for Cutting and Packing Problems

Presentata da Maxence Delorme

Coordinatore Dottorato	**Relatore**
Prof. Daniele Vigo	Prof. Silvano Martello
	Co-relatore
	Prof. Manuel Iori

Esame finale anno 2017

Contents

Acknowledgments

First of all, I would like to thank my tutors Prof. Silvano Martello and Prof. Manuel Iori for the help they provided during three years of Ph.D. studies, for giving me the opportunity to work on many different and interesting research projects, and for teaching me a lot about Operations Research and Combinatorial Optimization.

I would also like to thank all the other components of the group of Operations Research of the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI) of the University of Bologna, namely Prof. Paolo Toth, Prof. Daniele Vigo, Prof. Andrea Lodi, Prof. Enrico Malaguti, Prof. Michele Monaci, Dr. Valentina Cacchiani, Dr. Paolo Tubertini, Ph.D. students Carlos Emilio Contreras Bolton, Alberto Santini, and the former members Dr. Claudo Gambella, Dr. Tiziano Parriani, Dr. Dimitri Thomopulos, and Dr. Sven Wiese.

I am grateful to my co-authors, who helped me in writing the reports that form part of this thesis: Prof. Jean François Côté from the Université Laval, and Prof. Anand Subramanian from the Federal University of Paraíba. Thanks are also due to the undergraduate and master students I collaborated with during these years, in particular Gianluca Costa and Vitor Nesello.

I would also like to thank the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) for the financial support given to my Ph.D. course

Special thanks are addressed to Prof. François Clautiaux, Prof. Joshua D. Habiger, and Prof. Alexandru-Adrian Tantar for encouraging me to pursue a Ph.D. and to Prof. Clarisse Dhaenens, who made me discover and enjoy Combinatorial Optimization.

Finally, I would like to thank my friends Damien and Julien, my mother, my father, and all my family for always supporting me in my decisions.

Bologna, March 27, 2017

Maxence Delorme

List of Figures

List of Tables

List of Acronyms

Acronym	Meaning
BFD	Best-Fit Decreasing
BPP	Bin Packing Problem
BPPIF	Bin Packing Problem with Item Fragmentation
CBP	Contiguous Bin-Packing Problem
C&PP	Cutting and Packing Problems
CP	Constraint Programming
CSP	Cutting Stock Problem
DDT	Disaggregated Discrete-Time
DP	Dynamic Programming
DT	Discrete-time
DTCTP	Discrete Time-Cost Tradeoff Problem
FFD	First-Fit Decreasing
ILP	Integer Linear Programming
IRUP	Integer Round-Up Property
LB	Lower Bound
LP	Linear Programming
MILP	Mixed Integer Linear Programming
MIRUP	Modified Integer Round-Up Property
MMRCPSP	Multi-Mode Resource-Constrained Project Scheduling Problem
PLP	Pallet Loading Problem
PRPP	Plastic Rolls Production Problem
PRSR	Packing Rectangles into a Square with Rotation
PSS	Packing Squares into a Square
RCPSP	Resource-Constrained Project Scheduling Problem
SCP	orthogonal Stock Cutting Problem
SPP	Strip Packing Problem
UB	Upper Bound
VSBPP	Variable-Sized Bin Packing Problem
WCPR	Worst-Case Performance Ratio

Chapter 1

Introduction

Many real world optimization problems we face nowadays belong to the category of *cutting and packing problems* (C&PP). Wether one wants to pack a set of goods in a container, cut a set of small items from large pieces of wood, position articles in a newspaper, or even play the famous Tetris video game, they solve a C&PP. In addition, C&PP enter as a component of an incredible amount of more complex problems, such as routing problems with capacity and scheduling problems with resources constraints. It is therefore not surprising that researchers have focused on developing effective methods to deal with C&PP. Figure 1.1 shows the number of articles having in the title either the term bin packing, or the term cutting stock, two subclasses of C&PP, according to different bibliographic data bases, in the years 1991-2016. The picture shows the growing interest in these specific problems, with sharp increase in recent years.

As most of the problems from the C&PP family are $\mathcal{NP}$-hard, see Garey and Johnson [130], they are very difficult to solve to optimality in practice and many exact approaches have been proposed in the literature, including branch-and-bound and branch-and-cut algorithms.

With the recent improvements of *mixed integer linear programming* (MILP) solvers (see, e.g., Achterberg and Wunderling [1] and Lodi [192]), an alternative axe of research consists in finding better MILP formulations for the problem. For example, through the last decades, no less than six different MILP models have been proposed for the classical bin packing problem: in chronological order, the textbook model by Martello and Toth [214] derived from the work of Kantorovich [167], the set covering formulation by Gilmore and Gomory [134, 135], the one-cut by Dyckoff [111], the arc-flow by Valério de Carvalho [278], the DP-flow by Cambazard and O'Sullivan [52], and the general arc-flow with graph compression by Brandão and Pedroso [45]. A seventh, more powerful, formulation is even proposed in this thesis (Chapter 4). Vanderbeck and Wolsey [287] described in details generic procedures to obtain reformulations, and categorized them in several categories. In the *extended formulations*, new variables are introduced so as to better model the structure

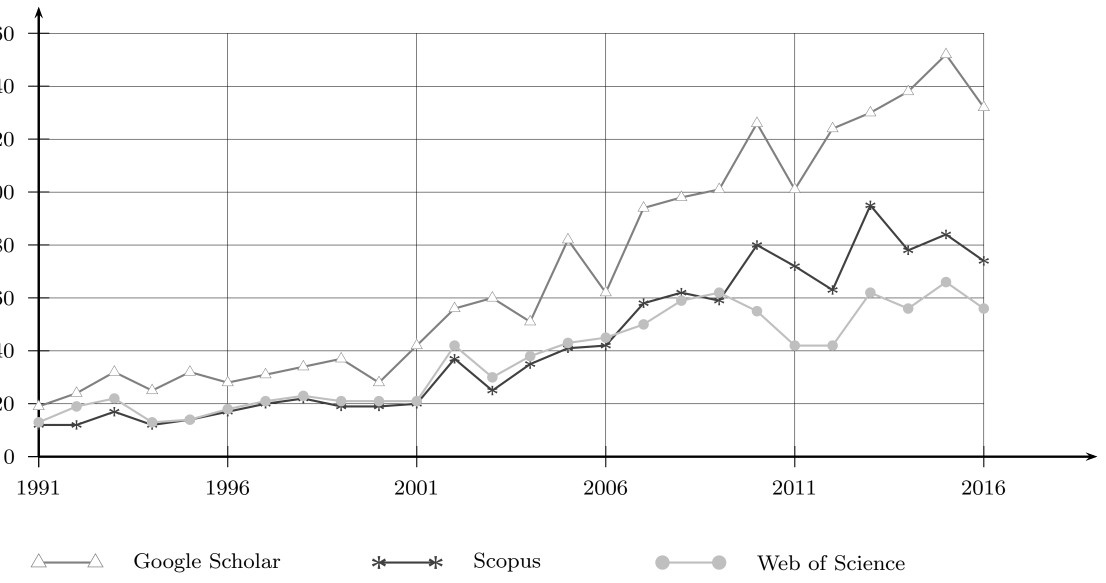


Figure 1.1: Number of papers dealing with bin packing and cutting stock problems, 1991-2016

of the problem. These new variables typically allow one to model some combinatorial structure more precisely and to induce integrality through tighter linear constraints linking the variables. The reformulations that use *projection* allow one to reduce the number of variables so that calculations are typically faster. Some other reformulations are just *alternative formulations* that aim at treating or eliminating symmetry among solutions or obtaining variables that are more effective as branching variables, or variables for which one can develop effective valid inequalities.

When the problem is very complex, which is often the case when C&PP involve two or three dimensions with non overlapping restrictions, MILP models usually struggle to find optimal solutions, as the number of variables and constraints they involve are too high. Therefore, other tools such as decomposition methods, e.g., Benders' decomposition [34], are used. Hooker and Ottosson[157] described the Benders' decomposition as a method that begins by partitioning the variables of a problem into two vectors x and y. It fixes y to a trial value in the master problem so as to define a slave problem that only contains x. If the solution of the slave reveals that the trial value of y is unacceptable, the slave's dual is used to identify a number of other values of y that are likewise unacceptable and are removed from the master problem by means of cuts. The next trial value must be one that has not been excluded. Eventually only acceptable values remain, and if all goes well, the algorithm terminates after enumerating only a few of the possible values of y. In the classical Benders' decomposition, the master is an MILP and the slave is a *linear programming* (LP) model. Hooker and Ottosson[157] proposed a similar decomposition called logic-based Benders decomposition, in which both the master and the subproblem are MILPs. They solved the slave problem by logical deduction methods, such as constraint programming, whose outcome was used to produce valid cuts. Côté et al. [83] also used Benders' decomposition where both subproblems were MILPs solved by combinatorial algorithm. To produce effective valid cuts, they used the combinatorial Benders' cuts introduced by Codato and Fischetti [69], in which a third combinatorial algorithm is used to find the cuts that are added to the master. Note that modern implementation of Benders' cuts involve branch-and-cut algorithms (see, e.g., Padberg and Rinaldi [228] for a description of branch-and-cut algorithms).

An alternative way to handle MILP models with too many variables is to use Dantzig-Wolfe decomposition [89]. As described by Vanderbeck [284], DantzigWolfe decomposition is a specific form of problem reformulation that aims at providing a tighter linear programming relaxation bound. The reformulation gives rise to an integer master problem, whose typically large number of variables is dealt by using an integer programming column generation procedure. Vance [281] showed that, when applied to the textbook model of Martello and Toth [214], the Dantzig-Wolfe decomposition leads to the set covering formulation of Gilmore and Gomory [134, 135]. Dantzig-Wolfe decomposition often leads to branch-and-price algorithms. As described in Barnhart et al. [23], in a branch-and-price algorithm, sets of columns are left out of the LP relaxation because there are too many columns to handle efficiently and most of them will have their associated variable equal to zero in an optimal solution anyway. Then to check the optimality of an LP solution, a subproblem, called the pricing problem, which is a separation problem for the dual LP, is solved to try to identify columns to enter the basis. If such columns are found, the LP is reoptimized. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality conditions. Branch-and-price allows column generation to be applied throughout the branch-and-bound tree.

The general scope of this thesis is to review or provide new and effective algorithms based on alternative MILP models and/or decomposition approaches to solve exactly various cutting and packing problems. Each chapter of the thesis is self-contained and can be read independently of the others. A unique bibliography is provided at the end of the thesis, to avoid the repetition of references.

In Chapter 2, we propose a survey on the classical bin packing and cutting stock problems. After a detailed review of the literature (over 150 references), we implement and computationally test the most common methods used to solve the problems, including branch-and-price, constraint programming and mixed integer programming, and we successfully propose new sets of instances that are difficult to solve in practice.

In Chapter 3, we describe the BPPLIB, a library for bin packing and cutting stock problems. We gather the most important results from Chapter 2, test some of the algorithms with free solvers, and make their code publicly available. We also make additional experimental results on new sets of instances and introduce BppGame, an interactive visual solver for the bin packing problem.

In Chapter 4, we study in details the main pattern-based and pseudo-polynomial MILP formulations that have been proposed for the bin packing problems and we provide a clear picture of the dominance and equivalence relations that exist among them. In addition, we introduce a new MILP formulation for the problem and show its effectiveness through some tests on benchmark instances, achieving state of the art results and finding several new proven optimal solutions. We also show how to adapt the formulation to the variable-sized bin packing problem and the bin packing problem with item fragmentation, and obtain results that consistently improve those available in the literature.

In Chapter 5, we propose a method based on Logic based Benders' decomposition for the orthogonal stock cutting problem and some extensions. We solve the master problem through an MILP model while constraint programming is used to solve the slave problem. The resulting method is hybridized with a state-of-the-art branch-and-bound algorithm and computational experiments on classical benchmarks from the literature show the effectiveness of the proposed approach.

In Chapter 6, we compare human performances with respect to simple heuristics and exact approaches on two-dimensional packing problems. After introducing TwoBinGame, a visual application we developed for students to interactively solve two-dimensional packing problems, we detail the experimental plan we adopted to measure human efficiency when various parameters of the test instances (e.g., the number of items to pack or the possibility of rotation) change. We analyze the results obtained by about 200 students and show that the human brain is able to obtain, for relatively small instances, results comparable or better than those produced by simple heuristics such as bottom left or best fit, even when coupled with powerful post-processing.

In Chapter 7, we study an optimization problem that originates from the packaging industry, in particular from the process of blown film extrusion, where a plastic film is used to produce rolls of different dimensions and colors. The film can be cut along its width, thus producing multiple rolls in parallel, and set-up times must be considered when changing from one color to another. The optimization problem that we face is to produce a given set of rolls on a number of identical parallel machines by minimizing the makespan. The problem combines cutting and scheduling decisions, and is very difficult to solve exactly. For its solution, we propose mathematical models and heuristic algorithms that involve a non-trivial decomposition method. By means of extensive computational experiments we show that proven optimality can be achieved on small instances, whereas for larger instances good quality solutions can be obtained especially by the use of an iterated local search algorithm.

In Chapter 8, we study the time-indexed formulations of the resource constrained project scheduling problem and propose some improvements based on preprocessing and lifting techniques. Then, we study the discrete time-cost tradeoff problem and introduce a new MILP model to solve the problem. Finally, we study the multi-mode resource-constrained project scheduling problem and propose a hybridized algorithm. For each of these problem, we compare the new algorithm we propose with a classical MILP formulation and a constraint programming approach from the literature.

Chapter 2

Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms

[1]

In this chapter we review the most important mathematical models and algorithms developed for the exact solution of the one-dimensional bin packing and cutting stock problems, and experimentally evaluate, on state-of-the art computers, the performance of the main available software tools.

Keywords: Bin packing, Cutting stock, Exact algorithms, Computational evaluation.

2.1 Introduction

The (one-dimensional) bin packing problem is one of the most famous problems in combinatorial optimization. Its structure and its applications have been studied since the thirties, see Kantorovich [167]. In 1961 Gilmore and Gomory [134] introduced, for this class of problems, the concept of column generation, by deriving it from earlier ideas of Ford and Fulkerson [125] and Dantzig and Wolfe [89]. This is one of the first problems for which, since the early seventies, the worst-case performance of approximation algorithms was investigated. In the next decades lower bounds were studied and exact algorithms proposed. As the problem is strongly $\mathcal{NP}$-hard, many heuristic and metaheuristic approaches have also been proposed along the years.

The *bin packing problem* (BPP) can be informally defined in a very simple way. We are

[1]The results of this chapter appears in: M. Delorme, M. Iori, and S. Martello, Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms, *European Journal of Operations Research*, 255:1-20, 2016 [98].

given n *items*, each having an integer *weight* w_j $(j = 1, \dots, n)$, and an unlimited number of identical *bins* of integer *capacity* c. The objective is to pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity. (In a different but equivalent *normalized* definition, the weights are real numbers in $[0, 1]$, and the capacity is 1.) We assume, with no loss of generality, that $0 < w_j < c$ for all j.

Many variants and generalizations of the BPP arise in practical contexts. One of the most important applications, studied since the sixties, is the *Cutting Stock Problem* (CSP). Although it has been defined in different ways according to specific real world cases, its basic definition, using the BPP terminology, is as follows. We are given m *item types*, each having an integer *weight* w_j and an integer *demand* d_j $(j = 1, \dots, m)$, and a sufficiently large number of identical *bins* of integer *capacity* c. (In the CSP literature the bins are frequently called *rolls*, the term coming from early applications in the paper industry, and "cutting" is normally used instead of "packing".) The objective is to produce d_j copies of each item type j (i.e., to cut/pack them) using the minimum number of bins so that the total weight in any bin does not exceed the capacity.

This chapter is devoted to a presentation of the main mathematical models that have been proposed, and to an experimental evaluation of the main available software tools that have been developed. The main motivations for writing this survey are to present, for the first time, a complete overview on these problems and to assess, through extensive computational experiments, the performance of the main computer codes that are available for their optimal solution. All the codes we evaluated are either linked or downloadable from a dedicated web page, but one that can be obtained by the authors. The same web page also provides the test instances we used, including new instances that were specifically created as challenging test cases. We believe that this study and the accompanying web page will be useful to many researchers who are still intensively studying this area. Indeed, a search on different bibliographic data bases for articles having in the title either the term "bin packing", or the term "cutting stock", or both, shows a growing interest in these problems in the last 25 years, with sharp increase in recent years (over 150 Google Scholar entries in 2015).

For exhaustive studies on specific research areas concerning the BPP and the CSP, the reader is referred to many surveys that have been published along the years. To the best of our knowledge, the following reviews have been proposed.

The first literature review on these problems was published in 1992 by Sweeney and Paternoster [270], who collected more than 400 books, articles, dissertations, and working papers appeared from 1961 to 1990. In 1990 Dyckhoff [112] proposed a typology of cutting and packing problems, and classified the BPP and the CSP as 1/V/I/M and 1/V/I/R, respectively. In the same year Martello and Toth included a chapter on the BPP in their book [214] on knapsack problems. Two years later Dyckhoff and Finke [113] published a book on cutting and packing problems arising in production and distribution, where they investigated the different structure of these problems, and classified the literature accordingly. A bibliography on the BPP has been compiled by Coffman et al. [76]. More recently, Wäscher et al. [289] re-visited the typology by Dyckhoff [112] and proposed more detailed categorization criteria: the problems we consider are classified as 1-dimensional SBSBPP (*Single Bin Size Bin Packing Problem*) and 1-dimensional SSSCSP (*Single Stock Size Cutting Stock Problem*).

Besides the general surveys discussed above, a number of reviews concerning specific methodologies have been proposed. Already in the early eighties Garey and Johnson [131] and Coffman et al. [77] presented surveys on approximation algorithms for the BPP. Other surveys on approximation algorithms for the BPP and a number of its variants were later proposed by Coffman et al. [72, 71] and Coffman and Csirik [74]. Coffman and Csirik [73] also proposed a four-field classification scheme for papers on bin packing, aimed at highlighting the results in bin packing theory to be found in a certain article. More recently, Coffman et al. [75] presented an overview of approximation algorithms for the BPP and a number of its variants, and classified all references according to [73].

Valério de Carvalho [279] presented a survey of the most popular *Linear Programming* (LP) methods for the BPP and the CSP. A review of models and solution methods was included by Belov [28] in his PhD thesis dedicated to one- and two-dimensional cutting stock problems.

We finally mention that extensions to higher dimensions have been investigated too. In the early nineties, Haessler and Sweeney [146] provided a description of one- and two-dimensional cutting stock problems, and a review of some of the methods to solve them. More recently, surveys on two-dimensional packing problems have been presented by Lodi et al. [193, 194, 197].

In the next section we provide a formal definition of the BPP and the CSP. In Section 2.3 we briefly review the most successful upper and lower bounding techniques for the considered problems. In Sections 2.4, 2.5, and 2.6 we examine pseudo-polynomial formulations, enumeration algorithms, and branch-and-price approaches, respectively. Finally, in Section 2.7, we experimentally evaluate the computational performance of twelve computer programs available for the solution of the considered problems. Conclusions follow in Section 2.8.

2.2 Formal statement

In order to give a formal definition of the problems, let u be any upper bound on the minimum number of bins needed (for example, the value of any approximate solution), and assume that the potential bins are numbered as $1, \ldots, u$. By introducing two types of binary decision variables

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used in the solution;} \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u),$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is packed into bin } i; \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u; j = 1, \ldots, n),$$

we can model the BPP as a basic *Integer Linear Program* (ILP) of the form (see Martello and Toth [214])

$$\min \quad \sum_{i=1}^{u} y_i \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad (i = 1, \ldots, u), \tag{2.2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \quad (j = 1, \ldots, n), \tag{2.3}$$

$$y_i \in \{0, 1\} \quad (i = 1, \ldots, u), \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, u; j = 1, \ldots, n). \tag{2.5}$$

Constraints (2.2) impose that the capacity of any used bin is not exceeded, while constraints (2.3) ensure that each item is packed into exactly one bin.

For the CSP let us define u and y_i as above, and let

$$\xi_{ij} = \text{number of items of type } j \text{ packed into bin } i \quad (i = 1, \ldots, u; j = 1, \ldots, m).$$

The CSP is then

$$
\begin{align}
\min \quad & \sum_{i=1}^{u} y_i && \tag{2.6}\\
\text{s.t.} \quad & \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i && (i = 1, \dots, u), \tag{2.7}\\
& \sum_{i=1}^{u} \xi_{ij} = d_j && (j = 1, \dots, m), \tag{2.8}\\
& y_i \in \{0, 1\} && (i = 1, \dots, u), \tag{2.9}\\
& \xi_{ij} \ge 0, \text{integer} && (i = 1, \dots, u; j = 1, \dots, m). \tag{2.10}
\end{align}
$$

The BPP can be seen as a special case of the CSP in which $d_j = 1$ for all j. In turn, the CSP can be modeled by a BPP in which the item set includes d_j copies of each item type j.

The BPP (and hence the CSP) has been proved to be $\mathcal{NP}$-hard in the strong sense by Garey and Johnson [130] through transformation from the 3-Partition problem.

2.3 Upper and lower bounds

Most exact algorithms for bin packing problems make use of upper and lower bound computations in order to guide the search in the solution space, and to fathom partial solutions that cannot lead to optimal ones. As previously mentioned, for deep reviews on these specific domains, the reader is referred to the surveys listed in Section 2.1. In this section we briefly review the most successful upper and lower bounding techniques that have been developed, with some focus on areas for which no specific survey is available. We use the term *approximation algorithm* for methods for which theoretical results (like, e.g., worst-case performance) can be established, while the term *heuristic* denotes methods for which the main interest relies in their practical behavior.

A classical way for evaluating upper and lower bounds is their absolute worst-case performance ratio. Given a minimization problem and an approximation algorithm A, let $A(I)$ and $OPT(I)$ be the solution value provided by A and the optimal solution value, respectively, for an instance I of the problem. The *worst-case performance ratio* (WCPR) of A is then defined as the smallest real number $\overline{r}(A) > 1$ such that $A(I)/OPT(I) \le \overline{r}(A)$

for all instances I, i.e.,

$$\overline{r}(A) = \sup_I \{A(I)/OPT(I)\}.$$

Similarly, the WCPR of a lower bound L is the largest real number $\underline{r}(L) < 1$ such that, for all instances I, the lower bound value $L(I)$ satisfies $L(I)/OPT(I) \geq \underline{r}(L)$, i.e.,

$$\underline{r}(L) = \inf_I \{L(I)/OPT(I)\}.$$

2.3.1 Approximation algorithms

The simplest BPP approximation algorithms consider the items in any sequence. Algorithm *Next-Fit* (NF) at each iteration packs the next item into the current bin (initially, into bin 1) if it fits, or into a new bin (which becomes the current one) if it does not fit. The WCPR of NF is $\overline{r}(NF) = 2$. Algorithm *First-Fit* (FF) at each iteration packs the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin. Algorithm *Best-Fit* (BF) at each iteration packs the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it. The exact WCPR of FF and BF has been an open problem for forty years, until recently Dósa and Sgall [107, 108] proved that $\overline{r}(FF) = \overline{r}(BF) = \frac{17}{10}$.

Better performances are obtained by preventively sorting the items according to decreasing weight. The WCPR of the resulting algorithms *First-Fit Deacreasing* (FFD) and *Best-Fit decreasing* (BFD) is $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$ (Simchi-Levi [263]). Moreover, this is the best achievable performance, in the following sense:

PROPERTY 1 *No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof Consider an instance of the $\mathcal{NP}$-complete PARTITION problem: is it possible to partition $S = \{w_1, \dots, w_n\}$ into S_1, S_2 so that $\sum_{j\in S_1} w_j = \sum_{j \in S_2} w_j$? Assume a polynomial-time approximation algorithm A for the BPP exists such that $OPT(I) > \frac{2}{3}\, A(I)$ for all instances I, and execute A for an instance $\hat{I}$ of the BPP defined by $(w_1, \dots, w_n)$ and $c = \sum_{j=1}^{n} w_j/2$. If $A(\hat{I}) = 2$ then we know that the answer to PARTITION is `yes`. If instead $A(\hat{I}) \geq 3$ then we know that $OPT(\hat{I}) > \frac{2}{3}3$, i.e., that $OPT(\hat{I}) > 2$, hence the answer to PARTITION is `no`. It follows that we could solve PARTITION in polynomial time. $\square$

Since FFD and BFD provide the best possible WCPR, most research on approximation algorithms for the BPP focused on the *asymptotic* WCPR, defined as the minimum real number $\overline{r}^{\infty}(A)$ such that, for some positive integer k, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances I satisfying $OPT(I) \geq k$. The number of results in this area is impressive and beyond the purpose of this study: we refer the reader to the various surveys that were listed in Section 2.1. The most recent survey (2013), by Coffman et al. [75], examines 200 references from the literature. Among the papers that appeared subsequently, we mention those by Dósa et al. [106] on the FFD algorithm, by Rothvoß [242], who improved a classical result by Karmarkar and Karp [169], and by Balogh et al. [21], who closed a long standing open issue on on-line bin packing.

2.3.2 Lower bounds

To our knowledge, no general survey on lower bounds for the BPP is available. Hence we provide in the following a brief review of the corresponding literature. An obvious lower bound for the BPP, computable in $O(n)$ time, is provided by the so-called *continuous relaxation*, namely

$$L_1 = \left\lceil \sum_{j=1}^{n} w_j/c \right\rceil , \tag{2.11}$$

which gives the rounded solution value of the linear programming relaxation of (2.1)-(2.5). It is easily seen that $\underline{r}(L_1) = \frac{1}{2}$ (see, e.g., Martello and Toth [214]).

A better lower bound was obtained by Martello and Toth [215]. Given any integer α $(0 \leq \alpha \leq c/2)$, let

$J_1 = \{j \in N : w_j > c - \alpha\}$;

$J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

$J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,

and observe that each item in $J_1 \cup J_2$ needs a separate bin, and that no item of J_3 can go to a bin containing an item of J_1. Then $L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2|c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right)$ is a valid lower bound. It can be shown that the overall bound

$$L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq c/2, \alpha \text{ integer}\} \tag{2.12}$$

can be computed in $O(n \log n)$ time and has WCPR equal to $\frac{2}{3}$. Similarly to what happens for algorithms FFD and BFD, this is the best achievable performance, namely:

PROPERTY 2 *No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof We use the same instance of PARTITION as in the proof of Property 1, and the same induced BPP instance $\hat{I}$. Assume a polynomial-time lower bound L for the BPP exists such that $OPT(I) < \frac{3}{2}\, L(I)$ for all instances I, and compute L for instance $\hat{I}$. If $L(\hat{I}) \geq 3$ then we know that the answer to PARTITION is `no`. If $L(\hat{I}) = 2$ then we know that $OPT(\hat{I}) < \frac{3}{2}2$, hence $OPT(\hat{I}) = 2$, i.e., that the answer to PARTITION is `yes`. We could then solve PARTITION in polynomial time. □

Lower bounds that generalize L_2 and can have better practical performance have been proposed by Labbé et al. [182] (lower bound L_{2LLM}), and by Chen and Srivastava [62]. Theoretical properties of such bounds were studied by Elhedhli [117]. Bourjolly and Rebetez [44] proved that the asymptotic WCPR of the bound L_{2LLM} proposed in [182] is $\underline{r}^{\infty}(L_{2LLM}) = \frac{3}{4}$.

Another lower bound, L_3, dominating L_2 was obtained by Martello and Toth [215] by iteratively reducing the instance, and invoking L_2 on the reduced instance. The time complexity grows to $O(n^3)$, and the asymptotic WCPR is $\underline{r}^{\infty}(L_3) = \frac{3}{4}$, as proved by Crainic et al. [85].

A different type of lower bound computation had been considered in the eighties by Lueker [203], who proposed a bounding strategy for the case where all the items are drawn from a uniform distribution, based on dual feasible functions, which were originally introduced by Johnson [164]. Consider the normalized definition of the BPP (see Section 2.1): a real-valued function $u(x)$ is called *dual feasible* if, for any finite set S of nonnegative real numbers, condition $\sum_{x\in S} x \leq 1$ implies $\sum_{x\in S} u(x) \leq 1$. It follows that any lower bound computed over weights $u(w)$ is also valid for the original weights w.

Later on, Fekete and Schepers [122] used dual feasible functions to produce new classes of fast BPP lower bounds. For example, given any normalized instance I of the BPP, any α $(0 \leq \alpha \leq 1/2)$, and an item weight w, let $w' = w/c$ and define

$$U^{(\alpha)}(w') = \begin{cases} 1 & \text{if } w' > 1-\alpha; \\ w' & \text{if } 1-\alpha \geq w' \geq \alpha; \\ 0 & \text{if } w' < \alpha. \end{cases}$$

Then $U^{(\alpha)}(w')$ is a dual feasible function. (Observe in particular that, by considering all α values in $[0, \frac{1}{2}]$ and computing the corresponding bounds L_1, the maximum resulting value coincides with the value provided by L_2.)

A number of other dual feasible functions have been proposed in the literature. We refer the reader to Clautiaux et al. [66] and Alves et al. [9] for recent surveys on these functions and their use for the computation of BPP lower bounds.

Chao et al. [60] and Crainic et al. [86] studied methods for computing "fast" lower bounds for the BPP, i.e., bounds requiring no more than $O(n \log n)$ time. Once a lower bound value, say ℓ, has been computed, it can sometimes be improved through additional considerations: for example, if it can be established that no feasible solution using ℓ bins exists, then $\ell + 1$ is a valid lower bound value. Improvement techniques of this kind have been studied by Dell'Amico and Martello [95], Alvim et al. [11], Haouari and Gharbi [147], and Jarboui et al. [163].

Other effective lower bounds, which however require a non-polynomial time, including the famous Gilmore-Gomory column generation method, are discussed in Section 2.6.

2.3.3 Heuristics and metaheuristics

The focus of this chapter is on the optimal solution of bin packing and cutting stock problems. Approximate and heuristic solutions have thus marginal interest here, but they are commonly used to provide an initial solution to exact algorithms. For the sake of completeness, in this section we briefly review a number of heuristic and metaheuristic approaches.

Heuristics

The first relevant contribution of this kind is probably the one by Eilon and Christofides [114] who presented a heuristic for a number of packing problems, basically consisting of algorithm BFD (see Section 2.3.1), plus a reshuffle routine when the solution is not equal to the continuous relaxation L_1. Roodman [239] presented a set of heuristics for variants of the CSP, mainly based on an initial greedy solution improved through local search. Vahrenkamp [277] proposed a random search for the CSP, based on a heuristic developed by Haessler [145] for generating cutting patterns for trim problems. Wäscher and Gau [288] considered a generalization of the CSP, and studied the computational behavior of heuristics based on rounding the solutions obtained from the LP relaxation of a generalization of the Gilmore and Gomory [134] model (see Section 2.6). The experiments were performed on random instances produced by their generator, CUTGEN (see Gau and Wäscher [132]), which creates CSP instances depending on five parameters: number of item types, minimum and maximum weight, bin capacity, and average demand.

Gupta and Ho [143] proposed a heuristic algorithm based on the minimization of the unused bin capacities, and successfully compared it with FFD and BFD (although at the expenses of higher CPU times). Mukhacheva et al. [222] presented a modified FFD algorithm which was later embedded in the exact algorithm by Belov and Scheithauer [30] (see Section 2.6.3). Osogami and Okano [226] proposed variants of some classical approximation algorithms, and investigated the effect of a local search based on item exchanges. Other modifications of classical approximation algorithms were proposed by Bhatia et al. [40], Kim and Wy [173], and Fleszar and Charalambous [123]. The effectiveness of a hill climbing local search strategy for the BPP, also based on item exchanges, was later investigated by Lewis [187].

As for most $\mathcal{NP}$-hard problems, starting from the early nineties many metaheuristic approaches of all kinds have been proposed for the BPP and the CSP. In the following, we list, grouped by metaheuristic paradigm, a number of contributions that provided interesting insights into the problems at hand.

Simulated annealing and Tabu search

A classical simulated annealing approach to the BPP was implemented by Kämpke [166], while a variant of the method (called weight annealing) was proposed by Loh et al. [199]. Scholl et al. [251] used a Tabu search procedure to speed up their well-known exact algorithm (BISON) for the BPP, treated in Section 2.5.1. Alvim et al. [11] embedded a Tabu search in a hybrid improvement heuristic for the BPP.

Population based algorithms

Probably, the first genetic approach to the BPP is the one by Falkenauer and Delchambre [120]: they showed that the classical genetic approach cannot work efficiently for certain kinds of problems (like the BPP), and presented a variant (the grouping genetic algorithm) capable of producing a good computational behavior. Falkenauer [119] improved this method through hybridization with the dominance criterion by Martello and Toth [215] (see Section 2.5.1), and proposed a set of benchmark instances that was later adopted by many authors for computationally testing BPP algorithms. Although Gent [133] showed that the majority of them are very easy, these instances were used, e.g., for testing the genetic approaches by Reeves [235], Bhatia and Basu [39], Singh and Gupta [264], Ülker et al. [276], and Stawowy [267]. Other genetic algorithms were proposed by Poli et al. [230] and by Rohlfshagen and Bullinaria [237, 238]. Recently, a very effective genetic algorithm was proposed by Quiroz-Castellanos et al. [233].

Levine and Ducatelle [186] used an ant colony approach combined with a local search to solve the BPP. Liang et al. [188] proposed an evolutionary programming algorithm for the CSP and some of its variants.

Hyper-heuristics

Ross et al. [240, 241] attacked the BPP through combinations of genetic algorithms and hyper-heuristics. Other combinations of evolutionary algorithms and hyper-heuristics for the BPP were proposed by López-Camacho et al. [200], Sim et al. [262], and Burke et al. [50].

Bai et al. [17] tested on BPP instances their simulated annealing hyper-heuristic approach. Sim and Heart [261] used genetic programming as a generative hyper-heuristic to create deterministic heuristics.

Other meta-heuristic approaches

Fleszar and Hindi [124] obtained new heuristics for the BPP by modifying the heuristic of Gupta and Ho [143] and proposed a variable neighborhood search algorithm. Gómez-Meneses and Randall [137] proposed a hybrid extremal optimization approach with local search for the BPP.

2.4 Pseudo-polynomial formulations

In this section we introduce considerations on polynomial and pseudo-polynomial models, we present the main pseudo-polynomial formulations proposed in the literature, and highlight some relations among them.

2.4.1 Considerations on the basic ILP model

The textbook BPP model (2.1)–(2.5), which has its roots in the seminal work by Kantorovich [167], was formally defined in 1990 by Martello and Toth [214]. It involves a polynomial number of variables and constraints but is not very efficient in practice, as shown in Section 2.7. Several attempts have been made since then to try and improve the computational behavior of the model, especially by providing families of valid inequalities. The simple inequality $y_i \geq y_{i+1}$ for $i = 1, \dots, u-1$ reduces the size of the enumeration tree by imposing that the bins are used in increasing order of index. Symmetric solutions can be further removed by setting $x_{ij} = 0$ for all $j = 1, \ldots, u - 1$ and $i = j + 1, \ldots, u$, as there is always an optimal solution in which item 1 is packed in bin 1, item 2 either in bin 1 or 2, and so on. The linear relaxation of the model can be further strengthened by imposing that full items cannot be packed into bins i with fractional y_i value, i.e., $x_{ij} \leq y_i$ for all $i = 1, \ldots, u$ and $j = 1, \ldots, n$. A number of enhanced families of inequalities, including the well-known *cover inequalities* and their generalizations, derive from studies on the knapsack polytope. For a detailed description of these inequalities, as well as of efficient separation procedures, we refer the reader to Gabrel and Minoux [128] and Kaparis and Letchford [168].

Despite these results, the computational behavior of model (2.1)–(2.5) remains quite poor. The literature has consequently focused on the study of models with better computational performance, including *pseudo-polynomial* models. The drawback of these models is that the number of variables depends not only on the number of items but also on the bin capacity. On the other hand, they provide a stronger linear relaxation than that given by (2.1)–(2.5).

In Section 2.4.2 we address the oldest such model, independently developed by Rao [234] in 1976 and by Dyckhoff [111] few years later. The most relevant approach of this kind (somehow anticipated by Wolsey [293] in 1977) was presented in 1999 by Valério de Carvalho [278] for the CSP. In 2010, Cambazard and O'Sullivan [52] presented a BPP pseudo-polynomial model based on a similar idea, but described in a form inspired by the graph construction used by Trick [274] for propagating knapsack constraints. We anticipate its description in Section 2.4.3, since this makes it easier to understand the Valério de Carvalho model, which is then discussed in Section 2.4.4 together with a recently proposed variant.

The following example is resumed a number of times in the next sections.

EXAMPLE 1 *For the BPP, we consider an instance with* $n=6$, $c=9$, *and* $w=(4,4,3,3,2,2)$. *The equivalent CSP instance has* $m=3$, $c=9$, $w=(4,3,2)$, *and* $d=(2,2,2)$. *An optimal solution has value 2, and packs three items (of weight 4, 3, and 2) in each bin.* □

2.4.2 One-cut formulation

The idea behind the Rao [234] and Dyckhoff [111] model for the CSP is to simulate the physical cutting process, by first dividing an ideal bin into two pieces (*left* and *right*), where the left piece is an item that has been cut, while the right piece is either a residual that can be re-used to produce other items or it is another item. The process is iterated on cutting residuals or new bins, until all demands are fulfilled. For the sake of clarity, we use in this section the term “width” for “weight”.

Let $W=\{w_1,w_2,\dots,w_m\}$ be the set of item widths. Let R be the set of all possible relevant *residual widths*, computed by subtracting from the bin capacity c all feasible combinations of item widths (including the empty combination), provided the resulting value is not less than the minimum item width. Formally,

$$R=\{c-\bar{w}:\bar{w}=\sum_{j=1}^{m} w_j x_j, \bar{w}\le c-\min_j\{w_j\}, x_j\in\{0,1,\dots,b_j\}(j=1.\dots,m)\}.$$

The level of demand for a certain width q is

$$L_q=\begin{cases} d_i & \text{if } q=w_i \text{ for some item type } i;\\ 0 & \text{otherwise.}\end{cases} \qquad (q\in W\cup R).$$

Additionally, for each $q\in W\cup R$, let

- $A(q)=\{p\in R: p>q\}$ if $q\in W$ (and $A(q)=\emptyset$ otherwise) denote the set of piece widths that can be used for producing a left piece (item) of width q,
- $B(q)=\{p\in W: p+q\in R\}$ denote the set of item widths that, if cut as a left piece, would leave a right piece (residual) of width q, and
- $C(q)=\{p\in W: p<q\}$ denote the set of item widths that can be cut, as a left piece, from a residual of width q.

By introducing an integer variable x_{pq}, that gives the number of times a bin, or a residual of width p, is cut into a left piece of width q and a right piece of width $p-q$ ($p \in R, q \in W, p > q$), the *one-cut* model can be defined as the ILP

$$\min \quad \sum_{q \in W} x_{cq} \tag{2.13}$$

$$\text{s.t.} \quad \sum_{p \in A(q)} x_{pq} + \sum_{p \in B(q)} x_{p+q,p} \geq L_q + \sum_{r \in C(q)} x_{qr} \qquad q \in (W \cup R) \backslash \{c\}, \tag{2.14}$$

$$x_{pq} \geq 0 \text{ and integer} \qquad p \in R, q \in W, p > q. \tag{2.15}$$

The objective function (2.13) minimizes the number of times an item is cut from a bin. Constraints (2.14) impose that, for each width q, the sum of the left pieces of width q plus the sum of the right pieces of width q is not smaller than the level of demand of width q plus the number of times a residual of width q is used to produce smaller items.

Example 1 (resumed) *For the CSP instance we have* W=$\{2,3,4\}$ *and* R=$\{2,3,4,5,6,7,9\}$. *We obtain:*

- $A(2) = \{3,4,5,6,7,9\}$, $A(3) = \{4,5,6,7,9\}$, $A(4) = \{5,6,7,9\}$, $A(5) = A(6) = A(7) = A(9) = \emptyset$;
- $B(2) = B(3) = \{2,3,4\}$, $B(4) = \{2,3\}$, $B(5) = \{2,4\}$, $B(6) = \{3\}$, $B(7) = \{2\}, B(9) = \emptyset$;
- $C(2) = \emptyset$, $C(3) = \{2\}$, $C(4) = \{2,3\}$, $C(5) = C(6) = C(7) = C(9) = \{2,3,4\}$.

An optimal solution is then given by $x_{9,4} = 2$, $x_{5,3} = 2$, *and* $x_{pq} = 0$ *otherwise. In other words, we cut two items of width 4 from two bins, and two items of width 3 from the two residuals we have obtained. The two resulting residuals provide two items of width 2.* □

Set R can be obtained by running a standard dynamic programming algorithm, or a recursive algorithm, that generates all possible item combinations. The one-cut model (2.13)-(2.15) has $O(mc)$ variables and $O(c)$ constraints.

Stadtler [266] studied the combinatorial structure of the one-cut model and extended it by including additional variables and constraints. He also worked on comparing the model and the classical column generation approach, and concluded that "The set of real world cutting stock problems solvable by the one-cut model (of Rao and Dyckhoff) is only a subset of those which could be tackled by the column generation approach (of Gilmore and Gomory)".

2.4.3 DP-flow formulation

A simple pseudo-polynomial model is obtained by associating variables to the decisions taken in a classical *dynamic programming* (DP) table. In the BPP model proposed by Cambazard and O'Sullivan [52], known as *DP-flow*, the DP states are represented by a graph in which a path that starts from an initial node and ends at a terminal node represents a feasible filling of a bin. Let us denote by (j, d) $(j = 0, \ldots, n$ and $d = 0, \ldots, c)$ a DP state in which decisions have been taken up to item j and result in a *partial bin filling* of d units. Let us also denote by $((j, d), (j + 1, e))$ an arc connecting states (j, d) and $(j + 1, e)$. Such arc expresses the decision on whether packing or not item $j + 1$ starting from the current state (j, d): the state reached by the arc is $(j + 1, d + w_{j+1})$ if item $j + 1$ is packed, and $(j + 1, d)$ otherwise.

Example 1 (resumed) *The DP table associated with our instance is shown in Figure 2.1, where states are represented by nodes and organized in $n + 1$ horizontal layers. The table includes an additional* terminal state $(n + 1, c)$*, and states in layer n are connected to it by* loss arcs *(dashed lines), that express the amount of unused capacity in a given bin.* □

Let A denote the set of all arcs. As a feasible bin filling is represented by a path that starts from node $(0, 0)$ and ends at node $(n + 1, c)$, the BPP is to select the minimum number of paths that contain all items. To formulate this decision problem, let us associate an integer variable $x_{j,d,j+1,e}$ to arc $((j, d), (j + 1, e)) \in A$, representing the number of times the arc has been chosen to form paths. Let $\delta^-((j, d))$ (resp. $\delta^+((j, d))$) denote the set of arcs entering (resp. emanating from) state (j, d). The BPP can be then modeled as

$$\min \quad z \tag{2.16}$$

$$\text{s.t.} \quad \sum_{((j,d),(j+1,e))\in\delta^+((j,d))} x_{j,d,j+1,e} - \sum_{((j-1,e),(j,d))\in\delta^-((j,d))} x_{j-1,e,j,d} = \begin{cases} z & \text{if } (j,d) = (0,0); \\ -z & \text{if } (j,d) = (n+1,c); \\ 0 & \text{otherwise,} \end{cases} \tag{2.17}$$

$$\sum_{((j-1,d),(j,d+w_j))\in A} x_{j-1,d,j,d+w_j} = 1 \qquad (j = 1, \ldots, n), \tag{2.18}$$

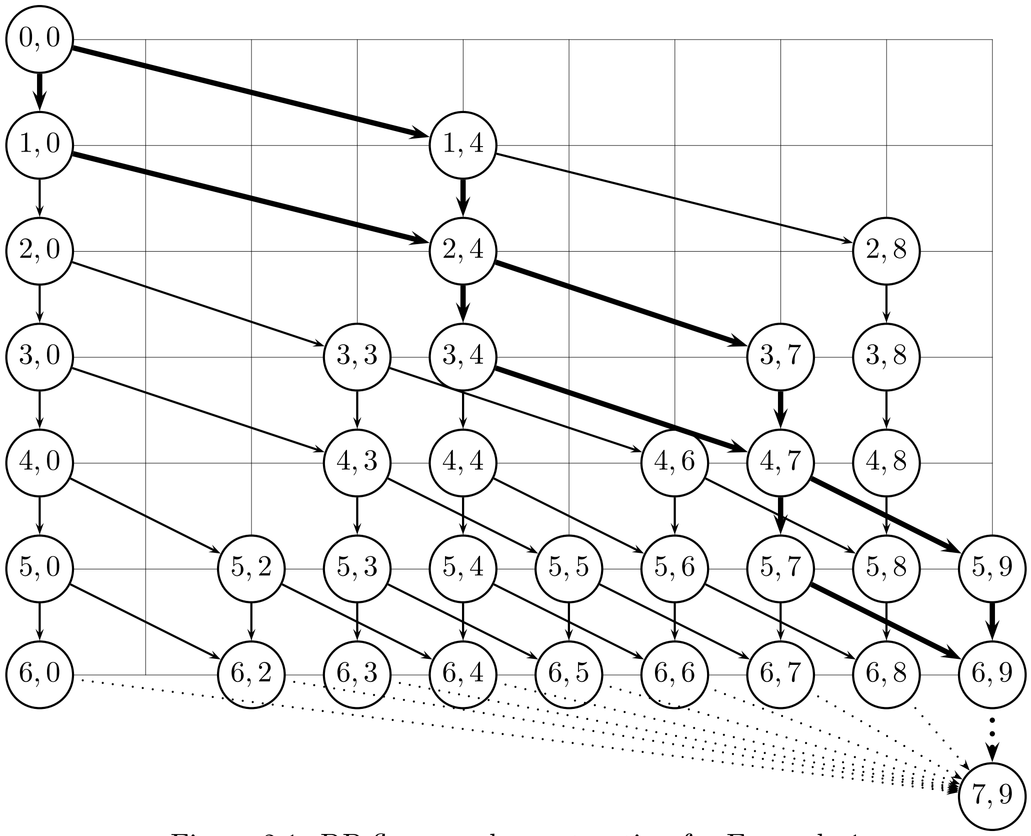


Figure 2.1: DP-flow graph construction for Example 1

$$x_{j,d,j+1,e} \geq 0 \text{ and integer} \qquad ((j,d),(j+1,e)) \in A. \quad (2.19)$$

The objective function (2.16) minimizes the number of bins. Constraints (2.17) impose the flow (number of bins) conservation at all nodes, while constraints (2.18) ensure that each item is packed exactly once. Note that a "$\geq$" sign could be used in (2.18) without affecting the correctness of the model.

Example 1 (resumed) *For the BPP instance an optimal solution is produced by the two paths highlighted in Figure 2.1, namely* [(0,0), (1,4), (2,4), (3,7), (4,7), (5,9), (6,9), (7,9)] *and* [(0,0), (1,0), (2,4), (3,4), (4,7), (5,7), (6,9), (7,9)]. $\square$

The DP-flow model (2.16)-(2.19) has $O(nc)$ variables and constraints. This formulation was developed in [52] for the BPP, but it could be extended to the CSP. The formulations introduced in the next section were instead specifically tailored on the CSP.

2.4.4 Arc-flow formulations

An effective CSP pseudo-polynomial formulation, denoted *arc-flow*, was presented by Valério de Carvalho [278], who used it in a branch-and-price algorithm (see Section 2.6). To make its comprehension easier, consider again Example 1, and the DP representation depicted in Figure 2.1. Now imagine that the graph is vertically shrunk, by grouping all states with the same partial bin filling into a single one. In this way, the "vertical" arcs disappear, while the "slanting" ones that connect the same pair of nodes merge into a single arc. Figure 2.2 shows the counterpart of Figure 2.1. Note that the loss arcs, which imply no bin filling variation, connect here consecutive nodes instead of (equivalently) going to the terminal node. Let A' denote the resulting arc set, and x_{de} the number of times arc $(d,e) \in A'$ is chosen. The filling of a single bin corresponds to a path from node 0 to node c in this graph. The CSP can then be modeled as the following ILP:

$$\min \quad z \tag{2.20}$$

$$\text{s.t.} \quad -\sum_{(d,e)\in\delta^-(e)} x_{de} + \sum_{(e,f)\in\delta^+(e)} x_{ef} = \begin{cases} z & \text{if } e = 0; \\ -z & \text{for } e = c; \\ 0 & \text{otherwise,} \end{cases} \tag{2.21}$$

$$\sum_{(d,d+w_i)\in A'} x_{d,d+w_i} \geq b_i \qquad (i = 1,\dots,m), \tag{2.22}$$

$$x_{de} \geq 0 \text{ and integer} \qquad (d,e) \in A', \tag{2.23}$$

where $\delta^-(e)$ (resp. $\delta^+(e)$) denotes the set of arcs entering (resp. emanating from) e.

Constraints (2.21) impose the flow conservation at all nodes. Constraints (2.22) impose that, for each item type i, at least b_i arcs of length w_i are used, i.e., that at least b_i copies of item type i are packed.

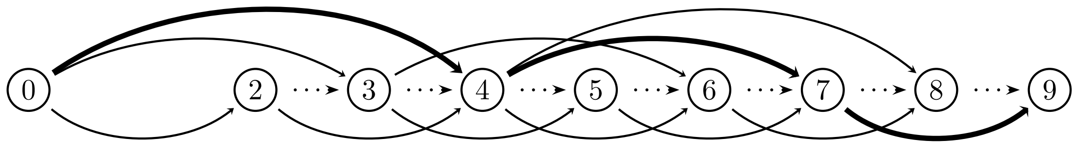


Figure 2.2: Arc-flow representation of the graph of Figure 2.1

EXAMPLE 1 (resumed) *An optimal solution to the CSP instance consists of two identical paths* $[0, 4, 7, 9]$*, highlighted in Figure 2.2.* □

The arc-flow model (2.20)-(2.23) has $O(mc)$ variables and $O(m+c)$ constraints. Valério de Carvalho [278] proposed however a number of improvements to the above basic model, aimed at reducing the number of arcs. For example (see again Figure 2.2), it is enough to only create nodes that correspond to feasible combinations of item weights. In addition, it is proved in [278] that the linear programming relaxation of (2.20)-(2.23) has the same solution value as the Gilmore and Gomory [134] model (see Section 2.6).

Very recently, Brandão and Pedroso [45] proposed an alternative CSP arc-flow formulation. They start with a multi-graph generalization of the arc-flow formulation by Valério de Carvalho [278] which uses a level per item type and can be seen as a CSP version of the DP-flow formulation. A three-index variable, say x_{dei}, is consequently associated with each arc (d, e, i), where d and e are the tail and the head, while i represents the item type. This leads to a three-index model analogous to (2.20)-(2.23) in which, however, those inequality constraints (2.22) for which $b_i = 1$ are changed to equalities. The resulting graph is then reduced through graph compression techniques, and solved through a standard ILP solver. The overall code (see Section 2.7) proved to be very efficient on benchmark instances.

2.5 Enumeration algorithms

The first attempts to exactly solve the BPP and the CSP were developed in the fifties and in the sixties using LP relaxations and dynamic programming (see Eisemann [115] and Gilmore and Gomory [134, 135, 136]). Starting from the early seventies, research in this field focused on branch-and-bound.

2.5.1 Branch-and-bound

To the best of our knowledge, the first branch-and-bound algorithm for the BPP was proposed by Elion and Christofides [114], who adapted the general enumerative scheme proposed by Balas [19] for solving LPs with zero-one variables. Their algorithm produces a binary decision tree in which a node generates two descendant nodes by assigning a certain item to a certain bin, or by excluding it from that bin. The process is initialized by the heuristic solution produced by the BFD algorithm (see Section 2.3.1) followed by a reshuffle routine. Lower bounds are obtained from a standard LP relaxation. The algorithm could only solve instances of very moderate size.

Later on, thanks to the development of better heuristics, improved lower bounds, and reduction procedures, a more powerful branch-and-bound algorithm for the BPP, called MTP, was developed by Martello and Toth [214]. During the nineties, this algorithm, whose Fortran code was available, has been the standard reference for the exact solution of the BPP. Their reduction procedures, which were later adopted by several authors, are based on the following dominance criterion. Given an instance I of the BPP, define a *feasible set* F as a set of items such that $\sum_{j \in F} w_j \leq c$. A feasible set F_1 *dominates* another feasible set F_2 if the optimal solution obtained by imposing F_1 as the content of a bin is not greater than that obtained by imposing F_2 as the content of a bin. Martello and Toth [215] proved the following

Property 3 *Given two distinct feasible sets F_1 and F_2, if there exists a partition $P = \{P_1, \dots, P_\ell\}$ of F_2, and a subset $\{j_1, \dots, j_\ell\}$ of F_1 such that $w_{j_h} \geq \sum_{k \in P_h} w_k$ for $h = 1, \dots, \ell$, then F_1 dominates F_2.*

Clearly, if a feasible set F containing an item j dominates all other feasible sets containing the same item j, then we can impose F to a bin and reduce the instance accordingly. Checking all such sets is computationally too heavy, and hence the Martello-Toth *reduction procedure* MTRP limits the search to feasible sets of cardinality at most three and has $O(n^2)$ time complexity. The procedure was also used (iteratively) to produce, in $O(n^3)$ time, an improved lower bound L_3. Algorithm MTP sorts the items according to non-increasing weight, and indexes the bins according to the order in which they are initialized: at each decision node, the next free item is assigned, in turn, to all initialized bins that can accommodate it, and to a new bin. The branch-decision tree is searched according to a depth-first strategy.

Some years after the development of MTP, Scholl et al. [251] proposed the other most successful branch-and-bound algorithm for the BPP, known as BISON. They adopted some of the most powerful tools from MTP, and added new lower bounds and emerging techniques like Tabu search, obtaining an improved exact method for the BPP. A couple of years later, Schwerin and Wäscher [255] improved the competitiveness of MTP with respect to BISON through a lower bound provided by the column generation method developed by Gilmore and Gomory [134] (see Section 2.6) for the CSP.

In the early noughties Mukhacheva et al. [222] proposed a pattern oriented branch-and-bound algorithm for both the BPP and the CSP, while Korf [179, 180] proposed a "bin completion" algorithm (later improved on by Schreiber and Korf [252]) in which decision nodes are produced by assigning a feasible set to a bin. However, starting from the late nineties, branch-and-price (see Section 2.6) proved to be very effective, and became the most popular choice for the exact solution of the BPP. Tree search enumeration is also an ingredient of constraint programming approaches, that are briefly examined in the next section.

2.5.2 Constraint programming approaches

In the last decade some attempts have been proposed to solve the BPP through *Constraint Programming* (CP). Shaw [257] presented a new dedicated constraint (later on implemented in the CP optimizer of CPLEX as `IloPack`) based on a set of pruning and propagation rules that also make use of lower bound L_2. In the following years, some improvements on Shaw's constraint were proposed. Cambazard and O'Sullivan [52] integrated pseudo-polynomial formulations discussed in Section 2.4 within the CP approach by Shaw. Dupuis et al. [110] used lower bound L_{2LLM} by Labbé et al. [182] and an additional reduction algorithm. Schaus et al. [245] introduced a filtering rule based on cardinality considerations.

2.6 Branch-and-price

The *Branch-and-Price* algorithms for the BPP and the CSP are based on the seminal work by Gilmore and Gomory [134, 135], who presented the classical set covering formulation for the CSP, and showed how to solve its continuous relaxation by means of a *column generation* approach. Although the branch-and-price approach could be used to solve all the models of Section 2.4, to the best of our knowledge, in the BPP and CSP literature it was mainly adopted for Gilmore-Gomory formulations, and hence our description follows such model.

2.6.1 Set covering formulation and column generation

The set covering formulation is based on the enumeration of all *patterns*, i.e., of all combinations of items that can fit into a bin. For the sake of conciseness, in the following we use p to define both a pattern and its index, and P to define both the set of patterns and the set of patterns indices.

For the CSP, a *pattern* p is described by an integer array $(a_{1p}, a_{2p}, \dots, a_{mp})$, where a_{jp} gives the number of copies of item j that are contained in pattern p, and satisfies $\sum_{j=1}^{m} a_{jp} w_j \leq c$, and $a_{jp} \geq 0$, integer $(j = 1, \dots, m)$. Let us introduce an integer variable y_p that gives, for each $p \in P$, the number of times pattern p is used. The *set covering* formulation of the CSP is given by the ILP

$$\min \sum_{p \in P} y_p \tag{2.24}$$

$$\text{s.t.} \sum_{p \in P} a_{jp} y_p \geq d_j \qquad (j = 1, \dots, m), \tag{2.25}$$

$$y_p \geq 0 \text{ and integer } (p \in P). \tag{2.26}$$

Objective function (2.24) requires the minimization of the number of bins, whereas constraints (2.25) impose that the subset of selected patterns contains at least d_j copies of each item j.

Similarly, for the BPP: (i) a pattern p is defined by a binary array $(a_{1p}, a_{2p}, \dots, a_{np})$, where a_{jp} is equal to 1 if item j is contained in pattern p and 0 otherwise; (ii) y_p is a decision variable taking the value 1 iff pattern p is used in the solution. The set covering formulation is then obtained by modifying (2.25) and (2.26) as $\sum_{p \in P} a_{jp} y_p \geq 1 \, (j = 1, \dots, n)$ and $y_p \in \{0, 1\} \, (p \in P)$, respectively.

Contrary to what happens with pseudo-polynomial models, in these formulations the number of feasible patterns is exponential in the number of items, so enumerating all of them is prohibitive even for moderate-size instances. *Column generation* techniques are consequently adopted for these cases, while they are less frequent for the other models. Let us briefly describe the basic technique for the CSP. We first define the continuous relaxation of (2.24)-(2.26) by removing the integrality constraints, and heuristically initialize it with a reduced set of patterns $P' \subseteq P$ that provides a feasible solution. The resulting optimization problem, called the *restricted master problem* (RMP), is

$$\min \sum_{p \in P'} y_p \tag{2.27}$$

$$\text{s.t.} \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \dots, m), \tag{2.28}$$

$$y_p \geq 0 \qquad (p \in P'). \tag{2.29}$$

Once (2.27)-(2.29) has been solved, let π_j be the dual variable associated with the jth constraint (2.28). The existence of a column $p \notin P'$ that could reduce the objective function value (*pricing problem*) is determined by the *reduced costs* $\overline{c}_p = 1 - \sum_{j=1}^{m} a_{jp}\pi_j$ $(p \notin P')$. The column with the most negative reduced cost may be determined by solving an *unbounded knapsack problem* in which the profits are given by the dual variables π_j, i.e., the *slave problem* (SP):

$$\max \quad \sum_{j=1}^{m} \pi_j v_j, \tag{2.30}$$

$$\sum_{j=1}^{m} w_j v_j \leq c, \tag{2.31}$$

$$v_j \geq 0 \text{ and integer} \qquad (j = 1, \ldots, m), \tag{2.32}$$

where v_j is the number of times item type j is used. If the solution to the SP has value greater than 1, then the corresponding column (i.e., the corresponding pattern) has negative reduced cost and it is added to the RMP. The process is iterated until no column with negative reduced cost is found, thus providing the optimal solution value to the continuous relaxation of the set covering formulation.

We finally observe that a pattern, as defined above, could contain more than d_j copies of an item j, and hence an equivalent definition (*proper pattern*) is obtained by also imposing $a_{jp} \leq d_j$ $(j = 1, \ldots, m)$. If this formulation is used, the SP consists of a *bounded* knapsack problem, defined by (2.30)-(2.32), and $v_j \leq d_j$ $(j = 1, \ldots, m)$.

This results in a (slightly) stronger lower bound, as the number of feasible patterns becomes smaller. Thorough discussions on this issue may be found in Nitsche et al. [225], and Caprara and Monaci [56]. The lower bound produced by the continuous relaxation is usually very tight, and has been extensively studied in the literature, both from a theoretical and a practical point of view. These studies are presented in the next section.

Alternative column generation approaches make use of the *set partitioning* formulation, in which the '$\geq$' sign is replaced by '$=$' in constraints (2.25). Indeed, if an optimal solution to model (2.24)-(2.26) contains more than d_j copies of an item j, then an equivalent solution can be obtained by arbitrarily removing excess copies from the bins. Consequently the set covering and the set partitioning formulations for the CSP lead to the same optimal solution value. Similar considerations hold for the BPP.

Valério de Carvalho [280] proposed dual cuts to accelerate the column generation process for the CSP. The idea is to add to the RMP "extra" columns (cuts in the dual) that can be found in a fast way and can accelerate the convergence to the continuous optimal solution by reducing the number of "standard" columns generated by the SP. This line of research was pursued by Ben Amor et al. [32], who used dual constraints that are satisfied by at least one optimal dual solution to reduce the typical long-tail effect of column generation. Clautiaux et al. [67] introduced additional dual cuts, as well as a method to tighten lower and upper bounds on the dual variables, in order to stabilize the column generation approach.

Most of the above methods have been used as a basis to produce effective branch-and-price algorithms, that we survey in Section 2.6.3.

We conclude this section by mentioning some variants of column generation. Briant et al. [46] compared bundle methods and column generation for solving the LP relaxation of the set covering model, testing them on some BPP and CSP instances. Kiwiel [174] proposed a special bundle method that allows inaccurate solutions to the SP, paired with a rounding heuristic to produce a feasible solution, and experimented it on the CSP. Elhedhli and Gzara [118] recently proposed another heuristic approach to the BPP, based on Lagrangian relaxation and column generation.

2.6.2 Integer round-up property

Let L_{LP} be the solution value of the continuous relaxation of the set covering formulation, and z_{opt} the optimal solution value. A BPP (or a CSP) instance is said to have the *Integer Round-Up Property* (IRUP) if the rounded up value of L_{LP}, $\lceil L_{\text{LP}} \rceil$, is equal to z_{opt}. We call such an instance an *IRUP instance.* On the basis of early computational experiments, it was conjectured in the seventies that the IRUP held for any BPP and CSP instance.

The IRUP conjecture was only proved for some special classes of instances (see, e.g., Berge and Johnson [37], and Marcotte [209]), until it was disproved in the eighties. Marcotte [210] provided an instance for which the IRUP does not hold (*Non-IRUP instance* in the following) with $n = 24$ and $c = 3\,397\,386\,255$. Later on, Chan et al. [59] presented a smaller disproving instance, with $n = 15$ and $c = 1\,111\,139$. For both instances the gap between the rounded up lower bound and the optimal solution is exactly one bin. It was then conjectured (see, e.g., Scheithauer and Terno [247, 248]) that $z_{opt} - \lceil L_{\mathrm{LP}} \rceil \leq 1$ holds for any BPP and CSP instance (*Modified Integer Round-Up Property*, MIRUP).

To the best of our knowledge the MIRUP conjecture is still open both for the BPP and the CSP, but a number of interesting results have been obtained while attempting to close it. Kartak [170] presented sufficient conditions under which an instance is Non-IRUP, as well as an algorithm to check them. By performing a huge number of computational tests on randomly generated instances, Schoenfield [250] (see also Belov and Scheithauer [30]) created a set of hard instances, including some satisfying $z_{opt} - L_{\mathrm{LP}} > 1$. Rietz and Dempe [236] presented methods to construct Non-IRUP instances through perturbations of the item weights that make certain cutting patterns infeasible. Caprara et al. [54] produced a large set of Non-IRUP instances by using a relationship between the BPP and the edge coloring problem. The smallest such instances have $n = 13$ and $c = 100$, showing that Non-IRUP instances may also appear in practical contexts. They also gave a method to transform an IRUP instance into a Non-IRUP one. Very recently, Kartak et al. [171] generated classes of Non-IRUP instances through an enumerative method, and showed that the IRUP holds when $n \leq 9$. Furthermore, they produced Non-IRUP instances with 10 items. Eisenbrand et al. [116] and Newman et al. [224] studied the relationship between the MIRUP conjecture for the BPP and the Becks three-permutation conjecture for discrepancy theory (see Beck and Sós [27]).

For the CSP, the MIRUP conjecture is an open issue both in the case where proper patterns are imposed or not. For the latter case it is easier to find Non-IRUP instances, because, as previously mentioned, the resulting lower bound is weaker.

We conclude by observing that all Non-IRUP instances of the literature have been solved exactly. In Section 2.7.1 we discuss a method to generate Non-IRUP instances that are difficult to solve exactly.

2.6.3 Branch(-and-cut)-and-price algorithms

When the solution obtained at the end of the column generation method of Section 2.6.1 is fractional, an additional effort is required to find a feasible integer solution. The generation of all possible patterns followed by the direct solution of (2.24)-(2.26) at integrality is the most obvious option, but it can only be adopted for instances of small size, or characterized by a special structure (see, e.g., Goulimis [138]). Other, non exact, methods simply use rounding heuristics (like, e.g., Roodman [239], Haessler and Sweeney [146], and

Holthaus [154]), but their efficiency strongly depends on the instances at hand. When these methods fail in producing an optimal integer solution, one can embed the column generation lower bound L_{LP} into an enumeration tree, thus obtaining a *branch-and-price* algorithm. The main difficulty of this approach is that the branching decisions that have been taken during the enumeration must be embedded in the master and/or the slave problem, so as to avoid the generation of columns that have been excluded by the branch decisions. We review in this section the main methods that have been proposed in the literature to handle this issue.

The first branch-and-price algorithm for the BPP is probably the one proposed by Vance et al. [282] in 1994. At each decision node the algorithm considers those bins for which the decision variable y_p is fractional, and selects the pair of items that are fractionally packed into the same bin and have largest total weight: following a branching rule originally developed by Ryan and Foster [243] for set partitioning problems, such items are then forced to be packed either together or separately. In the latter case the resulting subproblem is a knapsack problem with an additional constraint, while in the former case it is sufficient to merge the two items into a unique one. Early termination of nodes without performing the complete column generation is obtained by using a lower bound on the objective function value due to Farley [121]. The following year Scheithauer and Terno [246] proposed a hybrid strategy for the CSP, oriented to the conjecture that the MIRUP holds for the instance at hand. They first reduce the instance by solving its continuous relaxation and rounding down the solution, so as to obtain a partial integer solution and a residual instance. The residual instance is then attacked through heuristic algorithms and, if they fail in producing an overall optimal solution, it is exactly solved through a branch-and-bound algorithm which includes pricing ideas.

Some years later, Vance [281] focused on the CSP and showed that the application of the classical Dantzig–Wolfe [89] decomposition to the CSP model (2.6)-(2.10) leads to the set covering model (2.24)-(2.26), and used this result to implement two specifically tailored branching rules.

In the late nineties, Valério de Carvalho [278] proposed a column generation approach which is not based on the traditional Gilmore-Gomory model but on the arc-flow formulation of Section 2.4.4. The algorithm branches on a fractional flow variable x by imposing it to be either not smaller than $\lceil x \rceil$ or not greater than $\lfloor x \rfloor$. The branching constraints are directly added to the master. The slave is in this case a longest path problem in an acyclic directed graph, which is solved through dynamic programming.

Vanderbeck [283] introduced a branch-and-price algorithm whose rule is to branch on a set of columns. This is obtained by adding a constraint to the master to impose that the sum s of the variables associated with such set are either not smaller than $\lceil s \rceil$ or not greater than $\lfloor s \rfloor$. In this way the descendant nodes involve a complicated variant of the knapsack problem. The convergence of the algorithm is improved by cut generation at the decision nodes, so that the method can be seen as a *Branch-and-Cut-and-Price* algorithm. Later on, Vanderbeck [284, 286] tested on BPP and CSP instances some branching schemes he developed for general branch-and-price algorithms.

Degraeve and Schrage [93] proposed a branch-and-price approach to the CSP, which selects for branching a pattern associated with a fractional variable. A specific constraint is added to the slave in order to prevent such pattern to be generated at descendant nodes. Degraeve and Peeters [92] improved the algorithm by adding heuristics, pruning rules, and a sub-gradient procedure to speed up the solution of the LP relaxations. In addition, they adopted an efficient way to handle decision nodes, by focusing on the solution of the sub-problem obtained by subtracting from the item demands the values of the rounded-down LP solution.

In the early noughties, Scheithauer et al. [249] proposed an exact solution approach for the CSP based on cutting plane generation. The algorithm computes a lower bound by solving the continuous relaxation of the set covering formulation, and an upper bound by using heuristics. If there is a gap between these two values, Chvátal-Gomory cuts [65] are added to the formulation to possibly increase the lower bound value, and the process is iterated. The slave is solved by a specifically tailored branch-and-bound method that takes into account the dual variables associated with the additional constraints. The method was improved in Belov and Scheithauer [29], and then embedded into a branch-and-price algorithm in Belov and Scheithauer [30]. The resulting algorithm directly branches on the variables associated with the patterns, selecting the variable whose fractional value is closer to 0.5. Later on, Belov et al. [31] investigated the performance of combining Chvátal-Gomory cuts and arc-flow formulations, which however did not prove to be very effective.

The list of papers commented in this section is not exhaustive, as the literature on branch-and-price algorithms for the BPP and the CSP is huge. We mention here Desaulniers et al. [104], who introduce a generic framework for dealing with cutting planes in branch-and-price algorithms. For further details, we refer the reader to the specific survey by Ben Amor and Valério de Carvalho [33], who show how the set covering formulation

(2.24)-(2.26) can be derived from various compact formulations though Dantzig-Wolfe decompositions. Other relevant remarks can be found in the general survey by Lübbecke and Desrosiers [202], who treat a number of implementation issues, including specific considerations on the BPP and the CSP.

2.7 Experimental evaluation

Besides presenting the main mathematical models that have been proposed for the BPP and the CSP, one main purpose of this survey is to experimentally compare the different solution methods in order to evaluate their average efficiency. The objective of this section is twofold: provide information and benchmarks to researchers interested in developing new solution approaches and give a hands-on picture of the algorithmic landscape to practitioners having to deal with the practical solution of the problems. We performed the experiments on various sets of instances (all defined in BPP form) in order to understand which problem parameters make instances difficult to solve and which classes of problem instances are particularly hard. Benchmarks and computer codes are available in a dedicated web page.

2.7.1 Benchmarks

We used 3 different benchmarks: instances previously used in the literature, randomly generated instances, and instances especially designed so that an exact algorithm can hardly prove the optimality of a solution. All instances are downloadable from the web page `http://or.dei.unibo.it/library/bpplib` (referred to in the following as the *BPPLIB*).

Literature instances

We tested the algorithms on the instances proposed by:

- Falkenauer [119]: two classes of 80 instances each, available at Beasley's [24] OR library: the first class has uniformly distributed item sizes ('Falkenauer U') with n between 120 and 1000, and $c = 150$. The second class ('Falkenauer T') includes the so-called *triplets*, i.e., groups of three items (one large, two small) that need to be assigned to the same bin in any optimal packing, with n between 60 and 501, and $c = 1000$;

- Scholl et al. [251]: three sets of 720, 480, and 10, respectively, uniformly distributed instances (from `http://www.wiwi.uni-jena.de/entscheidung/binpp/`) with n between 50 and 500. The capacity c is between 100 and 150 (set 'Scholl 1'), equal to 1000 (set 'Scholl 2'), and equal to 100 000 (set 'Scholl 3'), respectively;
- Wäscher and Gau [288]: 17 hard instances ('Wäscher' in the tables) available at page `http://paginas.fe.up.pt/~esicup/tiki-list_file_gallery.php?galleryId=1` (which also hosts the next two sets), with n between 57 and 239, and $c = 10\,000$;
- Schwerin and Wäscher [254]: two sets ('Schwerin 1' and 'Schwerin 2') of 100 instances each with $n = 100$ and $n = 120$, respectively, and $c = 1000$;
- Schoenfield [250]: 28 instances ('Hard28') with n between 160 and 200, and $c = 1000$.

Randomly generated instances

In order to better evaluate the behavior of the exact algorithms with respect to the instance characteristics, we randomly generated BPP instances with different values of

– $n \in \{50, 100, 200, 300, 400, 500, 750, 1000\}$,

– $c \in \{50, 75, 100, 120, 125, 150, 200, 300, 400, 500, 750, 1000\}$,

– $w_{\min} \in \{0.1c, 0.2c\}$,

– $w_{\max} \in \{0.7c, 0.8c\}$,

For each quadruplet, 10 instances were obtained by uniformly randomly generating the weights in $[w_{\min}, w_{\max}]$, producing in total 3840 instances.

Difficult instances

As it is shown in Section 2.7.3, all the above instances can be solved in less than 10 minutes by at least one of the softwares we tested. In order to test them on more challenging benchmarks, we designed a new class of instances.

The *augmented Non-IRUP* (ANI) instances were derived from a benchmark, called B in the following, proposed by Caprara et al. [54]. Benchmark B (available at `http://www.or.unimore.it/resources/BPP_non_IRUP/instances.html`) consists of 15-item BPP Non-IRUP instances satisfying $\sum_{j=1}^{15} w_j = 3\,c$ (see Section 2.6.2), for which $L_{\text{LP}} = 3$ and the optimal solution has value 4. An augmented Non-IRUP instance was obtained from an instance of B by iteratively adding to it a triplet of items such that: (i) their total weight equals c, and (ii) for at least one of them, say having weight w_k, there is no subset S of items currently in the instance such that $w_k + \sum_{j \in S} w_j = c$. Whenever (ii) could not be satisfied for the current triplet, both the current capacity and the weights generated so far were doubled. We generated five sets of 50 ANI instances each, with $n \in \{201, 402, 600, 801, 1002\}$ (remind that n must be a multiple of 3). Concerning the capacities, it was imposed to the five sets that the value of c could not exceed an upper bound $\overline{c}$, respectively equal to 2 500, 10 000, 20 000, 40 000, and 80 000. Whenever this could not be ensured, the instance was discarded and a new instance was generated.

It is necessary to clarify in which sense the above ANI instances are *difficult* to solve exactly. Exact algorithms (or specifically tailored heuristics or metaheuristics) can indeed find an optimal solution, but they struggle with proving its optimality, as the solution value is higher than lower bound L_{LP}. For the sake of comparison, we also generated five sets of "easier" *augmented IRUP* (AI) instances with $n + 1$ items, obtained from the ANI ones by splitting one of the 15 original items into two items so that the resulting 16 items fit into 3 bins, i.e., the Non-IRUP is lost. For the AI instances, all bins are completely filled, so the continuous relaxation provides the optimal solution value, and the only difficulty is to construct a feasible solution having the same value.

2.7.2 Computer codes

We computationally evaluated, among the solution methods treated in the previous sections, all those for which the corresponding source code is available online, plus the classical Pascal code of Bison, provided by the authors. In addition, we included a number of methods for which the computer code can be easily implemented. The computer codes are either linked or downloadable from the BPPLIB.

We tested the following computer codes:

- Branch-and-bound (see Section 2.5.1):
  - **MTP**, Fortran code by Martello and Toth [214];
  - **BISON**, Pascal code by Scholl et al. [251];
  - **CVRPSEP**, C code by J. Lysgaard, included in a package, CVRPSEP, as a part of a separation routine for the Capacitated Vehicle Routing Problem (see

[205]). The code has been produced using the procedures of MTP. The whole package is available at `http://www.hha.dk/lys/CVRPSEP.htm`.

- Branch-and-price (see Section 2.6.3):
  - **VANCE**, C++ implementation of the algorithm by Vance et al. [282], using CPLEX 12.6.0 for the LP relaxations and the knapsack problems with additional constraints;
  - **BELOV**, C++ implementation by Belov of the algorithm by Belov and Scheithauer [30], using CPLEX 12.6.0 for the inner routines, available at web page `http://www.math.tu-dresden.de/~capad/cpd-sw.html`;
  - **SCIP-BP**, freeware SCIP 3.0.2 C code for a branch-and-price BPP algorithm, available at `http://scip.zib.de/doc/examples/Binpacking/BRANCHING.php`, that uses the Ryan and Foster [243] branching rule (also adopted in VANCE).
- Pseudo-polynomial formulations solved via ILP (see Section 2.4):
  - **ONECUT**, C++ implementation of the one-cut model by Rao [234], Dyckhoff [111], and Stadtler [266], solving the resulting ILP through CPLEX 12.6.0, available at the BPPLIB;
  - **ARCFLOW**, C++ implementation of the arc-flow model by Valério de Carvalho [278], solving the resulting ILP through CPLEX 12.6.0, available at the BPPLIB;
  - **DPFLOW**, C++ implementation of the DP-flow model by Cambazard and O'Sullivan [52], solving the resulting ILP through CPLEX 12.6.0, available at the BPPLIB;
  - **VPSOLVER**, C++ implementation by Brandão and Pedroso [45], which uses Gurobi 5.6 as inner routine, available at `https://code.google.com/p/vpsolver`.
- Other methods:
  - **BASIC ILP**, C++ implementation of the introductory ILP model (2.1)-(2.5), implemented using CPLEX 12.6.0;
  - **CSTRPROG**, C++ implementation of a simple constraint programming algorithm (Section 2.5.2), using the CP optimizer of CPLEX 12.6.0 and selecting

constraint `IloPack` (see Shaw [257]), and a search phase based on an FFD strategy.

All codes are oriented to the BPP but BELOV, ONECUT, ARCFLOW, and VPSOLVER, which are designed for the CSP. The Pascal code was compiled with `fpc (version 2.6.0-9 [2013/04/14] for x86_64)`, while the Fortran and C++ codes were all compiled with `gcc (version 4.4.7 20120313)`, using command `gfortran` and `g++`, respectively.

We preliminary computed lower and upper bounds through a simple procedure, **BFDL2**, which includes approximation algorithm BFD of Section 2.3.1 and lower bound L_2 of Section 2.3.2. The codes were only executed on instances for which lower and upper bound did not coincide. For our C++ implementations, the BFD upper bound was passed to CPLEX.

2.7.3 Experiments

All the experiments but those in Table 2.14 were executed on an Intel Xeon 3.10 GHz with 8 GB RAM, equipped with four cores. In order to allow fair comparisons with other algorithms and machines, all our experiments were performed with a single core, and the number of threads was set to one for all solvers.

Tables 2.1-2.3 give the results for the literature instances. Table 2.1 provides the results obtained by running the codes with a time limit of one minute. Columns 1 and 2 identify the benchmark and give the number of instances for which the codes were executed. The column associated with each code provides the number of such instances that were solved to proven optimality and, in parentheses, the average value of the absolute gap g between the solution value and the lower bound produced by the code. For the cases where an algorithm could solve all instances, the corresponding number appears in bold. When the time limit is very small, codes BELOV, SCIP-BP, and VPSOLVER can sometimes terminate

Table 2.1: Number of literature instances (average gap wrt lower bound) solved in less than one minute

Set	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
Falkenauer U	74	22 (1.7)	44 (0.4)	22 (1.8)	53 (1.2)	**74** (0.0)	18 (2.1)	**74** (0.0)	**74** (0.0)	37 (1.8)	**74** (0.0)	10 (2.3)	28 (2.0)
Falkenauer T	80	6 (7.0)	42 (0.5)	0 (11.0)	76 (0.1)	57 (0.3)	35 (4.5)	**80** (0.0)	**80** (0.0)	40 (8.8)	**80** (0.0)	7 (7.0)	39 (8.8)
Scholl 1	323	242 (0.3)	288 (0.1)	223 (0.3)	**323** (0.0)	**323** (0.0)	244 (0.2)	**323** (0.0)	**323** (0.0)	289 (0.1)	**323** (0.0)	212 (0.3)	90 (0.6)
Scholl 2	244	130 (0.6)	233 (0.0)	65 (1.4)	204 (0.2)	**244** (0.0)	67 (1.2)	118 (0.4)	202 (0.1)	58 (1.3)	208 (0.1)	90 (1.0)	122 (1.3)
Scholl 3	10	0 (1.5)	3 (0.7)	0 (4.1)	**10** (0.0)	**10** (0.0)	0 (4.1)	0 (4.1)	0 (4.1)	0 (4.1)	**10** (0.0)	0 (2.7)	0 (4.1)
Wäscher	17	0 (1.0)	10 (0.4)	0 (1.0)	6 (0.6)	**17** (0.0)	0 (1.0)	0 (1.0)	0 (1.0)	0 (1.0)	6 (0.6)	4 (0.8)	7 (0.6)
Schwerin 1	100	15 (0.9)	**100** (0.0)	9 (0.9)	**100** (0.0)	**100** (0.0)	0 (1.0)	**100** (0.0)	**100** (0.0)	0 (1.0)	**100** (0.0)	32 (0.7)	**100** (0.0)
Schwerin 2	100	4 (1.4)	63 (0.4)	0 (1.4)	**100** (0.0)	**100** (0.0)	0 (1.4)	**100** (0.0)	**100** (0.0)	0 (1.4)	**100** (0.0)	36 (0.7)	60 (0.8)
Hard28	28	0 (1.0)	0 (1.0)	0 (1.0)	11 (0.6)	**28** (0.0)	7 (0.8)	6 (0.8)	16 (0.4)	0 (1.0)	27 (0.0)	0 (1.0)	0 (1.0)
Total	976	419 (0.9)	783 (0.1)	319 (1.4)	883 (0.1)	953 (0.0)	371 (1.0)	801 (0.2)	895 (0.1)	424 (1.2)	928 (0.0)	391 (1.0)	446 (1.3)

Table 2.2: Average time in seconds (standard deviation) for solving literature instances

Set	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
Falkenauer U	74	42.8 (27.2)	24.5 (29.5)	42.2 (27.6)	24.1 (25.2)	0.0 (0.0)	50.1 (19.1)	0.2 (0.1)	0.2 (0.1)	38.8 (23.9)	0.1 (0.0)	61.4 (41.2)	38.8 (27.9)
Falkenauer T	80	55.5 (15.9)	30.6 (29.3)	60.2 (0.3)	14.8 (19.2)	24.7 (26.8)	39.4 (25.6)	8.7 (10.7)	3.5 (6.8)	41.7 (22.0)	0.4 (0.5)	58.1 (8.9)	34.2 (27.6)
Scholl 1	323	15.1 (26.0)	7.0 (18.8)	19.4 (27.6)	3.6 (7.5)	0.0 (0.0)	22.4 (24.3)	0.1 (0.1)	0.1 (0.3)	13.0 (19.1)	0.1 (0.1)	23.1 (28.3)	44.3 (25.9)
Scholl 2	244	28.2 (29.9)	3.0 (12.7)	44.2 (26.4)	18.6 (24.3)	0.3 (0.4)	49.2 (20.2)	38.7 (25.6)	18.9 (23.1)	50.4 (19.4)	14.0 (21.5)	40.7 (27.2)	31.7 (29.0)
Scholl 3	10	60.0 (0.0)	42.0 (29.0)	60.0 (0.0)	1.9 (0.8)	14.1 (1.5)	60.0 (0.0)	63.9 (3.1)	61.1 (0.3)	60.0 (0.0)	6.3 (3.9)	60.0 (0.0)	60.0 (0.0)
Wäscher	17	60.0 (0.0)	24.7 (30.4)	60.0 (0.0)	52.0 (18.9)	0.1 (0.1)	60.0 (0.1)	60.7 (0.2)	60.5 (0.3)	60.0 (0.0)	49.4 (26.6)	49.9 (19.3)	37.2 (28.4)
Schwerin 1	100	51.1 (21.3)	0.0 (0.0)	55.4 (15.6)	0.3 (0.0)	1.0 (0.3)	60.1 (0.0)	13.1 (9.5)	1.5 (0.6)	59.6 (0.3)	0.3 (0.2)	43.0 (25.8)	4.4 (7.4)
Schwerin 2	100	57.6 (11.8)	22.2 (29.1)	60.0 (0.0)	0.5 (0.1)	1.4 (0.3)	60.1 (0.0)	11.7 (7.8)	1.5 (0.7)	59.6 (0.3)	0.3 (0.1)	43.1 (25.3)	27.1 (27.8)
Hard28	28	60.0 (0.0)	60.0 (0.0)	60.0 (0.0)	48.9 (20.8)	7.3 (11.9)	51.2 (16.8)	54.6 (11.4)	40.6 (20.0)	60.0 (0.0)	14.2 (17.9)	60.0 (0.0)	60.0 (0.0)
Total	976	34.4 (22.8)	12.3 (17.9)	40.8 (19.5)	11.3 (13.0)	2.7 (2.7)	42.2 (17.1)	16.3 (9.5)	8.2 (7.2)	38.9 (14.8)	5.0 (6.5)	39.3 (25.6)	34.6 (24.3)

Table 2.3: Number of literature instances solved in less than ten minutes

Set	tested inst.	BISON	BELOV	ARCFLOW	VPSOLVER
Falkenauer U	74	50	**74**	**74**	**74**
Falkenauer T	80	47	**80**	**80**	**80**
Scholl 1	323	290	**323**	**323**	**323**
Scholl 2	244	234	**244**	231	242
Scholl 3	10	3	**10**	0	**10**
Wäscher	17	10	**17**	4	13
Schwerin 1	100	**100**	**100**	**100**	**100**
Schwerin 2	100	63	**100**	**100**	**100**
Hard28	28	0	**28**	26	26
Total	976	797	**976**	938	968

without producing a decent lower and/or upper bound. In such cases the value of g could be huge or undefined, so, in order to avoid anomalous results, the gap was always computed as the minimum between g and the gap produced by BFDL2. The last line of the table reports the total number of solved instances and, in parentheses, the overall average gap.

Table 2.2 has the same structure as Table 2.1 but the entries provide, for each computer code, the average CPU time expressed in seconds and, in parentheses, the corresponding standard deviation. The entries in the last line give in this case the average CPU time and, in parentheses, the standard deviation computed over all instances for which the code was executed. It must be observed that, for the computer codes that invoke CPLEX, SCIP, or Gurobi, the actual CPU time spent on an instance turns out, in some cases, to be greater than the time limit. In most cases the difference was irrelevant but for BASIC ILP. Indeed, when solving model (2.1)-(2.5), CPLEX can get stuck in the cutting plane loop, which needs a high time and cannot be interrupted freely. This explains a couple of average times higher that 60 seconds (in Tables 2.2 and 2.7). The instances that required a CPU time much larger than 60 seconds were counted as unsolved by BASIC ILP (while the improvement coming from the additional CPU time spent turned out to be irrelevant).

Tables 2.1 and 2.2 show that, for the literature instances,

1. among the (old) branch-and-bound codes, BISON is the only one capable of solving many instances;
2. two branch-and-price algorithms (VANCE and, in particular, BELOV) have satisfactory results, while SCIP-BP does not appear to be competitive. The only difficult instances for BELOV appear to be Falkenauer T, which are instead easily solved by the pseudo-polynomial models, probably because of the small capacities involved;
3. ARCFLOW and VPSOLVER are the most efficient algorithms among those that use pseudo-polynomial models. ONECUT, even if based on an older model, has an overall decent performance. We additionally observe that the computational experiments showed that its LP relaxation has the same quality as the LP relaxation of ARCFLOW;
4. as it could be expected, the efficiency of BASIC ILP and CSTRPROG is quite low.

We selected the winner of each algorithmic class (BISON, BELOV, and VPSOLVER) for an additional round of tests (on the same instances) with a time limit of 10 minutes. By considering that the performance of ARCFLOW is competitive with that of VPSOLVER, and that its graph construction is considerably simpler, we decided to include it in this round. The number of solved instances within the larger time limit are provided in Table 2.3. Overall, the four algorithms exhibited a satisfactory behavior. In particular BELOV solved all instances and VPSOLVER almost all of them. ARCFLOW and BISON solved 96% and 82% of the instances, respectively. Instances with very large capacity values turned out to be particularly hard for ARCFLOW.

The next group of six tables refers to the randomly generated instances. Tables 2.4-2.6 provide the number of instances solved by each computer code (and, in parentheses, the gap), with a time limit of one minute, when varying the items characteristics. In Table 2.4 the results are listed according to the number of items, in Table 2.5 according to the capacity, and in Table 2.6 according to the weight over capacity ratios. The entries give the same information as in Table 2.1. Similarly, Tables 2.7-2.9 report average CPU times and standard deviations with the same grouping policy. Globally, the results confirm observations 1.-4. made for the literature instances. We additionally observe that:

5. BELOV solved all instances within one minute, and it appears to be clearly superior to all other codes but VPSOLVER, which solved just 10 instances less (out of the 2901 instances for which the initial lower and upper bound did not coincide);

6. SCIP-BP is effective on small-size instances ($n \leq 100$);

7. the performance of branch-and-price algorithms is not affected by the capacity, while that of algorithms based on pseudo-polynomial models is. In particular, the behavior of ARCFLOW and DPFLOW depends on the three considered parameters, especially on the capacity and the item weights.

Overall, the best-in-class algorithms turned out to be the same as for the previous benchmark.

Moreover, additional computational experiments with different weight ranges (0.1/0.4, 0.1/0.5, 0.2/0.4, 0.2/0.5) produced similar results and the same ranking of the algorithms. In this case too, we created 3840 instances. We ran the codes, for one minute, on the 3403 instances for which the initial lower and upper bound did not coincide. BELOV solved all of them but one, while VPSOLVER, ARCFLOW, and ONECUT solved 3346, 3283, and 3027 instances, respectively. (BELOV and VPSOLVER solved however all of them in less than ten minutes.)

Table 2.4: Number of random instances solved in less than one minute (average gap wrt lower bound) when varying n.

n	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
50	165	163 (0.0)	**165** (0.0)	164 (0.0)	**165** (0.0)	**165** (0.0)	**165** (0.0)	**165** (0.0)	**165** (0.0)	**165** (0.0)	**165** (0.0)	157 (0.0)	71 (0.7)
100	271	243 (0.1)	257 (0.1)	239 (0.1)	**271** (0.0)	**271** (0.0)	**271** (0.0)	**271** (0.0)	**271** (0.0)	**271** (0.0)	**271** (0.0)	237 (0.1)	132 (0.6)
200	359	237 (0.4)	290 (0.2)	220 (0.6)	358 (0.0)	**359** (0.0)	293 (0.2)	358 (0.0)	**359** (0.0)	292 (0.2)	**359** (0.0)	201 (0.4)	171 (0.8)
300	393	166 (0.8)	265 (0.3)	144 (1.1)	387 (0.0)	**393** (0.0)	155 (0.8)	385 (0.0)	391 (0.0)	243 (0.6)	**393** (0.0)	115 (0.8)	140 (1.2)
400	425	151 (1.1)	244 (0.5)	138 (1.4)	416 (0.0)	**425** (0.0)	114 (1.1)	408 (0.1)	421 (0.0)	193 (1.1)	**425** (0.0)	92 (1.0)	104 (1.7)
500	414	121 (1.4)	208 (0.6)	128 (1.6)	394 (0.0)	**414** (0.0)	69 (1.7)	394 (0.1)	402 (0.0)	169 (1.3)	413 (0.0)	60 (1.5)	61 (2.0)
750	433	93 (2.0)	214 (0.7)	98 (2.3)	99 (2.1)	**433** (0.0)	22 (2.7)	401 (0.2)	415 (0.1)	120 (2.0)	431 (0.0)	54 (2.5)	23 (2.8)
1000	441	78 (2.6)	196 (0.8)	73 (3.1)	62 (2.8)	**441** (0.0)	0 (3.6)	407 (0.2)	416 (0.1)	67 (3.1)	434 (0.0)	39 (3.3)	7 (3.6)
Overall	2901	1252 (1.2)	1839 (0.5)	1204 (1.5)	2152 (0.8)	**2901** (0.0)	1089 (1.5)	2789 (0.1)	2840 (0.0)	1520 (1.2)	2891 (0.0)	955 (1.4)	709 (1.9)

Table 2.5: Number of random instances solved in less than one minute (average gap wrt lower bound) when varying c.

c	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
50	223	125 (0.5)	191 (0.1)	145 (0.6)	162 (0.4)	**223** (0.0)	86 (0.8)	**223** (0.0)	**223** (0.0)	205 (0.1)	**223** (0.0)	83 (0.8)	73 (0.9)
75	240	137 (0.7)	187 (0.2)	141 (0.8)	176 (0.5)	**240** (0.0)	96 (1.0)	**240** (0.0)	**240** (0.0)	208 (0.2)	**240** (0.0)	92 (0.9)	84 (1.2)
100	234	111 (0.8)	177 (0.2)	116 (1.0)	172 (0.6)	**234** (0.0)	89 (1.1)	**234** (0.0)	**234** (0.0)	185 (0.3)	**234** (0.0)	71 (1.0)	64 (1.3)
120	241	110 (0.8)	172 (0.3)	112 (1.0)	181 (0.5)	**241** (0.0)	91 (1.1)	**241** (0.0)	**241** (0.0)	168 (0.5)	**241** (0.0)	82 (1.0)	66 (1.3)
125	251	127 (0.8)	176 (0.3)	129 (1.0)	192 (0.5)	**251** (0.0)	101 (1.1)	**251** (0.0)	**251** (0.0)	174 (0.6)	**251** (0.0)	84 (1.1)	77 (1.3)
150	240	101 (0.9)	165 (0.3)	90 (1.2)	181 (0.6)	**240** (0.0)	95 (1.1)	**240** (0.0)	**240** (0.0)	143 (0.8)	**240** (0.0)	72 (1.1)	57 (1.5)
200	246	95 (1.0)	156 (0.3)	89 (1.2)	184 (0.6)	**246** (0.0)	99 (1.1)	**246** (0.0)	**246** (0.0)	127 (0.9)	**246** (0.0)	74 (1.1)	53 (1.5)
300	237	86 (1.0)	134 (0.4)	77 (1.2)	172 (0.6)	**237** (0.0)	80 (1.2)	**237** (0.0)	**237** (0.0)	79 (1.3)	**237** (0.0)	67 (1.2)	59 (1.5)
400	245	96 (1.1)	122 (0.5)	81 (1.4)	184 (0.6)	**245** (0.0)	95 (1.3)	**245** (0.0)	**245** (0.0)	71 (1.5)	**245** (0.0)	80 (1.2)	51 (1.6)
500	243	90 (1.1)	125 (0.5)	76 (1.4)	179 (0.6)	**243** (0.0)	77 (1.3)	241 (0.0)	242 (0.0)	56 (1.6)	**243** (0.0)	79 (1.2)	45 (1.6)
750	249	82 (1.1)	119 (0.6)	70 (1.4)	183 (0.6)	**249** (0.0)	91 (1.3)	211 (0.2)	229 (0.1)	55 (1.6)	**249** (0.0)	84 (1.2)	36 (1.7)
1000	252	92 (1.1)	115 (0.6)	78 (1.4)	186 (0.6)	**252** (0.0)	89 (1.3)	180 (0.5)	212 (0.3)	49 (1.6)	242 (0.0)	87 (1.2)	44 (1.6)
Overall	2901	1252 (0.9)	1839 (0.3)	1204 (1.1)	2152 (0.6)	**2901** (0.0)	1089 (1.1)	2789 (0.1)	2840 (0.0)	1520 (0.9)	2891 (0.0)	955 (1.1)	709 (1.4)

Table 2.6: Number of random instances solved in less than one minute (average gap wrt lower bound) when varying weight range.

Range	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
0.1 / 0.7	785	337 (0.7)	590 (0.2)	385 (0.6)	541 (0.6)	**785** (0.0)	162 (1.1)	700 (0.2)	737 (0.1)	274 (1.0)	776 (0.0)	106 (1.1)	384 (0.8)
0.1 / 0.8	729	222 (0.7)	339 (0.5)	242 (0.7)	476 (0.5)	**729** (0.0)	233 (0.9)	703 (0.1)	716 (0.0)	328 (0.8)	728 (0.0)	197 (0.9)	171 (1.0)
0.2 / 0.7	878	229 (2.2)	406 (0.7)	206 (2.9)	646 (1.2)	**878** (0.0)	340 (2.3)	877 (0.0)	**878** (0.0)	569 (1.6)	**878** (0.0)	281 (2.1)	116 (3.0)
0.2 / 0.8	509	464 (0.1)	504 (0.0)	371 (0.3)	489 (0.0)	**509** (0.0)	354 (0.3)	**509** (0.0)	**509** (0.0)	349 (0.3)	**509** (0.0)	371 (0.2)	38 (0.9)
Overall	2901	1252 (0.9)	1839 (0.3)	1204 (1.1)	2152 (0.6)	**2901** (0.0)	1089 (1.1)	2789 (0.1)	2840 (0.0)	1520 (0.9)	2891 (0.0)	955 (1.1)	709 (1.4)

Table 2.7: Average time in seconds (standard dev.) for solving random instances when varying n.

n	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
50	165	0.8 (6.6)	0.0 (0.0)	0.4 (4.7)	0.0 (0.1)	0.0 (0.0)	0.9 (0.7)	0.1 (0.3)	0.1 (0.1)	0.5 (0.8)	0.0 (0.0)	4.5 (14.8)	34.8 (29.4)
100	271	7.4 (18.9)	3.8 (13.7)	8.4 (19.7)	0.1 (0.2)	0.0 (0.0)	4.6 (7.1)	0.8 (2.5)	0.3 (0.4)	5.0 (7.5)	0.1 (0.1)	9.4 (20.5)	31.4 (29.6)
200	359	21.6 (28.3)	12.0 (23.7)	25.0 (28.9)	1.1 (4.1)	0.0 (0.0)	22.6 (21.8)	2.4 (7.0)	0.8 (2.6)	21.0 (22.1)	0.3 (0.9)	29.4 (28.8)	33.0 (29.0)
300	393	35.7 (29.1)	20.7 (28.2)	38.7 (28.3)	4.3 (8.0)	0.1 (0.2)	44.1 (21.6)	4.5 (11.6)	2.0 (6.3)	33.9 (23.9)	0.6 (1.4)	45.4 (24.4)	41.6 (26.0)
400	425	39.1 (28.4)	26.1 (29.5)	41.2 (27.4)	9.3 (10.2)	0.2 (0.3)	49.8 (17.7)	5.1 (13.3)	3.0 (8.7)	42.4 (22.0)	0.8 (2.0)	49.5 (21.4)	47.7 (22.7)
500	414	43.0 (26.7)	30.3 (29.8)	42.6 (26.7)	19.2 (14.1)	0.2 (0.5)	55.1 (12.1)	6.3 (14.8)	4.0 (11.2)	44.8 (20.4)	1.7 (6.4)	53.5 (18.7)	53.1 (17.3)
750	433	47.3 (24.4)	30.9 (29.9)	47.3 (24.2)	50.4 (21.6)	0.4 (1.0)	59.5 (2.6)	7.8 (17.2)	6.0 (14.3)	52.6 (14.3)	2.4 (7.1)	59.0 (23.3)	58.0 (9.6)
1000	441	49.5 (22.7)	33.9 (29.5)	50.8 (21.3)	52.4 (20.5)	0.7 (1.8)	60.0 (0.0)	8.1 (17.4)	6.8 (15.6)	56.4 (10.0)	3.4 (10.3)	90.4 (57.4)	59.2 (6.2)
Overall	2901	34.7 (29.4)	22.6 (28.8)	36.0 (28.9)	20.3 (25.3)	0.2 (0.9)	42.4 (24.4)	5.0 (13.4)	3.3 (10.4)	36.7 (25.1)	1.4 (5.6)	48.4 (39.0)	46.9 (23.9)

Table 2.8: Average time in seconds (standard deviation) for solving random instances when varying c.

c	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
50	223	27.2 (29.5)	9.4 (21.5)	23.2 (27.9)	20.2 (25.5)	0.0 (0.0)	43.0 (23.5)	0.0 (0.0)	0.0 (0.0)	13.6 (17.7)	0.0 (0.0)	48.7 (44.7)	42.6 (25.9)
75	240	26.1 (29.6)	14.8 (25.5)	26.4 (29.2)	20.2 (25.5)	0.0 (0.0)	42.0 (24.3)	0.0 (0.0)	0.0 (0.0)	19.5 (20.7)	0.0 (0.0)	53.2 (58.9)	41.1 (26.7)
100	234	32.0 (29.7)	14.9 (25.7)	31.1 (29.6)	20.3 (25.2)	0.0 (0.0)	42.8 (24.3)	0.0 (0.0)	0.1 (0.0)	26.4 (23.1)	0.0 (0.0)	50.5 (38.0)	45.0 (25.2)
120	241	33.5 (29.5)	17.8 (27.1)	33.7 (29.3)	19.7 (25.0)	0.0 (0.0)	42.2 (24.5)	0.1 (0.1)	0.1 (0.1)	29.6 (24.2)	0.1 (0.0)	49.8 (40.7)	45.1 (25.0)
125	251	30.0 (29.8)	18.0 (27.5)	30.0 (29.6)	19.2 (24.8)	0.0 (0.0)	41.0 (25.1)	0.1 (0.1)	0.1 (0.1)	30.9 (24.6)	0.1 (0.1)	50.4 (45.4)	44.3 (25.3)
150	240	35.1 (29.4)	19.3 (27.8)	38.2 (28.6)	19.5 (24.9)	0.0 (0.0)	41.8 (24.5)	0.1 (0.1)	0.2 (0.1)	34.9 (24.2)	0.1 (0.1)	50.9 (40.7)	47.3 (23.6)
200	246	37.6 (28.8)	22.8 (28.8)	39.2 (28.1)	20.5 (25.4)	0.0 (0.0)	40.8 (25.1)	0.3 (0.3)	0.3 (0.4)	39.3 (24.1)	0.2 (0.2)	51.4 (39.7)	48.7 (22.3)
300	237	39.0 (28.3)	26.8 (29.5)	41.4 (27.2)	21.9 (25.5)	0.1 (0.3)	44.4 (23.7)	1.5 (2.0)	1.3 (2.2)	45.6 (22.6)	0.6 (0.7)	48.1 (29.0)	46.5 (24.2)
400	245	36.7 (29.2)	30.5 (29.8)	40.4 (28.0)	20.3 (25.4)	0.2 (0.1)	41.6 (24.8)	3.9 (5.7)	2.8 (5.7)	47.4 (21.7)	1.1 (1.9)	47.0 (31.8)	48.8 (22.6)
500	243	38.3 (28.6)	29.5 (29.8)	41.8 (27.3)	20.7 (25.4)	0.3 (0.4)	44.8 (23.8)	8.6 (11.6)	5.1 (10.2)	49.2 (20.9)	1.9 (4.5)	45.7 (31.0)	50.0 (21.6)
750	249	40.8 (27.8)	32.0 (29.7)	43.4 (26.7)	21.0 (25.8)	0.8 (1.2)	42.4 (24.5)	18.6 (22.1)	11.2 (17.2)	49.9 (20.2)	4.3 (8.2)	43.1 (28.6)	52.0 (19.9)
1000	252	39.0 (28.4)	33.1 (29.8)	41.9 (27.4)	20.7 (25.6)	1.3 (2.3)	42.5 (24.8)	25.3 (25.1)	17.6 (21.9)	51.5 (18.9)	7.4 (14.8)	42.1 (27.5)	50.5 (21.3)
Overall	2901	34.7 (29.4)	22.6 (28.8)	36.0 (28.9)	20.3 (25.3)	0.2 (0.9)	42.4 (24.4)	5.0 (13.4)	3.3 (10.4)	36.7 (25.1)	1.4 (5.6)	48.4 (39.0)	46.9 (23.9)

Table 2.9: Average time in seconds (standard deviation) for solving random instances when varying weight range.

Range	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
0.1 / 0.7	785	35.1 (29.1)	15.3 (25.9)	31.9 (29.3)	25.3 (25.2)	0.2 (0.4)	50.6 (19.9)	11.0 (19.7)	7.3 (16.2)	45.5 (22.2)	3.5 (9.8)	60.6 (35.1)	33.7 (28.1)
0.1 / 0.8	729	42.2 (27.2)	33.3 (29.4)	41.3 (27.2)	25.7 (26.5)	0.2 (0.6)	45.0 (23.5)	5.9 (13.7)	3.9 (10.3)	40.8 (23.9)	1.2 (3.8)	48.7 (28.5)	47.7 (23.2)
0.2 / 0.7	878	44.8 (25.8)	32.8 (29.6)	46.5 (24.7)	20.8 (25.6)	0.3 (1.4)	41.2 (25.3)	1.7 (5.5)	1.0 (2.9)	30.2 (25.3)	0.3 (0.6)	53.1 (47.5)	52.8 (19.0)
0.2 / 0.8	509	5.8 (17.4)	0.7 (6.2)	16.8 (26.7)	4.2 (13.8)	0.2 (0.4)	28.4 (24.3)	0.3 (0.7)	0.4 (1.3)	28.7 (25.0)	0.2 (0.3)	20.8 (25.8)	55.9 (14.9)
Overall	2901	34.7 (29.4)	22.6 (28.8)	36.0 (28.9)	20.3 (25.3)	0.2 (0.9)	42.4 (24.4)	5.0 (13.4)	3.3 (10.4)	36.7 (25.1)	1.4 (5.6)	48.4 (39.0)	46.9 (23.9)

All algorithms were also evaluated, on a subset of the instances of Tables 2.4-2.9, with respect to the average *item multiplicity*, computed as $\mu = n/m$, where m is the number of item types in the equivalent minimal CSP instance (see Section 2.2). We considered instances with capacity not greater than 150, as for higher capacities μ turned out to always be very small. The results are reported in Table 2.10 and, as usual, refer to instances for which the initial lower and upper bound did not coincide. As it could be expected, the methods devoted to the solution of the CSP are unaffected by the item multiplicity. Indeed, BELOV, ONECUT, ARCFLOW, and VPSOLVER can solve all instances to proven optimality within one minute. Instead, the performance of all other algorithms (which are devoted to the BPP) gets worse when the value of μ increases. This is particularly evident for SCIP-BP, that can solve to optimality all instances with $\mu \leq 2$, but less than one tenth of those with $\mu \geq 10$.

In Table 2.11 (the counterpart of Table 2.3) we consider again the instances studied in Tables 2.4-2.9, and show the results obtained by the four best codes within a time limit of 10 minutes, grouped by number of items. The results confirm the algorithms' ranking.

As all the considered instances were solved to optimality, we used the four selected algorithms for a set of experiments on the new, difficult, ANI instances we have described in Section 2.7.1. In this case, each algorithm was given a time limit of one hour per instance. The outcome of the experiments is reported in Tables 2.12 (number of solved instances and average absolute gap with respect to the lower bound) and 2.13 (average CPU time and standard deviation). The results confirm that the ANI instances are not solved satisfactorily: even BELOV, which closed all other instances, was unable to solve them to proven optimality. It turns out that the AI instances as well look quite hard, although a good heuristic, specially tailored wrt the special structure of these instances, is likely to find an optimal solution (whose optimality could then easily be proved).

Overall, our experiments show that, among the algorithms we tested, BELOV and VPSOLVER are the best ones. As both use quite complex tools, ARCFLOW can be seen as a reasonable compromise between simplicity and performance. Basic ILP and SCIP-BP can be used for small instances. Branch-and-bound algorithms MTP, CVRPSEP and, in particular, BISON can be an alternative when one wants to avoid the use of solvers. CSTRPROG is generally inefficient, but it has the advantage of easily allowing additional constraints. Although ONECUT is based on a very old model, it is competitive with much more recent approaches. Among the approaches based on pseudo-polynomial models, DPFLOW has mainly theoretical interest, but has the advantage of being easily understan-

Table 2.10: Number of random instances solved in less than one minute when varying the average item multiplicity μ.

μ	tested inst.	Branch-and-bound			Branch-and-price			Pseudo-polynomial				Others	
		MTP	BISON	CVRPSEP	VANCE	BELOV	SCIP-BP	ONECUT	ARCFLOW	DPFLOW	VPSOLVER	BASIC ILP	CSTRPROG
$[1, 2)$	138	133	137	136	**138**	**138**	**138**	**138**	**138**	**138**	**138**	127	69
$[2, 3)$	119	104	113	103	**119**	**119**	109	**119**	**119**	**119**	**119**	81	68
$[3, 5)$	251	154	199	144	249	**251**	141	**251**	**251**	221	**251**	96	104
$[5, 10)$	458	192	316	198	390	**458**	133	**458**	**458**	333	**458**	115	122
$[10, n]$	463	128	303	152	168	**463**	37	**463**	**463**	272	**463**	65	58
Total	1429	711	1068	733	1064	**1429**	558	**1429**	**1429**	1083	**1429**	484	421

Table 2.11: Number of random instances solved in less than ten minutes when varying n.

n	tested inst.	BISON	BELOV	ARCFLOW	VPSOLVER
50	165	**165**	**165**	**165**	**165**
100	271	261	**271**	**271**	**271**
200	359	299	**359**	**359**	**359**
300	393	269	**393**	**393**	**393**
400	425	250	**425**	**425**	**425**
500	414	212	**414**	**414**	**414**
750	433	217	**433**	431	**433**
1000	441	200	**441**	434	**441**
Total	2901	1873	**2901**	2892	**2901**

Table 2.12: Number of difficult instances (ANI) solved in less than 1 hour (average gap wrt lower bound). The AI instances are included for the sake of comparison.

n(ANI)	n(AI)	$\overline{c}$	BISON		BELOV		ARCFLOW		VPSOLVER	
			ANI	*AI*	ANI	*AI*	ANI	*AI*	ANI	*AI*
201	202	2500	0 (1.0)	*3 (0.9)*	**50** (0.0)	***50*** *(0.0)*	16 (0.7)	*44 (0.1)*	47 (0.1)	***50*** *(0.0)*
402	403	10000	0 (1.0)	*0 (1.0)*	1 (1.0)	*45 (0.1)*	0 (1.0)	*0 (1.0)*	6 (0.9)	*42 (0.2)*
600	601	20000	-	-	0 (1.0)	*21 (0.6)*	-	-	0 (1.0)	*8 (0.8)*
801	802	40000	-	-	0 (1.0)	*0 (1.0)*	-	-	0 (1.0)	*0 (1.0)*
1002	1003	80000	-	-	-	-	-	-	-	-
	Overall		0 (1.0)	*3 (1.0)*	51 (0.7)	*116 (0.4)*	16 (0.8)	*44 (0.6)*	53 (0.7)	*100 (0.5)*

Table 2.13: Average time in seconds (standard deviation) for solving difficult instances (ANI). The AI instances are included for the sake of comparison.

n(ANI)	n(AI)	$\overline{c}$	BISON		BELOV		ARCFLOW		VPSOLVER	
			ANI	*AI*	ANI	*AI*	ANI	*AI*	ANI	*AI*
201	202	2500	3600 (0)	*3384 (862)*	144 (119)	*91 (119)*	2723 (1376)	*964 (1099)*	415 (1056)	*54 (128)*
402	403	10000	3600 (0)	*3600 (0)*	3556 (321)	*699 (1043)*	3601 (0)	*3601 (0)*	3304 (846)	*1130 (1201)*
600	601	20000	-	-	3602 (3)	*2539 (1321)*	-	-	3600 (0)	*3509 (293)*
801	802	40000	-	-	3602 (5)	*3601 (5)*	-	-	3600 (0)	*3600 (0)*
1002	1003	80000	-	-	-	-	-	-	-	-
Overall			3600 (0)	*3492 (616)*	2726 (1504)	*1733 (1639)*	1581 (1064)	*2943 (1269)*	2730 (1504)	*2073 (1653)*

-dable. Among branch-and-price algorithms, VANCE has mainly historical interests, but it has an acceptable performance.

A final relevant observation concerns the fact that, in the past, the pseudo-polynomial models were not seen as realistic solution approaches because of the huge number of constraints and variables they involve, and hence they were rarely directly used in practice as ILP formulations. Our results show that they turn out to be extremely competitive today. This phenomenon is explained by Table 2.14, produced thanks to IBM CPLEX, which compares the performance of eight versions of the code (from CPLEX 6.0, dated 1998, to CPLEX 12.6.0, dated 2013) in the solution of the ILPs produced by ARCFLOW for 20 selected random instances. The instances had n ranging between 300 and 1000, and c ranging between 400 and 1000. The resulting ILPs had a number of rows (resp. columns) ranging between 482 and 1093 (resp. between 32 059 and 111 537). Each CPLEX version was run on a single core of an Intel Xeon Processor E5430 running at 2.66 GHz and equipped with 24 GB of memory, both with a time limit of 10 minutes and a time limit of one hour. The entries provide the number of instances solved to proven optimality and, in square brackets, the average CPU time.

Table 2.14: Number of selected instances solved [average time in seconds] using different versions of CPLEX.

Time	tested inst.	6.0 (1998)	7.0 (1999)	8.0 (2002)	9.0 (2003)	10.0 (2006)	11.0 (2007)	12.1 (2009)	12.6.0 (2013)
10 minutes	20	13 [366]	10 [420]	5 [570]	17 [268]	19 [162]	**20** [65]	19 [117]	**20** [114]
60 minutes	20	16 [897]	15 [1210]	15 [2009]	**20** [343]	**20** [186]	**20** [65]	19 [267]	**20** [114]

The results in the first line show that, up to the early noughties, only a relatively small number of these instances could be solved within ten minutes, while the recent versions are very effective. The “irregular” behavior of the solver (previous versions give sometimes better results) is only apparently surprising. It is indeed known (see, e.g., Lodi [192] or Achterberg and Wunderling [1]) that, on specific instances, an older version of CPLEX can beat a newer one. In our case, the experiments were made on a small set of instances of a specific problem, so a fortiori irregularities could be expected. The second line shows that, in one CPU hour, about 75% of our instances could be solved prior to 2003 while the subsequent versions could solve practically all of them. The number of solved instances is in this case much more regular, but the average CPU time is not: compare, e.g., versions 11.0 and 12.6.0. Overall, by considering that ONECUT was developed in the mid-seventies, and ARCFLOW in the late nineties, these results well explain on one hand the choice of not pursuing their direct use, and on the other hand the good computational performance obtained nowadays.

2.8 Conclusions

We have reviewed the mathematical models and the exact algorithms developed in the last fifty years for one of the most famous combinatorial optimization problems. The bin packing problem and its main generalization (the cutting stock problem) have attracted many researchers, whose contributions have accompanied the development of algorithmic tools for the exact solution of combinatorial optimization problems. We have discussed the main approaches proposed in the literature, and we have provided an experimental evaluation of the available software on different classes of benchmarks, including a newly developed class of instances for which the exact algorithms can hardly obtain a provably optimal solution. We have additionally evaluated the influence that the improvement of ILP solvers has had on the performance of pseudo-polynomial formulations. The tested software and the benchmarks are now available in a dedicated library. Our study also shows that there is room for future research. While many classes of instances are more or less closed, there are still benchmarks (the AI and the ANI instances) which are not satisfactorily solved by the best available algorithms, even in the case of moderate sizes. In addition, branch-and-cut-and-price appears today to be the most effective approach, but the improving computational power of the ILP solvers could stimulate new algorithmic research lines on pseudo-polynomial methods. From a theoretical point of view, a relevant issue concerns the MIRUP conjecture, which is still open. We hope that our picture will stimulate future research in this fascinating area.

Chapter 3

BPPLIB: A Library for Bin Packing and Cutting Stock Problems

[1]

In this chapter, we present a library of computer codes, benchmark instances, and pointers to relevant surveys for these two problems. The computer code section includes twelve programs: seven are directly downloadable from the library page, while for the remaining five we provide addresses where they can be obtained or downloaded. Some of the codes for which we provide an original C++ implementation need an ILP solver. For such cases, the library provides two versions: one that uses the commercial solver Cplex, and one that uses the freeware solver SCIP. The benchmark section provides over six thousands instances (partly coming from the literature and partly randomly generated), together with the corresponding solutions. Instances that are difficult to solve to proven optimality are included. The library also includes a BibTeX file of more than 150 references on this topic and an interactive visual tool to manually solve bin packing and cutting stock instances. We conclude this chapter by reporting the results of new computational experiments on a number of computer codes and benchmark instances.

Keywords: Bin packing, Cutting stock, Computer codes, Benchmark instances, Surveys.

3.1 Introduction

In the *bin packing problem* (BPP), n *items* of given integer *weight* w_j $(j = 1, \dots, n)$ have to be packed into the minimum number of identical containers (*bins*) of integer *capacity*

[1]The results of this chapter appears in: M. Delorme, M. Iori, and S. Martello, BPPLIB: A Library for Bin Packing and Cutting Stock Problems, *Technical Report*, 2016 [99].

c. Let u be any upper bound on the solution value. Let us introduce two sets of binary variables: y_i $(i = 1, \dots, u)$, taking the value one if and only if bin i is used in the solution, and x_{ij} $(i = 1, \dots, u; j = 1, \dots, n)$, taking the value one if and only if item j is packed into bin i. A possible simple *Integer Linear Programming* (ILP) model of the problem is then (see Martello and Toth [214])

$$\min \sum_{i=1}^{u} y_i \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad (i = 1, \dots, u), \tag{3.2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \quad (j = 1, \dots, n), \tag{3.3}$$

$$y_i \in \{0, 1\} \quad (i = 1, \dots, u), \tag{3.4}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \dots, u; j = 1, \dots, n). \tag{3.5}$$

Among the many variants and generalizations of the problem, the most intensively studied is probably the *Cutting Stock Problem* (CSP). In this case, instead of single items, we have m *item types* of weight w_j and an integer *demand* d_j $(j = 1, \dots, m)$ per item type. The objective is to pack d_j copies of each item type j into the minimum number of bins. By introducing an additional set of integer variables ξ_{ij} $(i = 1, \dots, u; j = 1, \dots, m)$ giving the number of items of type j packed into bin i, the CSP can be modeled by the ILP

$$\min \sum_{i=1}^{u} y_i \tag{3.6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i \quad (i = 1, \dots, u), \tag{3.7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \quad (j = 1, \dots, m), \tag{3.8}$$

$$y_i \in \{0, 1\} \quad (i = 1, \dots, u), \tag{3.9}$$

$$\xi_{ij} \ge 0, \text{integer} \quad (i = 1, \dots, u; j = 1, \dots, m). \tag{3.10}$$

The BPP is known to be $\mathcal{NP}$-hard in the strong sense (by transformation from the 3-Partition problem, see Garey and Johnson [130]). As any instance of either problem can easily be transformed into an equivalent instance of the other, the same holds for the CSP.

These two problems are among the most intensively studied problems in combinatorial optimization. Two recent surveys on exact methods (Delorme et al. [98]) and approximation algorithms (Coffman et al. [75]) consider in total over 230 different references. Previous surveys were presented by Garey and Johnson [131], Coffman et al. [77, 72], Sweeney and Paternoster [270], Dyckhoff [112], Martello and Toth [214] (Chapter 8), Dyckhoff and Finke [113], Valério de Carvalho [279], Wäscher et al. [289], among others. Most solution methodologies have been tried on these problems: different kinds of ILP models, lower bound computations, branch-and-bound, branch-and-price, constraint programming, approximation algorithms, heuristics, and metaheuristics.

A number of web-based libraries for optimization problems can be found on the Internet. The oldest one is probably the famous OR-Library, a collection of test data sets for a variety of Operations Research problems, implemented by Beasley [24]. Other relevant libraries are those implemented by Burkard et al. [48, 49] (*Quadratic Assignment Problem*), Applegate et al. [12] (the well-known *TSP web page* `http://www.math.uwaterloo.ca/tsp/`), Groër et al. [141] and Uchoa et. al [275] (*Vehicle Routing Problems*), Koch et al. [176] (*Mixed Integer Programming*), and Friberg [126] (*Conic Optimization*). An earlier, smaller version of the BPPLIB was implemented as an auxiliary instrument for the computational experiments presented in [98]. The current BPPLIB contains pointers to the literature, original computer codes, links to computer codes from the Internet, benchmark instances, and an open source visual application to interactively solve BPP instances.

In the next section we introduce the computer codes and the visual solver provided by the BPPLIB. In Section 3.3 we describe the available benchmarks: some of them were used in [98] for the computational evaluations of the different exact approaches, using commercial solver Cplex when needed. As the library has been enriched by also providing versions based on the freeware solver SCIP, in Section 3.4 we provide new experiments aiming at evaluating the computational difference between the two versions. In addition, we describe new test instances, that appeared after the publication of [98], and present the corresponding computational experiments.

3.2 Computer codes

The BPPLIB provides twelve computer codes of different types for the exact solution of the BPP and the CSP.

Branch-and-bound

The first effective exact algorithms for the BPP were based on a branch-and-bound approach. The library provides, in chronological order:

- **MTP**: Fortran code of the BPP algorithm by Martello and Toth [214], originally available in the diskette accompanying the book. The algorithm adopts a depth-first strategy to explore a branch-decision tree that considers one item per level: descendant nodes are generated by assigning the current item, in turn, to all already initialized bins and possibly to a new bin. While the approach is effective for BPP instances, considering one item at a time is clearly inefficient for CSP instances with high item multiplicity. The code can be run using the Fortran front end of the GNU Compiler Collection GCC;

- **BISON**: Scholl et al. [251] obtained a very efficient BPP algorithm by enriching MTP through new lower bounds and a Tabu search algorithm to help the search by means of effective heuristic solutions. The code was implemented in Pascal, and can be obtained from the authors, using the address provided in the library. Worth is mentioning that, in spite of its 'age', this program is still working and quite effective (see [98]): at the time of writing, it can be run using compiler `fpc (version 3.0.0 for x86_64)`;

- **CVRPSEP**: we provide a link to the C code implemented by J. Lysgaard as part of a separation routine within the algorithm by Lysgaard et al. [205] for the capacitated vehicle routing problem. The routine was obtained by using procedures from MTP. It is generally less efficient than MTP, but we decided to include it in the library mainly because one may prefer a C code to a Fortran code. The implementation details can be found in a technical report by Lysgaard [204].

Branch-and-price

This modern evolution of the branch-and-bound approach can produce very effective algorithms for the problems at hand. We provide links to two computer codes:

- **BELOV**: C++ implementation by G. Belov of the algorithm by Belov and Scheithauer [30], using Cplex for the inner routines. The algorithm is tailored to the exact solution of CSP instances, and it computationally proved to be the most powerful approach both in the case of low and high item multiplicity;

- **SCIP-BP**: freeware SCIP C code for a branch-and-price BPP algorithm based on the classical Ryan and Foster [243] branching rule and available at the SCIP web page. This code is only effective for instances with small number of item types and low item multiplicity.

Pseudo-polynomial formulations solved via ILP

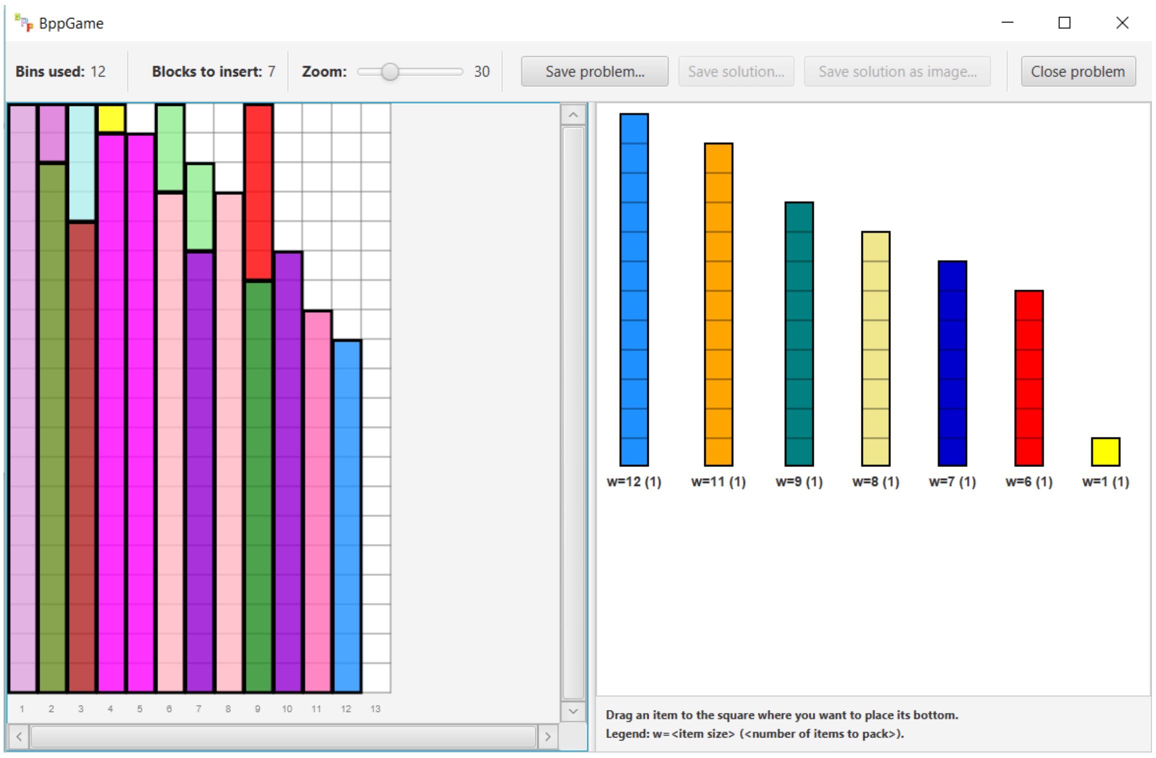
Already in the Seventies, pseudo-polynomial models coming from a graph representation of the solution space were proposed. For many years, solution approaches based on such models have been regarded as very theoretical, with no practical interest, due to the huge number of variables and constraints they imply. Up to few years ago, these methods were mainly used within branch-and-price algorithms (see, e.g., Valério de Carvalho [278]). However, nowadays computational power of ILP solvers made them competitive with branch-and-price algorithm also for the case of realistic size instances, provided the number of generated variables (that depends on capacity, number of items, and item weights) is not too big. The BPPLIB provides four algorithms based on pseudo-polynomial models:

- **ONECUT**: C++ implementation of the *one-cut* CSP model independently defined in the Seventies by Rao [234], and Dyckhoff [111];
- **ARCFLOW**: C++ implementation of the *arc-flow* CSP model by Valério de Carvalho [278];
- **DPFLOW**: C++ implementation of the *DP-flow* BPP model by Cambazard and O'Sullivan [52];
- **VPSOLVER**: link to the C++ implementation by Brandão and Pedroso [45] of their CSP algorithm. This is currently the most effective pseudo-polynomial approach, and its performance is often competitive with that of BELOV.

For the first three codes we provide both a version that uses Cplex as an inner routine, and a version that uses SCIP. Code VPSOLVER was instead implemented by the authors in a version that invokes Gurobi.

BppGame: An interactive visual solver

The library includes the pointer to an open source visual ScalaFX application to interactively solve BPP and CSP instances. The application is derived from a more general tool for the solution of two-dimensional packing problems, see Costa et al. [82]. It allows an easy interaction to obtain a feasible solution of a given problem instance. The application has a number of features, that are fully described in its own web page `http://gianlucacosta.info/BppGame/`. The easiest way to test it consists in following the hyperlink and executing the sequence of actions: Download zip and extract its contents → BppGame-x.x → bin → BppGame.bat (Windows) or BppGame (Linux) → Sample problems. Figure 3.1 shows the BppGame visualization of an instance. The user can click on an item on the right frame, and drag and drop it to a selected position in the left frame.



Drag an item to the square where you want to place its bottom.
Legend: w=<item size> (<number of items to pack>).

Figure 3.1: The interactive visual solver

3.3 Benchmarks

The BPPLIB provides in total 6 195 test instances belonging to four categories. Each instance is provided, using unified formats, both in BPP and CSP version.

Literature instances

This section contains the 1 615 instances proposed by

- Falkenauer [119]: 80 (easy) instances with uniformly distributed item sizes and 80 (more difficult) instances obtained through triplets of items that must be packed into the same bin in any optimal solution;
- Scholl et al. [251]: three sets of instances with uniformly distributed item sizes. The first set is composed by 720 easy instances, the second set by 480 instances of medium difficulty, and the third set by 10 difficult instances characterized by huge capacities;
- Wäscher and Gau [288]: 17 very hard instances selected by the authors from a much larger set of instances belonging to different typologies;
- Schwerin and Wäscher [254]: two sets of 100 relatively easy instances each;
- Schoenfield [250]: 28 hard instances that do not involve huge capacities.

In the library, each set is identified by the name of the (first) author.

Randomly generated instances

The library provides the 3 840 instances that were randomly generated for the computational experiments reported in [98]. The instances have different values of n (50, 100, 200, 300, 400, 500, 750, 1 000), of c (50, 75, 100, 120, 125, 150, 200, 300, 400, 500, 750, 1 000), and of the minimum ($0.1\,c$, $0.2\,c$) and maximum ($0.7\,c$, $0.8\,c$) item weight. The benchmark contains 10 instances for each of the 384 quadruplets (n, c, minimum weight, maximum weight). These instances are relatively easy, and the algorithms listed in Section 3.2 could solve most of them within reasonable CPU times.

Hard instances

In order to perform experiments on challenging instances, a number of so called *augmented Non-IRUP* and *augmented IRUP* instances were proposed in [98], using as a basis a set of Non-IRUP instances presented in Caprara et al. [54]. For the 250 instances of the former class an optimal solution is easy to find, but its optimality is very difficult to prove. Even the continuous relaxation of the set covering formulation (the basis of branch-and-price algorithms) and that of the pseudo-polynomial formulations fail in reaching the optimal value. As a consequence, algorithms based on such relaxations require either a huge branching process or a heavy cut generation: already for $n \approx 400$, no algorithm is capable of solving all of them to proven optimality. For the 250 instances of the latter class, it is easy to produce a lower bound whose value is equal to the optimum, but it is difficult to build an optimal solution.

GI instances

The library includes 240 new instances, proposed by Gschwind and Irnich [142] after the publication of [98]. Such instances, uniformly randomly generated, are characterized by very large capacities. They are organized into four sets of 60 instances each. As shown in the next section, two of such sets are generally difficult to solve.

3.4 Computational experiments

We report the results of some experiments executed on an Intel Xeon 3.10 GHz (equipped with four cores) with 8 GB RAM, all executed with a single core. In order to test the codes on non-trivial instances, we preliminarily obtained an upper bound through the classical *best fit decreasing heuristic* and computed the lower bound value known as L2 (see [24]): only instances for which these two values were different were then tested.

Tables 3.1 and 3.2 give the number of literature instances that were solved in one CPU minute (and, in parentheses, the average CPU time), by, respectively, the enumeration algorithms and the pseudo-polynomial models. (For the non-solved instances, one CPU minute was considered.) For each instance set, boldface highlights the cases where all instances were solved to proven optimality.

The results in Table 3.1 summarize the (much more detailed) tables presented in [98]: they are provided here in order to give the reader information on the performance of the codes provided in the BPPLIB. The results in Table 3.2 include new results obtained using SCIP as the ILP solver. The tables confirm the clear superiority of BELOV and VPSOLVER over the other algorithms.

Table 3.1: Literature instances, enumerative algorithms. Number of instances solved in less than one minute (average CPU time in seconds).

Set	Number of tested instances	Branch-and-bound			Branch-and-price	
		MTP	BISON	CVRPSEP	BELOV	SCIP-BP
Falkenauer U	74	22 (42.8)	44 (24.5)	22 (42.2)	**74** (0.0)	18 (50.1)
Falkenauer T	80	6 (55.5)	42 (30.6)	0 (60.0)	57 (24.7)	35 (39.4)
Scholl1	323	242 (15.1)	288 (7.0)	223 (19.4)	**323** (0.0)	244 (22.4)
Scholl2	244	130 (28.2)	233 (3.0)	65 (44.2)	**244** (0.3)	67 (49.2)
Scholl3	10	0 (60.0)	3 (42.0)	0 (60.0)	**10** (14.1)	0 (60.0)
Wäscher	17	0 (60.0)	10 (24.7)	0 (60.0)	**17** (0.1)	0 (60.0)
Schwerin1	100	15 (51.1)	100 (0.0)	9 (55.4)	**100** (1.0)	0 (60.0)
Schwerin2	100	4 (57.6)	63 (22.2)	0 (60.0)	**100** (1.4)	0 (60.0)
Hard28	28	0 (60.0)	0 (60.0)	0 (60.0)	**28** (7.3)	7 (51.2)
Total (average)	976	419 (34.4)	783 (12.3)	319 (40.8)	953 (2.7)	371 (42.2)

Table 3.2: Literature instances, pseudo polynomial models. Number of instances solved in less than one minute (average CPU time in seconds).

Set	Number of tested instances	ONECUT		ARCFLOW		DPFLOW		VPSOLVER
		Cplex	SCIP	Cplex	SCIP	Cplex	SCIP	
Falkenauer U	74	**74** (0.2)	67 (23.8)	**74** (0.2)	70 (18.7)	37 (38.8)	0 (60.0)	**74** (0.1)
Falkenauer T	80	**80** (8.7)	21 (44.9)	**80** (3.5)	33 (41.4)	40 (41.7)	20 (50.8)	**80** (0.4)
Scholl1	323	**323** (0.1)	318 (5.0)	**323** (0.1)	320 (5.1)	289 (13.0)	178 (34.0)	**323** (0.1)
Scholl2	244	118 (38.7)	20 (56.3)	202 (18.9)	39 (53.7)	58 (50.4)	11 (58.5)	208 (14.0)
Scholl3	10	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	**10** (6.3)
Wäscher	17	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	6 (49.4)
Schwerin1	100	**100** (13.1)	0 (60.0)	**100** (1.5)	0 (60.0)	0 (60.0)	0 (60.0)	**100** (0.3)
Schwerin2	100	**100** (11.7)	0 (60.0)	**100** (1.5)	1 (59.5)	0 (60.0)	0 (60.0)	**100** (0.3)
Hard28	28	6 (54.6)	0 (60.0)	16 (40.6)	0 (60.0)	0 (60.0)	0 (60.0)	27 (14.2)
Total (average)	976	801 (16.3)	426 (36.9)	895 (8.2)	463 (35.6)	424 (38.9)	209 (50.3)	928 (5.0)

Table 3.2 shows in addition that the performance of ONECUT and ARCFLOW is not affected by the ILP solver for most of the easy instances, while their performance for the difficult instances sharply worsens when SCIP is used instead of Cplex. This behavior could be explained by the difference in the number of generated variables and constraints between different benchmarks. For example, ARCFLOW produces, on average, 1 735 variables and 103 constraints for Scholl 1 instances, while for "Scholl 2" it produces, on average, 39 307 variables and 840 constraints.

Tables 3.3 and 3.4 refer to the randomly generated instances used in [98], and provide the same information as in Tables 3.1 and 3.2. The previous observations are confirmed:

BELOV and VPSOLVER outperform the other approaches, and the use of SCIP decreases the algorithms' performance, especially for large values of n.

Table 3.3: Random instances, enumerative algorithms. Number of instances solved in less than one minute (average CPU time in seconds).

n	Number of tested instances	Branch-and-bound			Branch-and-price	
		MTP	BISON	CVRPSEP	BELOV	SCIP-BP
50	165	163 (0.8)	**165** (0.0)	164 (0.4)	**165** (0.0)	**165** (0.9)
100	271	243 (7.4)	257 (3.8)	239 (8.4)	**271** (0.0)	**271** (4.6)
200	359	237 (21.6)	290 (12.0)	220 (25.0)	**359** (0.0)	293 (22.6)
300	393	166 (35.7)	265 (20.7)	144 (38.7)	**393** (0.1)	155 (44.1)
400	425	151 (39.1)	244 (26.1)	138 (41.2)	**425** (0.2)	114 (49.8)
500	414	121 (43.0)	208 (30.3)	128 (42.6)	**414** (0.2)	69 (55.1)
750	433	93 (47.3)	214 (30.9)	98 (47.3)	**433** (0.4)	22 (59.5)
1000	441	78 (49.5)	196 (33.9)	73 (50.8)	**441** (0.7)	0 (60.0)
Total (average)	2901	1252 (34.7)	1839 (22.6)	1204 (36.0)	**2901** (0.2)	1089 (42.4)

Table 3.4: Random instances, pseudo polynomial models. Number of instances solved in less than one minute (average CPU time in seconds).

n	Number of tested instances	ONECUT		ARCFLOW		DPFLOW		VPSOLVER
		Cplex	SCIP	Cplex	SCIP	Cplex	SCIP	
50	165	**165** (0.1)	163 (2.0)	**165** (0.1)	**165** (1.6)	**165** (0.5)	162 (5.1)	**165** (0.0)
100	271	**271** (0.8)	249 (8.6)	**271** (0.3)	262 (10.1)	**271** (5.0)	168 (34.5)	**271** (0.1)
200	359	358 (2.4)	286 (15.4)	**359** (0.8)	278 (20.2)	292 (21.0)	76 (51.8)	**359** (0.3)
300	393	385 (4.5)	272 (22.2)	391 (2.0)	262 (24.9)	243 (33.9)	31 (57.3)	**393** (0.6)
400	425	408 (5.1)	293 (22.0)	421 (3.0)	276 (25.8)	193 (42.4)	23 (58.1)	**425** (0.8)
500	414	394 (6.3)	275 (24.0)	402 (4.0)	258 (26.5)	169 (44.8)	13 (58.8)	413 (1.7)
750	433	401 (7.8)	284 (24.3)	415 (6.0)	279 (25.7)	120 (52.6)	12 (59.1)	431 (2.4)
1000	441	407 (8.1)	280 (25.8)	416 (6.8)	281 (26.1)	67 (56.4)	7 (59.6)	434 (3.4)
Total (average)	2901	2789 (5.0)	2102 (20.0)	2840 (3.3)	2061 (22.3)	1520 (36.7)	492 (52.5)	2891 (1.4)

3.4.1 GI instances

We report in Table 3.5 the results of computational experiments for the GI benchmark, a set of CSP instances recently proposed by Gschwind and Irnich [142] for testing their dual inequalities aimed at stabilizing column generation processes. They are organized into four groups (AA, AB, BA, and BB), characterized by different item weight ranges and capacities. Each group has three sets of 20 instances each, characterized by the number of item types (125, 250, and 500). We tested the best enumerative algorithm (BELOV) and the best pseudo-polynomial approaches (ARCFLOW and VPSOLVER) with a time

Table 3.5: Number of GI instances solved in less than one hour (average time in seconds).

Set	m	Number of tested instances	BELOV	ARCFLOW	VPSOLVER
AA	125	20	**20** (0.1)	19 (1 092.6)	**20** (0.9)
	250	20	**20** (0.9)	0 (3 600.0)	**20** (14.5)
	500	20	**20** (7.5)	0 (3 600.0)	16 (1 345.9)
AB	125	20	**20** (0.7)	0 (3 600.0)	0 (3 600.0)
	250	20	**20** (2.1)	0 (3 600.0)	0 (3 600.0)
	500	20	**20** (29.9)	0 (3 600.0)	0 (3 600.0)
BA	125	20	**20** (0.1)	**20** (1 120.9)	**20** (1.4)
	250	20	**20** (1.3)	0 (3 600.0)	**20** (23.1)
	500	20	**20** (7.2)	0 (3 600.0)	17 (1 450.2)
BB	125	20	**20** (0.2)	0 (3 600.0)	0 (3 600.0)
	250	20	**20** (2.3)	0 (3 600.0)	0 (3 600.0)
	500	20	**20** (29.1)	0 (3 600.0)	0 (3 600.0)
Total (average)		240	**240** (6.8)	39 (3 234.8)	113 (2 036.3)

limit of one hour. BELOV could solve all of these instances very quickly, while they turned out to be extremely difficult for the pseudo-polynomial models. The behavior of the latter approaches was particularly poor for the instances that have items with very small weight and huge capacities (AB and BB, with $c \geq 500\,000$), which induce a high number of variables and constraints. For example, the ILP model produced by ARCFLOW has on average 549 441 variables and 131 219 constraints for instances AA with $m = 125$, but 5 754 617 variables and 404 283 constraints for instances AB with $m = 125$.

Chapter 4

Enhanced Pseudo-Polynomial Formulations for Bin Packing and Cutting Stock Problems

[1]

In this chapter, we study pseudo-polynomial formulations for the one dimensional bin packing and cutting stock problems. We first give a complete overview of the dominance and equivalence relations that exist among the main pattern-based and pseudo-polynomial MILP formulations that have been proposed in the literature: Gilmore and Gomory, the proper relaxation, the one-cut, the arc-flow and the DP-flow formulations. Then, we introduce reflect, a new MILP formulation that uses just half of the bin and needs significantly less constraints and variables than the classical arc-flow. We propose heuristics and lower bounding techniques that can be used to compensate reflect weaknesses when the capacity of the instance is too hight and we show how reflect can be modified to solve the variable size bin packing problem and the bin packing problem with item fragmentation. We test reflect on benchmark instances of the three problems and achieve state of the art results, improving upon previous algorithms in the literature and finding several new proven optimal solutions.

Keywords: Bin packing, Cutting stock, Pseudo-Polynomial, Equivalent Models, Variable Size, Fragmentation.

[1]The results of this chapter appears in: M. Delorme and M. Iori, Enhanced Pseudo-Polynomial Formulations for Bin Packing and Cutting Stock Problems, *Technical Report*, 2017 [97].

4.1 Introduction

The *bin packing problem* (BPP) requires to pack a set of weighted items into the minimum number of identical capacitated bins. The *cutting stock problem* (CSP) is the BPP version in which all items having the same weight are grouped together into item types. The term packing is normally adopted for applications devoted to the minimization of the number of boxes (containers, cells, ...) required to allocate a set of items, and the term cutting for industrial process where stocks of a material (steel bars, pipes, ...) have to be cut into demanded items while minimizing waste. Apart from these applications, the BPP and the CSP also serve in the optimization of a variety of real-world problems, including capacitated vehicle routing, resource-constrained scheduling problems, and production systems, just to name a few.

Their wide range of applications motivated a large research interest, and, indeed, the two problems have been tackled by almost all known combinatorial optimization techniques. We refer the interested readers to the recent surveys by Coffman et al. [75], who focused on approximation algorithms, by Alves et al. [9], who studied the use of dual-feasible functions, and by Delorme et al. [98], who reviewed and tested exact algorithms and mathematical models.

The BPP and the CSP have also been the training ground of some *mixed integer linear programming* (MILP) models, and corresponding solution algorithms, that later became useful tools for many other combinatorial problems. Gilmore and Gomory [134] modeled the CSP as a set-covering by using a pattern-based representation, and then proposed the well-known column generation algorithm. The strength of this model comes from the very high-quality of its continuous relaxation value. This relaxation (and other variants, which are discussed in detail in Section 4.3) are at the basis of the most effective *branch-and-price* algorithms for the solution of the BPP and the CSP (see, e.g., Vanderbeck [283] and Belov and Scheithauer [30]).

The number of variables in the pattern-based models is exponential in the number of items. A different branch of the literature focused, instead, on modeling the BPP and the CSP with *pseudo-polynomial* MILP models, that is, formulations in which the numbers of variables and constraints are polynomials in both the number of items and in the bin capacity. Already in the 1960s, Shapiro [256] showed the connection that exists between integer programming and *dynamic programming* (DP), by studying the knapsack problem as a shortest path problem on a network of pseudo-polynomial size, where (i) nodes are associated with different levels of occupation of the knapsack, (ii) items are associated with arcs having length equal to the item weight, and (iii) decision variables are associated with arcs. Wolsey [293] extended this idea to the case of problems involving multiple knapsack inequalities, showing how to solve the CSP as a network flow problem with additional constraints. The research was later continued by Valério de Carvalho [278], who solved the CSP with a model called *arc-flow*, and showed that the continuous relaxation value of this model is equal to that of Gilmore and Gomory [134]. A related MILP model, known as *one-cut*, was independently developed by Rao [234] and Dyckhoff [112]. One-cut does not associate variables with arcs on a network, but solves instead the CSP by using variables that represent the physical positions of the cuts. Items are obtained by performing the selected cuts one at a time, either along the bin or along residual bin portions obtained by previous cuts.

The mentioned research on pseudo-polynomial formulations mainly had a theoretical interest, due to the fact that the large size of these models made them unsolvable in practice even for moderate-size instances. In recent years, however, the sharp development of MILP commercial software and the increase in computational power made these formulations a viable tool to solve the BPP, the CSP, and other related combinatorial optimization problems. Among others, Brandão and Pedroso [45] obtained good results on a number of cutting and packing problems, by embedding into arc-flow a set of improvements. Furini et al. [127] generalized one-cut model to solve a set of two-dimensional guillotine cutting problems. Delorme et al. [100] used a modified arc-flow as stepping stone of a decomposition algorithm for two-dimensional packing problems with items rotation.

In this chapter, we continue this established line of research by providing new results that are interesting both from a theoretical and from a computational point of view. We first focus on the relation that exists among well-known pattern-based and pseudo-polynomial formulations, providing a complete picture of equivalences and dominances among them. Then, we select one of these formulations, namely, the arc-flow, and show how its size can be conveniently reduced leading to an equivalent model formulation called *reflect*, in which just half of the bin is needed to model a CSP instance. We then generalize reflect to deal with other important BPP and CSP variants, showing that on each variant our exact algorithm achieves state of the art computational results.

In detail, we provide the following contributions:

- We prove that one-cut and arc-flow formulations for the CSP are equivalent (i.e., they have the same continuous relaxation value), closing a long-standing research question.

- We extend the previous result and provide a clear picture of the dominance and equivalence relations that exist among the main pattern-based and pseudo-polynomial MILP formulations that have been proposed for the BPP and the CSP.
- We introduce a new formulation, called *reflect*, that improves the classical arc-flow by using just half of the bin and thus needing significantly less constraints and variables.
- We test reflect on BPP and CSP benchmark instances, achieving state of the art results and finding several new proven optimal solutions.
- We also efficiently solve instances with very large bin capacities, by devising heuristics and lower bounding techniques that are based on the combined use of column generation and reflect. In particular, we demonstrate that our heuristics are faster and more efficient than traditional approaches based on solving set-covering MILP models with restricted sets of columns.
- We show the easy replicability of our results, by adapting reflect to solve the *variable-sized BPP* (VSBPP) and the *BPP with item fragmentation* (BPPIF). For both problems, we obtain results that consistently improve those available in the literature.

The remainder of the chapter is organized as follows. In Section 4.2, we give the necessary notation and review the main formulations proposed for the BPP and the CSP. In Section 4.3, we establish the full set of relations among the existing formulations. In Section 4.4, we present reflect and other procedures that we developed to improve its performance. In Section 4.5, we extend our results and algorithms to deal with the VSBPP and the BPPIF. In Section 4.6, we present the outcome of extensive computational experiments and then, in Section 4.7, we draw some conclusions. For the sake of conciseness, all proofs are reported in the *Supplementary Material* (SM).

4.2 The BPP, the CSP, and their well-known formulations

In this section, we formally describe the problems, give the necessary notation, and present the main formulations developed for the BPP and the CSP.

4.2.1 Problem description and notation

In the BPP, we are given a set of n items, each having weight w_j $(j = 1, \ldots, n)$, and an unlimited supply of identical bins of capacity c. The aim is to pack all items into the minimum number of bins, so that the sum of the item weights in any bin does not exceed the capacity. To better adapt to either the concept of packing or that of cutting, in the following we use alternatively the terms *weight* and *width* when referring to w_j. We suppose that all input values are integer and $0 < w_j < c$ holds for any j. The CSP has the same aim of the BPP, but, apart from the unlimited supply of bins (stocks) of capacity c, its input consists of a set of m item types. Each item type j has a demand of d_j items, all having width w_j $(j = 1, \ldots, m)$. A CSP instance can be obtained from a BPP one by grouping into item types all items having the same width. So, a solution method for the CSP also solves the BPP, and viceversa. Unless stated otherwise, the formulations that we report below refer to the CSP.

We use F_{XX} to denote a given MILP formulation. We use $L(F_{XX})$ to denote the continuous (linear programming) relaxation of F_{XX}, which is obtained by dropping the integrality constraints from F_{XX}. When no confusion arises, we also use $L(F_{XX})$ to define the optimal solution value of $L(F_{XX})$. We say that a formulation F_{XX} is *equivalent* to a formulation F_{YY}, if $L(F_{XX}) = L(F_{YY})$ holds for any problem instance. We say that a formulation F_{XX} *dominates* a formulation F_{YY}, if $L(F_{XX}) \geq L(F_{YY})$ holds for any instance and $L(F_{XX}) > L(F_{YY})$ holds for at least one instance. If F_{XX} dominates F_{YY}, then the convex hull of F_{XX} is included into that of F_{YY}; in such a case we also say that F_{XX} *is included* into F_{YY}. Suppose a given formulation contains a variable x; we use the notation $\bar{x}$ to denote the value taken by x in a given solution of the formulation.

4.2.2 Pattern-based formulations

We define a pattern p as an array $(a_{1p}, \ldots, a_{mp})$, with a_{jp} being a non-negative integer that gives the number of items of type j that are included in the pattern. Let P define the class of feasible patterns, i.e., those patterns p for which $\sum_{j=1}^{m} a_{jp} w_j \leq c$ holds. Gilmore and Gomory [134] associated with each pattern an integer decision variable ξ_p, indicating the number of times pattern p is selected, and modeled the CSP as the following set-covering formulation

$$(F_{GG}) \quad \left\{ \min\ z = \sum_{p \in P} \xi_p : \sum_{p \in P} a_{jp}\xi_p \geq d_j \text{ for } j = 1, \ldots, m,\ \xi_p \in \mathbb{N} \text{ for } p \in P \right\}. \quad (4.1)$$

Note that in (4.1) nothing prevents a_{jp} from being larger than d_j, thus there could be optimal solutions of F_{GG}, or of its continuous relaxation $L(F_{GG})$, with patterns containing a number of items greater than their demands. Consider the following example.

EXAMPLE 2 *A CSP instance with* $m = 3$, $w = (7, 4, 3)$, $d = (1, 1, 1)$, *and* $c = 11$.

An optimal solution of $L(F_{GG})$ has value 4/3 and consists of selecting three patterns: pattern $(1, 1, 0)$ (i.e., a pattern containing a copy of the item of width 7 and a copy of the item of width 4) with value $\bar{\xi}_1 = 2/3$, pattern $(1, 0, 1)$ with value $\bar{\xi}_2 = 1/3$, and pattern $(0, 1, 2)$ with value $\bar{\xi}_3 = 1/3$.

The *proper relaxation* (see, e.g., Nitsche et al. [225]) tries to overcome this drawback by focusing on a restricted set P' of feasible patterns. Each pattern $p \in P'$ satisfies $0 \leq a_{jp} \leq d_j$, for $j = 1, \ldots, m$, and $\sum_{j=1}^{m} a_{jp}w_j \leq c$. The CSP can then be modeled as

$$(F_{PR}) \quad \left\{ \min\ z = \sum_{p \in P'} \xi_p : \sum_{p \in P'} a_{jp}\xi_p \geq d_j \text{ for } j = 1, \ldots, m,\ \xi_p \in \mathbb{N} \text{ for } p \in P' \right\}. \quad (4.2)$$

Because $P' \subseteq P$, we can state that $L(F_{PR}) \geq L(F_{GG})$. Consider now Example 2. An optimal solution of $L(F_{PR})$ has value 3/2 and consists of selecting the three following patterns: $(1, 1, 0)$ with value $\bar{\xi}_1 = 1/2$, $(1, 0, 1)$ with value $\bar{\xi}_2 = 1/2$, and $(0, 1, 1)$ with value $\bar{\xi}_3 = 1/2$. We can thus conclude with the (known) fact that F_{PR} dominates F_{GG}.

Note that, after rounding up to the nearest integer, for most of the CSP instances the lower bound values obtained by solving $L(F_{PR})$ and $L(F_{GG})$ coincide. Still, there are many instances for which this fact does not hold. Such instances have been investigated intensively in studies over the *Mixed-Integer Round-Up* (MIRUP) conjecture, which states that both $(z_{opt} - \lceil L(F_{GG}) \rceil) \leq 1$ and $(z_{opt} - \lceil L(F_{PR}) \rceil) \leq 1$ hold for any instance of the CSP, with z_{opt} being the optimal integer value. For further details on this branch of the literature, we refer to, e.g., Caprara et al. [54] and Kartak et al. [171]. Note also that practically all branch-and-price algorithms devoted to the solution of the BPP are based on patterns with binary a_{jp} values, and hence they use the proper relaxation.

4.2.3 Pseudo-polynomial formulations

Pseudo-polynomial formulations involve a large number of variables and constraints, and hence they have been usually proposed in the literature together with appropriate reduction techniques. In this section, we describe basic models that do not make use of any reduction technique. Existing and new reduction techniques are discussed in the next Sections 4.3 and 4.4.

The *one-cut formulation* (F_{OC}) was formally introduced by Rao [234] and Dyckhoff [111]. It simulates the physical cutting process by choosing a series of cuts each dividing the bin (or a portion of the bin) into two smaller pieces, a left one and a right one. While the left piece is an item, the right piece is either an item or a residual that can be re-used to produce smaller items with successive cuts. To describe the model, we need some additional notation. Let $\mathcal{W} = \{w_1, w_2, ..., w_m\}$ define the set of item widths. Let $\mathcal{S}$ be the set of all combinations of item widths whose total width does not exceed c. This can computed trough a standard DP, obtaining

$$\mathcal{S} = \left\{\bar{w} = \sum\nolimits_{j=1}^{m} w_j x_j,\ \bar{w} \le c,\ x_j \in \mathbb{N} \text{ for } j = 1, \dots, m\right\}. \tag{4.3}$$

Let $\mathcal{R}$ define the set of residual widths, computed as $\mathcal{R} = \{c - \bar{w} : \bar{w} \in \mathcal{S} \text{ and } \bar{w} \le c - \min_j\{w_j\}\}$. The length of any left piece is in $\mathcal{W}$, while the length of any right piece (including the full bin capacity) is in $\mathcal{R}$. For a given width $q \in \mathcal{W} \cup \mathcal{R}$, let $L_q = d_j$ if there exists an item j having width $w_j = q$, and 0 otherwise. In other words, L_q gives the demand of a certain width q. The model makes use of three additional sets of widths. Set $A(q)$ contains the piece widths that can be used for producing a left piece having width q, if any; formally, $A(q) = \{p \in \mathcal{R} : p > q\}$ for any $q \in \mathcal{W}$, and $A(q) = \emptyset$ for any $q \notin \mathcal{W}$. Set $B(q)$ contains all item widths that, whether cut as a left piece, would leave a right piece having width q; formally, $B(q) = \{p \in \mathcal{W} : p + q \in \mathcal{R}\}$. Set $C(q)$ contains the set of item widths that can be cut as a left piece by using a residual of width q; formally, $C(q) = \{p \in \mathcal{W} : p < q\}$.

We consider an integer decision variable y_{pq}, which provides the number of times a piece of width p is cut into a left piece (item) of width q and a right piece (item or residual) of width $p - q$. The one-cut model is then defined as

$$(F_{OC}) \quad \min \quad z = \sum_{q \in \mathcal{W}} y_{cq} \tag{4.4}$$

$$\text{s.t.} \quad \sum_{p \in A(q)} y_{pq} + \sum_{p \in B(q)} y_{p+q,p} \geq L_q + \sum_{r \in C(q)} y_{qr} \quad q \in \mathcal{W} \cup \mathcal{R} \setminus \{c\} \tag{4.5}$$

$$y_{pq} \in \mathbb{N} \quad p \in \mathcal{R}, q \in \mathcal{W}, p > q. \tag{4.6}$$

While function (4.4) minimizes the number of bins used, constraints (4.5) ensures that the sum of the left and right pieces having width q is not smaller than the demand of width q plus the number of times a residual of width q is re-cut to produce other items.

The *arc-flow formulation* (F_{AF}), formally proposed for the CSP by Valério de Carvalho [278], is a position-indexed formulation that builds upon the graph used in Shapiro [256] and Wolsey [293]. Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a graph having vertex set $\mathcal{V} = \{0, 1, \dots, c-1, c\}$ and arc set $\mathcal{A} = \mathcal{A}_\text{I} \cup \mathcal{A}_\ell$, where $\mathcal{A}_\text{I} = \{(d, e) : 0 \leq d < e \leq c, d \in \mathcal{S}, \text{ and } \exists\, j \in \{1, \dots, m\} : e - d = w_j\}$ is the set of *item arcs*, and $\mathcal{A}_\ell = \{(d, d+1) : d = 0, 1, \dots, c-1\}$ is the set of *loss arcs*. Items arcs model the position of the items in the bin, which are limited by (4.3), whereas loss arcs model empty bin portions. The filling of a bin corresponds then to a path from root node 0 to sink node c. Let $\delta^-(e)$ (respectively, $\delta^+(e)$) give the subset of arcs entering (respectively, emanating from) vertex e. By introducing an integer variable x_{de} giving the number of times arc (d, e) is chosen, the CSP can be modeled as

$$(F_{AF}) \quad \min \quad z \tag{4.7}$$

$$\text{s.t.} \quad \sum_{(e,f) \in \delta^+(e)} x_{ef} - \sum_{(d,e) \in \delta^-(e)} x_{de} = \begin{cases} z & \text{if } e = 0 \\ -z & \text{if } e = c \\ 0 & \text{for } e = 1, \dots, c-1 \end{cases} \tag{4.8}$$

$$\sum_{(d, d+w_i) \in \mathcal{A}} x_{d,d+w_i} \geq d_i \quad i = 1, \dots, m \tag{4.9}$$

$$x_{de} \in \mathbb{N} \quad (d, e) \in \mathcal{A}. \tag{4.10}$$

Constraints (4.8) impose flow conservation and constraints (4.9) ensure that the demand is fulfilled.

The *dynamic programming-flow formulation* (F_{DP}) has been formally introduced for the BPP only by Cambazard and O'Sullivan [52], but, similarly to the arc-flow, has origins in the early works by Shapiro [256] and Wolsey [293]. It can be seen as a disaggregated form of F_{AF}, which uses an expanded graph directly obtained from a DP table to model the bin fillings. In detail, let $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$ be the DP graph. Set $\mathcal{V}' = \{(j, d) : j = 0, \dots, n; d = 0, \dots, c\} \cup \{(n+1, c)\}$ has a vertex for each DP state plus a dummy node

$(n+1,c)$, whereas set $\mathcal{A}' = \{((j,d),(j+1,e)) : (j,d) \in \mathcal{V}'; (j+1,e) \in \mathcal{V}'\}$ contains arcs connecting two consecutive DP states. There are two types of arcs $((j,d),(j+1,e))$: those for which $e = d + w_j$, that model the selection of item j when the bin is partially filled by d units; and those having $e = d$, that state that the selection of j in d has been discarded. All vertices (j,d) having $d = c$ are connected to the last dummy vertex $(n+1,c)$. A path from $(0,0)$ to $(n+1,c)$ represents a feasible bin filling. With each arc $((j,d),(j+1,e)) \in \mathcal{A}'$ we associate an integer decision variable $\varphi_{j,d,j+1,e}$. By setting $\mathcal{V}'_0 = \mathcal{V}' \setminus \{(0,0),(n+1,c)\}$, we can model the BPP as

$$(F_{DP}) \quad \min \quad z \tag{4.11}$$

$$\text{s.t.} \sum_{((j,d),(j+1,e)) \in \delta^+((j,d))} \varphi_{j,d,j+1,e} - \sum_{((j-1,e),(j,d)) \in \delta^-((j,d))} \varphi_{j-1,e,j,d} = \begin{cases} z & \text{if } (j,d) = (0,0) \\ -z & \text{if } (j,d) = (n+1,c) \\ 0 & \text{if } (j,d) \in \mathcal{V}'_0 \end{cases} \tag{4.12}$$

$$\sum_{((j-1,d),(j,d+w_j)) \in \mathcal{A}} \varphi_{j-1,d,j,d+w_j} = 1 \qquad j = 1,\dots,n \tag{4.13}$$

$$\varphi_{j,d,j+1,e} \in \mathbb{N} \qquad ((j,d),(j+1,e)) \in \mathcal{A}'. \tag{4.14}$$

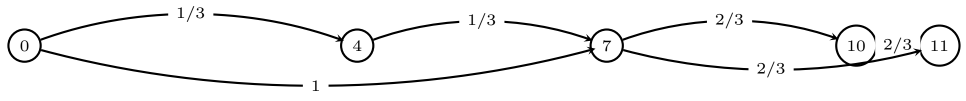
Constraints (4.12) force flow conservation and constraints (4.13) impose that each item (adopting the BPP notation) is selected once. A CSP instance can be modeled by F_{DP} by splitting each item type j into d_j items, obtaining a BPP instances with $n = \sum_{i=1}^{m} d_j$ items.

4.3 Relations among models

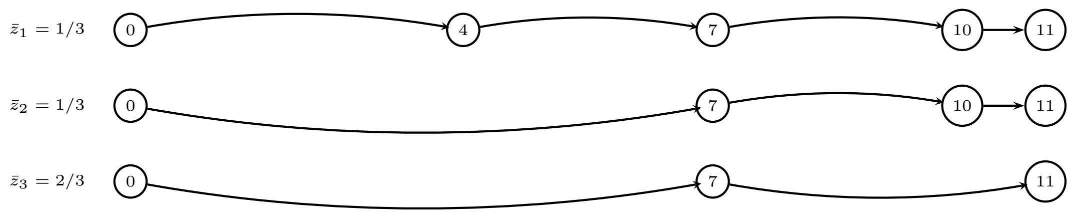
In this section, we prove the relations that exist among the introduced patterns-based and pseudo-polynomial formulations. To the best of our knowledge, this is the first time that a complete characterization of this area of research is provided in the literature. We first need two preliminary results.

LEMMA 1 *(Valério de Carvalho [278]) Any solution of arc-flow or of its continuous relaxation can be decomposed into a set of paths.*

The mentioned decomposition is based on the decomposition of non-negative flows into paths and cycles (see Ahuja et al. [3], Chapter 3), with the only remark that, being the arc-flow graph acyclic, only paths may occur. For the sake of clarity, the procedure that we use for this decomposition is given in Section 4.A of the SM. Consider Example 2. An optimal solution of $L(F_{AF})$ has value 4/3 and consists of $\bar{x}_{0,7} = 1$, $\bar{x}_{0,4} = 1/3$, $\bar{x}_{4,7} = 1/3$, $\bar{x}_{7,10} = 2/3$, $\bar{x}_{7,11} = 2/3$, and $\bar{x}_{10,11} = 2/3$, as shown in Figure 4.1-(a). By applying Algorithm 2 of Section 4.A, we obtain a decomposed flow that is made by three paths and is depicted in Figure 4.1-(b).



(a) Optimal $L(F_{AF})$ solution of value 4/3



(b) Decomposition of $L(F_{AF})$ into paths ($\bar{z}_p$ = flow on path p).

Figure 4.1: $L(F_{AF})$ solution and path decomposition for Example 2

LEMMA 2 *Any solution of one-cut or of its continuous relaxation can be decomposed into a set of binary trees.*

Proof. Proof Given in Section 4.B of the SM.

Consider again Example 2. An optimal solution of $L(F_{OC})$ has value 4/3 and consists of $\bar{y}_{11,7} = 1$, $\bar{y}_{11,4} = 1/3$, $\bar{y}_{7,3} = 1/3$, and $\bar{y}_{4,3} = 2/3$. By applying the modified algorithm 3 of Section 4.B to this solution, we obtain the three trees depicted in Figure 4.2. The leaves of each tree correspond to either items produced (as leaves 4, 3, and 3 in the left-most tree) or residuals (as leaf 1 in the left-most tree).

Intuitively, we can state a relation between the paths of the decomposed arc-flow solution and the trees of the decomposed one-cut solution. Figures 4.1 and 4.2 show a well-constructed example of this relation, which can be noticed by considering one at a time the paths from top to bottom and the trees from left to right. In reality, a few cases should be considered, especially in consideration of the fact that solutions of arc-flow and one-cut are not guaranteed to be left-aligned (as happens in the figures). We leave the technicalities to the SM, and provide our first result in terms of models relations.

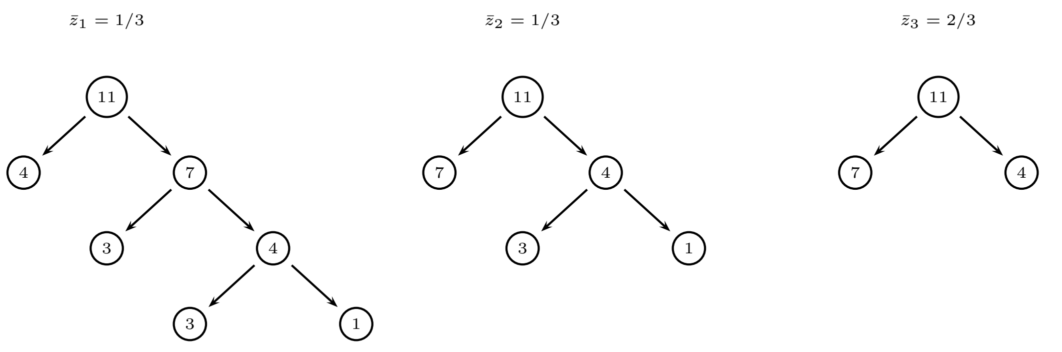


Figure 4.2: An $L(F_{OC})$ solution of Example 2 represented as a set of trees ($\bar{z}_t$ = value of tree t).

THEOREM 1 *F_{AF} is equivalent to F_{OC}.*

Proof. Proof Given in Section 4.C of the SM. □

Note that the proof of Theorem 1 contains two algorithmic transformations, from F_{AF} to F_{OC} and viceversa, which we believe can be useful when implementing solutions algorithms for the CSP (and for some of its generalizations). Note also that, by using the equivalence between F_{AF} and F_{GG} proved in Valério de Carvalho [278], we can observe that F_{OC} too is equivalent to F_{GG}.

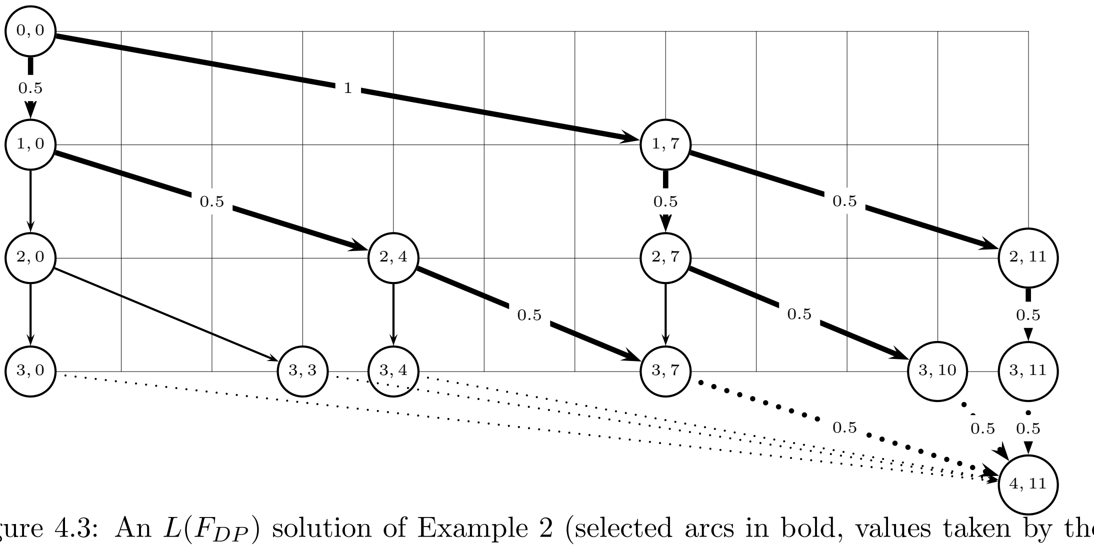


Figure 4.3: An $L(F_{DP})$ solution of Example 2 (selected arcs in bold, values taken by the selected variables on the arcs)

Now, let us concentrate on F_{DP}. An optimal $L(F_{DP})$ solution of Example 2 has value 3/2 and is shown in Figure 4.3. The arcs in bold lines are associated with the selected variables (whose values are reported on the corresponding arcs). This example is useful for the second relation that we prove.

THEOREM 2 *F_{DP} dominates F_{AF} (and hence F_{OC}).*

Proof. Proof Given in Section 4.D of the SM.

The relations among the pseudo-polynomial formulations are graphically depicted in the right part of Figure 4.4. The discussed equivalence of F_{AF} and F_{OC} with F_{GG} is depicted by the use of a dashed line. The other dashed line depicts our next result.

THEOREM 3 *F_{DP} is equivalent to F_{PR}.*

Proof. Proof Given in Section 4.E of the SM.

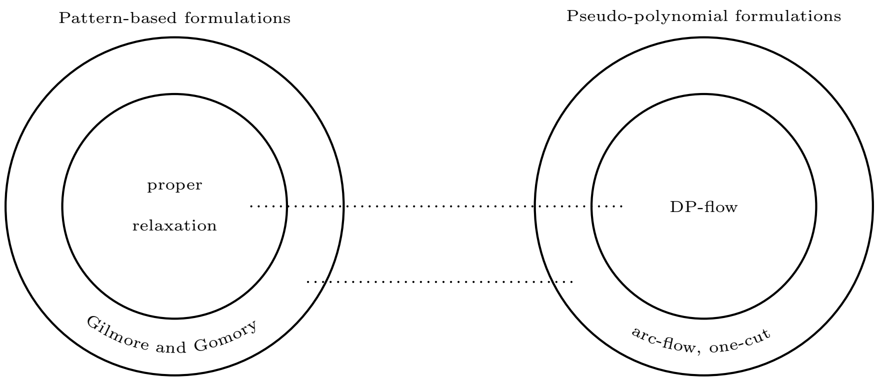


Figure 4.4: Graphical representation of relations among CSP formulations.

We conclude this section with some remarks. Figure 4.4 does not depict the "descriptive" formulation of the CSP (using integer variables for the assignments items-bins and binary variables for the bins), which is dominated by all other formulations (see, e.g., Martello and Toth [214]). As previously noted, the main pseudo-polynomial formulations were proposed with some reduction criteria. Dyckhoff [111] reduced the size of set $\mathcal{S}$ in (4.3) by forcing $x_j \leq d_j$, thus focusing on the set of so-called *normal patterns* (see Herz [153] and Christofides and Whitlock [64]):

$$\mathcal{N} = \left\{ \bar{w} = \sum\nolimits_{j=1}^{m} w_j x_j,\ \bar{w} \leq c,\ x_j \in \mathbb{N}, x_j \leq d_j \text{ for } j = 1, \ldots, m \right\}. \tag{4.15}$$

This improves F_{OC} but does not make it equivalent to F_{PR}. To state this fact, it is enough to consider that the $L(F_{OC})$ solution of Example 2 already accomplishes with (4.15) but is worse than the $L(F_{DP})$ solution. Valério de Carvalho [278] proposed a set of reduction techniques for F_{AF}, namely, he built the arcs by using a DP algorithm that considers item types according to decreasing width, thus reducing symmetries, and removed loss arcs whose head is lower than or equal to the minimum item width. But also in this case, the continuous relaxation of the resulting formulation is not as strong as that of F_{DP} (once again, the $L(F_{AF})$ solution of Example 2 already accomplishes with the proposed reductions but is worse than the one found by $L(F_{DP})$. Formulation F_{DP} provides a very high-quality lower bound, but at the expenses of an excessive number of variables. The extensive tests in Delorme et al. [98] show indeed that the computational performance of F_{DP} is much weaker than that of F_{AF} and F_{OC} on all CSP benchmarks.

Other formulations which are equivalent to F_{AF} but have a better computational performance have been recently presented by Brandão and Pedroso [45] and Côté and Iori [84]. Brandão and Pedroso [45] proposed, among other improvements, a lifting procedure that is based on the fact that the packing an item in a given position might lead to an unused capacity in the bin. For each possible packing, the procedure estimates the minimum certified unused capacity by solving a subset sum problem, and then makes use of this value to extend the head of the arc that corresponds to the packing. The underlying graph has to be modified into a multi-graph, because arcs having the same widths may now correspond to different items, but the process leads to a speed-up in the solution time.

Côté and Iori [84] considered the normal patterns in (4.15), but conveniently decreased their number by means of a *meet-in-the-middle* procedure. Instead of performing a classical DP, they solved a two-way DP which created valid patterns starting from the left and from the right. They then built an arc-flow formulation that makes use of this reduced set of patterns, thus requiring less variables, less constraints, and a quicker solution time.

Côté and Iori [84] also proposed to modify F_{AF} by removing the unit-width loss arcs $(d, d+1)$ and considering only loss arcs that connect two consecutive vertices $d, e \in \mathcal{N}$. If $c \notin \mathcal{N}$, a final loss arc connects the last normal pattern vertex to c (see top part of Figure 4.5). Note that this leads to reducing constraints (4.8) to $e \in \mathcal{N} \cup \{c\}$, and adopting a multi-graph structure for the model (because there can be standard and reflected arcs having the same head and tail values). From now on we consider this straightforward improvement in all our formulations.

4.4 Reflect, an improved arc-flow formulation

In this section, we propose a new arc-flow formulation, called *reflect* (F_{RE}), which models a CSP instance by considering only half of the bin capacity, thus resulting in a sharp decrease in the required number of variables and constraints. The two main features of F_{RE} are:

- In terms of vertices, F_{RE} considers only those corresponding to normal patterns with size smaller than $c/2$ (including 0), plus an additional vertex, called R, corresponding to the size $c/2$;
- In terms of arcs, F_{RE} considers the same ones of F_{AF}, but: (i) it "reflects" each item arc (d, e) having $d < c/2$ and $e > c/2$ into an arc $(d, c - e)$; (ii) removes all item and loss arcs (d, e) having $d \geq c/2$; and (iii) creates a last loss arc by connecting the right most vertex before R with R.

Intuitively, a path in F_{AF} becomes in F_{RE} *a pair of colliding paths*, i.e., two paths both starting in 0 and ending in the same vertex, but only one of the two passing through R. For Example 2, the arcs required by F_{AF} and F_{RE} are shown in Figure 4.5. F_{AF} contains 6 items arcs and 5 loss arcs. To build F_{RE}, (i) we reflect item arcs (0,7) in (0,4) and (4,7) in (4,4); (ii) we remove item arcs (7,10) and (7,11) and loss arcs (7,10) and (10,11); and (iii) we replace loss arc (4,7) with (4,R), thus resulting in 4 item arcs and 3 loss arcs. The reduction is small for this toy instance, but can be impressive for large-size instances (see Section 4.6).

Before presenting a formal mathematical model, we first define the multi-graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ that is at the basis of F_{RE}. The set of vertices is $\mathcal{V} = \{0\} \cup \{e \in \mathcal{N}, 0 < e < c/2\} \cup \{c/2\}$. The set or arcs $\mathcal{A}$ is partitioned into $\mathcal{A}_s$ and $\mathcal{A}_r$, where $\mathcal{A}_s$ denotes the set of *standard* arcs, i.e., all those item and loss arcs that proceed from left to right as in F_{AF}, and $\mathcal{A}_r$ denotes the set of *reflected* arcs, i.e., those item arcs (d, e) from F_{AF} that have been reflected into item arcs $(d, c - e)$. Each arc in $\mathcal{A}_s$ is defined by the triplet (d, e, s), whereas each arc in $\mathcal{A}_r$ by the triplet (d, e, r) (note indeed there can be a standard and a reflected arc having the same head and tail values). We also include in $\mathcal{A}_r$ an arc $(c/2, c/2, r)$ to express pairs of paths that collide in $c/2$. We use (d, e, κ) to denote a generic arc belonging to either $\mathcal{A}_s$ or $\mathcal{A}_r$ and $\mathcal{A}_j$ to define the subset of item arcs associated with item type j, formally $\mathcal{A}_j = \{(d, d + w_j, s) \in \mathcal{A}_s\} \cup \{(d, c - (d + w_j, r)) \in \mathcal{A}_r\}$. Let also $\delta_s^-(e) \subseteq \mathcal{A}_s$ (respectively, $\delta_r^-(e) \subseteq \mathcal{A}_r$) denote the subset of standard (respectively, reflected) arcs that

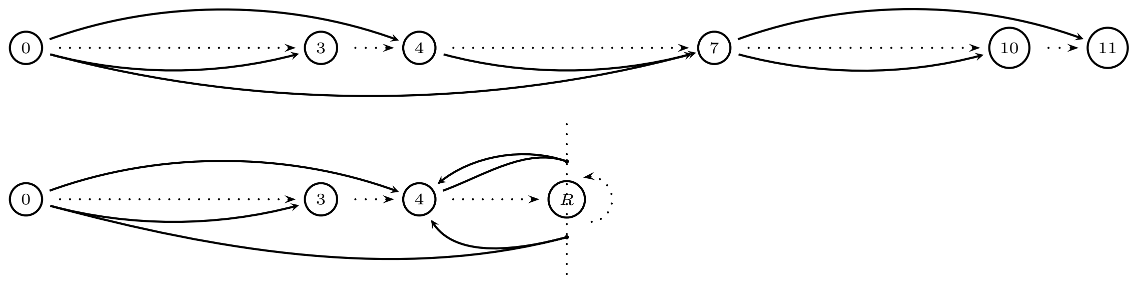


Figure 4.5: Set of arcs required by the standard arc-flow (above) and by reflect (below) for Example 2 (item arcs are depicted in straight lines, loss arcs in dotted lines)

enters e. By associating an integer decision variables $\xi_{de\kappa}$ to each arc $(d, e, \kappa) \in \mathcal{A}$, we can model the CSP as

$$(F_{RE}) \quad \min \quad z = \sum_{(d,e,r)\in\mathcal{A}_r} \xi_{der} \tag{4.16}$$

$$\text{s.t.} \quad \sum_{(d,e,s)\in\delta_s^-(e)} \xi_{des} = \sum_{(d,e,r)\in\delta_r^-(e)} \xi_{der} + \sum_{(e,f,\kappa)\in\delta^+(e)} \xi_{ef\kappa} \qquad e \in \mathcal{N}, 0 < e < c/2 \tag{4.17}$$

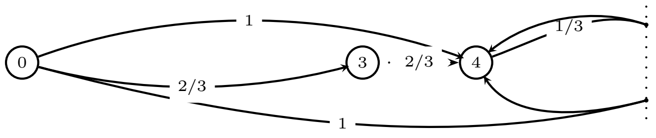
$$\sum_{(0,e,\kappa)\in\delta^+(0)} \xi_{0e\kappa} = 2 \sum_{(d,e,r)\in\mathcal{A}_r} \xi_{der} \tag{4.18}$$

$$\sum_{(d,e,\kappa)\in\mathcal{A}_j} \xi_{de\kappa} \geq d_j \qquad j = 1, \ldots, m \tag{4.19}$$

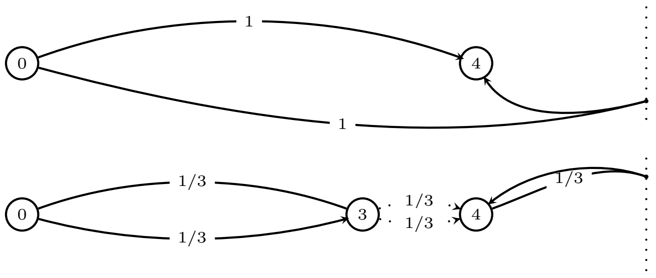
$$\xi_{de\kappa} \in \mathbb{N} \qquad \kappa \in \{s, r\}, (d, e, \kappa) \in \mathcal{A}_\kappa \tag{4.20}$$

The objective function (4.16) minimizes the number of reflected arcs, which is equivalent to the number of bins. Constraints (4.17) ensure that the amount of flow from standard arcs entering a node e is equal to the amount of flow (for both standard and reflected arcs) emanating from e plus the amount of flow from reflected arcs entering e. Constraint (4.18) impose boundary conditions by enforcing the amount of flow emanating from of 0 to be twice the amount of bins used. Constraints (4.19) ensure that the demand of each item type is fulfilled.

An optimal $L(F_{RE})$ solution for Example 2 having value 4/3 is given in Figure 4.6-(a), and is decomposed in the pairs of colliding paths shown in Figure 4.6-(b). The first bin contains an item of width 4 and another of width 7, and is made by the standard arc $(0, 4, s)$ and the reflected arc $(0, 4, r)$, which collide in 4. The second bin is constructed by the standard arcs $(0, 3, s)$ and $(3, 4, s)$, and the reflected arc $(4, 4, r)$. Note that the two arcs $(0, 3, s)$ and $(3, 4, s)$ are both depicted twice in the figure, to show that the flow on these arcs is split to form the two colliding paths: the first path is $\{(0, 3, s), (3, 4, s)\}$, and the second is $\{(0, 3, s), (3, 4, s), (4, 4, r)\}$, both with flow 1/3. Note also that, if no reflected arc enters e, then e is just a partial filling of one or more bins (e.g., vertex 3 in the example), if instead some reflected arcs enter e, then e is a vertex of collision for one or more bins (e.g., vertex 4).



(a) Solution of $L(F_{RE})$



(b) Solution of $L(F_{RE})$ decomposed into pairs of colliding paths

Figure 4.6: Solution of $L(F_{RE})$ for Example 2 (selected item arcs are depicted in straight lines, selected loss arcs in dotted lines, variable values on the arcs)

The following result proves the correctness of F_{RE}.

THEOREM 4 *F_{RE} models the CSP.*

Proof. Proof Given in Section 4.F of the SM.

For the sake of completeness, we provide in Algorithm 7 (Section 4.G of the EC) the required steps to construct the multigraph required by F_{RE}. In the same section, we also provide, in Algorithm 8, the procedure that we use to decompose a F_{RE} (or $L(F_{RE})$) solution into pairs of colliding paths. One can notice that Algorithm 7 does not create any reflected arc (d, e, r) having $d > e$ (step 16 of the algorithm). This reduction criterion is motivated by the following result.

THEOREM 5 *Any feasible pattern can be represented in F_{RE} by a pair of colliding paths whose reflected arc (d, e) has $d \leq e$.*

Proof. Proof Given in Section 4.H of the SM.

4.4.1 Adapting reflect to solve large size instances: Reflect+

Even if the number of arcs used by reflect is considerably reduced with respect to those used by arc-flow, some instances with huge capacity and many small items may still generate models that contain millions of variables and are thus too difficult to tackle. To overcome this issue, we propose some lower and upper bounding techniques and embed them into a new algorithm, called *reflect+*, that produces solutions of very good quality even for difficult instances.

Column generation. We first solve $L(F_{PR})$, the linear relaxation of (4.2), by means of a standard column generation technique. The reduced master problem is initialized with the identity matrix and solved as a linear program to obtain dual variable values $\bar{\pi}_j$ for each item type j. Columns with negative reduced costs are found and added to the reduced master on the fly, by solving a knapsack subproblem, until a proof of optimality is reached. For the subproblem, we make use of `combo` by Martello et al. [213], which solves the binary knapsack. We first use `combo` as a heuristic by feeding it with m items j of profit $\bar{\pi}_j$ and weight w_j (just one item per item type). If this attempt fails in finding a negative reduced cost column, then we use `combo` as an exact approach by feeding it with the entire set of items, but invoking a binary expansion (see, e.g., Vanderbeck and Wolsey [287]): each item type j having demand d_j is represented by $\lfloor \log d_j \rfloor + 1$ items having profit $2^k \pi_j$ and weight $2^k w_j$, for $k = 0, 1, \dots, \lfloor \log d_j \rfloor - 1$. To avoid patterns that contain more than d_j items, the values of the last item $l = \lfloor \log d_j \rfloor$ are set to $(d_j - (2^l - 1))\pi_j$ for the price and $(d_j - (2^l - 1))w_j$. The first heuristic has the purpose of avoiding patterns with many small items that can appear in early iterations of the column generation approach and slightly deteriorate our successive upper bounding procedure.

Let $LB = \lceil L(F_{PR}) \rceil$ denote the lower bound that we obtained, $P_A \subseteq P'$ the set of columns that have been generated to reach linear optimality, and $P_B \subseteq P_A$ the set of columns that belong to the optimal basis. A classical way to obtain an upper bound from this information is to solve to optimal integrality the restricted master problem with the set P_A of columns. This heuristic is easy to implement but might produce low quality solutions, and several attempts have been proposed for trying to improve it (see, e.g., Sadykov et al. [244]). Here we propose a simple yet effective improvement, that consists in solving F_{RE} with a small time limit on a multigraph that contains only the arcs produced by P_A. In detail, we consider all items contained in a column p by non-increasing weight and generate an arc in the F_{RE} multigraph for each item in that order. We repeat the process for all $p \in P_A$ and then solve F_{RE} on this reduced graph.

Our method is motivated by the following remark: Let $z(F_{PR}(P_A))$ be the optimal solution value of the restricted F_{PR} that contains only the columns in P_A, and $z(F_{RE}(P_A))$ be the optimal solution value of the restricted F_{RE} that contains only the arcs produced by the columns in P_A, then $z(F_{RE}(P_A)) \leq z(F_{PR}(P_A))$. The remark follows from the fact all patterns $p \in P_a$ can be produced by F_{RE}, but F_{RE} can also produce patterns that do not belong to P. Consider for example the bottom part of Figure 4.5 that may be obtained through the mapping of patterns (1,0,1) and (0,1,1). It is possible for F_{RE} to produce the additional pattern (1,1,0) through arcs {(0,4,r),(0,4,s)}. Note that it could also produce the non-proper patterns (0,2,1) through arcs {(0,4,s),(4,4,r),(0,4,s)}, (0,1,2) through arcs {(0,4,s),(4,4,r),(0,3,s), (3,4,s)}, and (0,3,0) through arcs {(0,3,s),(3,4,s),(4,4,r),(0,3,s),(3,4,s)}, thus providing a large number of possible heuristic solutions. In our implementation we first solve $z(F_{PR}(P_B))$, and then $z(F_{PR}(P_A))$ if needed.

Node deactivation and dual cuts. We solve $L(F_{RE})$ with the complete set of arcs, and then use its linear solution $\bar{\xi}$ as a basis for our second heuristic attempt. This consists in first splitting the set of vertices in $\mathcal{V} = \mathcal{V}_a \cup \mathcal{V}_n$, where $\mathcal{V}_a = \{d \in \mathcal{V} : \exists \bar{\xi}_{(d,e,\kappa)} > \epsilon\}$ is the set of *active* vertices, and $\mathcal{V}_n = \{d \in \mathcal{V} : \nexists \bar{\xi}_{(d,e,\kappa)} > \epsilon\}$ is the set of *non-active* vertices. Then, we solve F_{RE} with the additional set of constraints

$$\xi_{de\kappa} = 0 \quad d \in \mathcal{V}_n, \kappa \in \{s, r\}, (d, e, \kappa) \in \mathcal{A}_\kappa \tag{4.21}$$

Constraints (4.21) make the solution of the model much faster but might remove too many feasible solutions. We experimentally noticed that better solutions could be found by allowing some large items to be split into smaller items. To this aim, we create a set T of possible transformations (i, j, k), in which $i, j, k = 1, \ldots, m,$, $i < j \leq k$, and $w_i = w_j + w_k$. We add to the model a set of integer variables t_{ijk}, for $(i, j, k) \in T$, each counts the number of times an item i is transformed into items j and k, we replace (4.19) with

$$\sum_{(d,e,\kappa)\in\mathcal{A}_j} \xi_{de\kappa} + \sum_{(i,j,k)\in T} t_{i,j,k} + \sum_{(i,k,j)\in T} t_{i,k,j} - \sum_{(j,k,l)\in T} t_{j,k,l} \geq d_j \quad j = 1,\dots,m \tag{4.22}$$

$$t_{ijk} \in \mathbb{N} \qquad (i,j,k) \in T \tag{4.23}$$

and then solve model (4.16)–(4.18), (4.20)–(4.23). This procedure is reminiscent of the dual cuts by Valério de Carvalho [280].

Arc deactivation. Our third heuristic attempts to find a solution having exactly value $LB = \lceil L(F_{RE}) \rceil$. To this aim, we gather in a set $\mathcal{A}_z$ all arcs whose reduced cost is greater than $LB - L(F_{RE}) + \epsilon$, and restrict the F_{RE} model by setting

$$\xi_{de\kappa} = 0 \quad (d,e,\kappa) \in \mathcal{A}_z \tag{4.24}$$

Indeed, the selection of one or more of arcs in $\mathcal{A}_z$ would lead to a solution of value greater than LB. We then solve (4.16)–(4.20) and (4.24). If no solution of value LB is found, we increase LB by one unit, update $\mathcal{A}_z$, and iterate the process. Note that, if the MIRUP conjecture holds, then the process is iterated at most once.

The resulting algorithm reflect+ makes use of these techniques in the order in which we presented them. An informal pseudocode is given in Algorithm 1. The algorithm stops as soon as upper and lower bound values are equal. Each time a model is solved as an MILP, it is allowed only a restricted execution time, as discussed in Section 4.6 below.

Algorithm 1 `reflect+`

```
 1: solve $L(F_{PR})$ through column generation and obtain $P_A$, $P_B$, and $LB = \lceil L(F_{PR}) \rceil$
 2: solve $F_{RE}(P_B)$ and obtain $U_1$
 3: solve $F_{RE}(P_A)$ and obtain $U_2$
 4: solve $L(F_{RE})$ and obtain $\bar{\xi}$, $\mathcal{V}_n$, and $\mathcal{A}_z$
 5: solve $F_{RE}$ + (4.21) − (4.23) and obtain $U_3$
 6: $UB \leftarrow \min(U_1, U_2, U_3)$
 7: while $UB > LB$ do
 8:     solve $F_{RE}$ + (4.24) and obtain $U_4$
 9:     $UB = \min(UB, U_4)$
10:     update $\mathcal{A}_z$ and set $LB \leftarrow LB + 1$
11: end while.
```

4.5 Generalizations

In this section, we detail the modifications that we used on reflect to handle the *variable sized bin packing problem* (VSBPP) and the *bin packing problem with item fragmentation* (BPPIF).

4.5.1 Variable sized BPP

In the VSBPP, instead of a unique bin of capacity c, we are given a set of K bin types with capacity c_k, price p_k, and availability b_k, $k = 1, \dots, K$. The objective is now to pack all the items into a minimum cost set of bins that respects the availability and the capacity of each bin type. The problem was extensively studied in the literature and various exact and heuristic approaches were proposed, see, e.g., Belov and Scheithauer [29], Alves and Valério de Carvalho [10], and Hemmelmayr et al. [152]).

Valério de Carvalho [279] described a possible adaptation for the arc-flow model in which he considered some additional integer variables for each bin type, modified accordingly the objective function (4.7) and the flow conservation constraints (4.8), and added a set of specific constraints that limit to b_k the number of times bin type k is selected. Such an extension is not as straightforward for reflect, as we have to ensure that the variables associated with the bin types are selected exactly once per pair of colliding paths. To satisfy this constraint, we integrate the bin type decision into the reflected arcs.

To this aim, we partition the set of reflected arcs $\mathcal{A}_r$ into $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_K$. Each arc in $\mathcal{A}_k$ is defined by the triplet (d, e, k), where $k = 1, \dots, K$ represents the bin type.

Example 3 *A VSBPP instance with* $K = 2$, $m = 3$, $w = (7, 4, 3)$, $d = (1, 1, 1)$, $c = (11, 6), p = (11, 6)$, *and* $b = (2, 2)$.

For Example 3, the arcs required by reflect are shown in Figure 4.7. The construction is very similar to the one of Example 2, but as we have to consider an additional bin of size 6, node 3 becomes a reflection point. Thus, a copy of arc (0,4,s) is reflected in 3 and creates the new reflected arc $(0, 3, r)$, where $r = 2$ and corresponds to a reflection of the second bin type. To ensure the feasibility of the model, an additional arc (3,3,2) is also created.

By introducing W_k, a set of integer variables that counts the number of times bin type

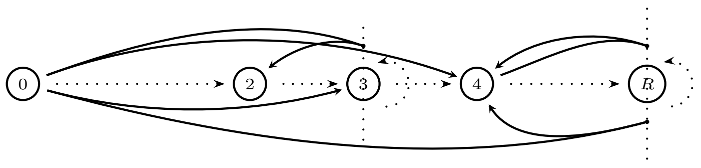


Figure 4.7: Set of arcs required by reflect for Example 3 (item arcs are depicted in straight lines)

k is selected, for $k = 1, \ldots, K$, the VBSPP can be modelled as

$$\min \quad z = \sum_{k=1}^{K} p_k \, W_k \tag{4.25}$$

$$\text{s.t.} \quad \sum_{(d,e,s)\in\delta_s^-(e)} \xi_{des} = \sum_{(d,e,r)\in\delta_r^-(e)} \xi_{der} + \sum_{(e,f,\kappa)\in\delta^+(e)} \xi_{ef\kappa} \quad e \in \mathcal{N}, 0 < e < c/2 \tag{4.26}$$

$$\sum_{(0,e,\kappa)\in\delta^+(0)} \xi_{0e\kappa} + \sum_{(d,0,r)\in\delta^-(0)} \xi_{d0r} = 2\sum_{k=1}^{K} W_k \tag{4.27}$$

$$\sum_{(d,e,k)\in\mathcal{A}_k} \xi_{d,e,k} = W_k \quad k = 1, \ldots, K \tag{4.28}$$

$$W_k \le b_k \quad k = 1, \ldots, K \tag{4.29}$$

$$\sum_{(d,e,\kappa)\in\mathcal{A}_j} \xi_{de\kappa} \ge d_j \quad j = 1, \ldots, m \tag{4.30}$$

$$\xi_{de\kappa} \in \mathbb{N} \quad \kappa \in \{s, r\}, (d, e, \kappa) \in \mathcal{A}_\kappa \tag{4.31}$$

$$W_k \in \mathbb{N} \quad k = 1, \ldots, K \tag{4.32}$$

The objective function (4.25) minimizes now the price of the bins selected, while constraints (4.26) remain unchanged. Constraints (4.27) still force the amount of flow emanating from 0 to be twice the number of bins used, but takes into account the fact that it is now possible for a reflected arcs to directly enter vertex 0. Constraints (4.28) match the variables W_k with the sum of the variables associated with their corresponding reflected arcs having $r = k$, and constraints (4.29) ensure that the bin availabilities are respected. Note that variables W_k are not strictly mandatory in the model and could be replaced by $\sum_{(d,e,k)\in\mathcal{A}_k} \xi_{d,e,k}$, but there use proved to be computationally useful, especially for the instances where p_k are not proportional to c_k.

4.5.2 BPP with item fragmentation

In the BPPIF, the constraint imposing that an item has to be packed into a unique bin is removed. Now an item is allowed to be fractionally packed into different bins, as long as the sum of the fractions is equal to the weight of the item. Several variants of the BPPIF have been studied in the literature, see, e.g., the recent work by Casazza and Ceselli [57]. In this section, we study two variants of the BPPIF: one whose objective is to minimize the number of bins used while the number of fragmentation is limited (bm-BPPIF) and another one whose objective is to minimize the number of fragmentation while the number of bins is limited (fm-BPPIF). We first propose an extension of the classical arc-flow model for the BPPIF and then we show how the VSBPP can be used to derive valid upper bounds for the BPPIF.

In contrast with reflect, where the arcs were reflected into the first half of the bin, now we develop a version of arc-flow which allows the arcs to go over the capacity of the bin c. When such an arc $(d,e), e > c$ is created, its tail e is transposed into $e \bmod c$. We gather in $\mathcal{A}_t$ the set of transposed arcs.

Example 4 *A BPPIF instance with $m = 3$, $w = (7,4)$, $d = (2,1)$, and $c = 11$.*

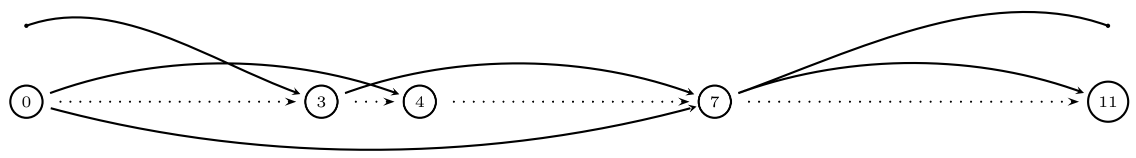


Figure 4.8: Set of arcs built by arc-flow for the BPPIF Example 4 (item arcs are depicted in straight lines)

For Example 4, the arcs required by the adaptation of arc-flow are shown in Figure 4.8. A first arc (0,7) is created by the first item of width 7. An arc (7,14) is created by the second item of width 7 and is transposed into (7,3). Then, arcs (7,11),(3,7), and (0,4) are created by the item of width 4. Finally, loss arcs link all the active nodes to the following one. For the bm-BPPIF, if we denote by F the maximum number of fragmentation allowed, the problem can be modelled as follow

$$\min \quad z + \sum_{(d,e)\in\mathcal{A}_t} x_{de} \tag{4.33}$$

$$\text{s.t.} \quad \sum_{(e,f)\in\delta^+(e)} x_{ef} - \sum_{(d,e)\in\delta^-(e)} x_{de} = \begin{cases} z & \text{if } e = 0 \\ -z & \text{if } e = c \\ 0 & \text{for } e \in \mathcal{N}, 0 < e < c/2 \end{cases} \tag{4.34}$$

$$\sum_{(d,d+w_i)\in\mathcal{A}} x_{d,d+w_i} \geq d_i \qquad i = 1,\ldots,m \tag{4.35}$$

$$\sum_{(d,e)\in\mathcal{A}_t} x_{de} \leq F \tag{4.36}$$

$$x_{de} \in \mathbb{N} \qquad (d,e) \in \mathcal{A}. \tag{4.37}$$

The objective function (4.33) minimizes the number of bins, that is equal to the number of flow emanating from 0 plus the amount of transposed flows. While the flow conservation constraints (4.34) and the item satisfaction constraints (4.35) remain unchanged, constraint (4.36) ensures that the number of fragmentation is not greater than F. The adaptation of the model to the fm-BPPIF is very straightforward as it is sufficient to consider the number of fragmentation $\sum_{(d,e)\in\mathcal{A}_t} x_{de}$ in the objective function (4.33) and transform (4.36) to $z + \sum_{(d,e)\in\mathcal{A}_t} x_{de} \leq B$, where B is the maximum number of bins allowed.

As it will be shown in the experimental evaluation in Section 4.6, this approach does not lead to outstanding results, but is useful to get good quality lower bounds. To obtain good quality upper bounds, we solve with reflect a VSBPP in which the price of a bin is equal to the number of fragmentations it represents, and its capacity is set to a multiple of c. We create K bins of capacity $c_k = k\ c$ and price $p_k = k - 1$ for $k = 1,\ldots,K$ and model the fm-BPPIF as

$$\min \quad z = \sum_{k=1}^{K} p_k\ W_k \tag{4.38}$$

$$\text{s.t.} \quad \sum_{(d,e,s)\in\delta_s^-(e)} \xi_{des} = \sum_{(d,e,r)\in\delta_r^-(e)} \xi_{der} + \sum_{(e,f,\kappa)\in\delta^+(e)} \xi_{ef\kappa} \quad e \in \mathcal{N}, 0 < e < c/2 \tag{4.39}$$

$$\sum_{(0,e,\kappa)\in\delta^+(0)} \xi_{0e\kappa} + \sum_{(d,0,r)\in\delta^-(0)} \xi_{d0r} = 2\sum_{k=1}^{K} W_k \tag{4.40}$$

$$\sum_{(d,e,k)\in\mathcal{A}_k} \xi_{d,e,k} = W_k \qquad k = 1,\ldots,K \tag{4.41}$$

$$\sum_{k=1}^{K} k\ W_k \leq B \tag{4.42}$$

$$\sum_{(d,e,\kappa)\in\mathcal{A}_j} \xi_{de\kappa} \geq d_j \qquad j = 1,\dots,m \tag{4.43}$$

$$\xi_{de\kappa} \in \mathbb{N} \qquad \kappa \in \{s,r\}, (d,e,\kappa) \in \mathcal{A}_\kappa \tag{4.44}$$

$$W_k \in \mathbb{N} \qquad k = 1,\dots,K \tag{4.45}$$

Objective function (4.38) minimizes the number of fragmentations, while constraints (4.39)-(4.41) and (4.43) remain unchanged. Constraint (4.42) ensures that the total number of bins used does not exceed the bin limit. Again, the adaptation of the model for the bm-BPPIF is easy as it only requires to consider the total number of bins $\sum_{k=1}^{K} k\ W_k$ in (4.38) and transform (4.42) into $\sum_{k=1}^{K} p_k\ W_k \leq F$.

When used with sufficiently large value of K (for example, $K = (\sum_{j=1}^{m} d_j) - 1$), this approach is exact. However, as huge values of K implies huge capacities, it is not advisable to use this approach to solve exactly the BPPIF. According to our experimental results, it is generally enough to consider restricted values, e.g., $K \leq 3$, to solve most of the benchmark instances to proven optimality.

4.6 Computational results

In this section, we experimentally compare the efficiency of reflect and its improvements with the classical arc-flow and the best algorithms that have been proposed in the literature on diverse benchmark instances. All our experiments were executed on an Intel Xeon 3.10 GigaHertz with 8 GigaByte RAM, equipped with four cores, and we used Gurobi 6.5 as MILP solver. All our experiments were performed with a single core, and the number of threads was set to one for the MILP solver. In each table, the formulation that finds the largest number of optimal solutions is highlighted in bold, and when an instance is not solved, its associated time is set to the time limit.

4.6.1 Results on BPP and CSP

We used three BPP and CSP benchmark sets:

- Literature instances (1), a set of 1615 instances that have been proposed in various articles related to the BPP these last decades. These instances have diverse characteristics and a complete description on their parameters can be found in Delorme et al. [98].

- Difficult instances (2), two classes of 100 instances each that were proposed in Delorme et al. [98]. The instances of the first class, called ANI instances, do not have the integer round up property (IRUP), and their difficulty resides in raising the continuous lower bound. The 50 first instances have $n = 201$ and $c < 2500$, while the 50 last instances have $n = 402$ and $c < 10000$. Each instance of the first set has a copy in the second set in which exactly one item has been split into two smaller pieces so that the IRUP is recovered.

- Large capacitated instances (3), 4 sets of 60 instances each that were proposed in Gschwind and Irnich [142]. The instances are uniformly randomly generated following 2 parameters, $c \in \{500\,000, 1\,500\,000\}$ and $[w_{min}, w_{max}] \in \{[(2/15)c, (2/3)c], [(1/150)c, (2/3)c]\}$. For each duet, 60 instances were generated, 20 for each value of $m \in \{125, 250, 500\}$.

Table 4.1 provides the results obtained by running F_{RE} and F_{PR} with the restricted set of patterns P_A and P_B and a time limit of 300 seconds. The 2 first columns identify the benchmark and the corresponding number of instances. Each of the 4 following set of columns associate with each formulation the number of instances that were solved within the time limit, the average CPU time expressed in seconds, and the number of times the solution obtained was optimal (i.e., matched with $\lceil L(F_{PR}) \rceil$).

Table 4.1 shows that, when used on a restricted set of patterns P', $F_{PR}(P')$ is clearly outperformed by $F_{RE}(P')$. Sometimes, e.g., for Waescher instances, $F_{PR}(P')$ is very fast, but terminates with a solution that does not match with $\lceil L(F_{PR}) \rceil$ while some other times, e.g., for Scholl 3 instances, $F_{PR}(P')$ does not finish within the time limit. We obtained outstanding results for both $F_{RE}(P_B)$ and $F_{RE}(P_A)$, as they can close all but 30 tested instances. When it comes to select which restricted set of patterns should be chosen for $F_{RE}(P')$, no clear conclusion can be drawn from the results: indeed, it looks like $F_{RE}(P_B)$ is faster, especially for instances with very large capacity. However, $F_{RE}(P_A)$ terminates more often with a proven optimal solution, especially for the triplet instances Falkenauer T.

Table 4.2 provides the results obtained by running F_{AF} and F_{RE} with their complete set of arcs and a time limit of 3600 seconds. The 2 first columns identify the benchmark and the corresponding number of instances. Each of the 2 following set of columns associate with each formulation the number of instances that were solved to proven optimality, the average CPU time expressed in seconds, the number of variables, and the number of constraints

Table 4.1: Evaluation of heuristic 1 with restricted sets of patterns P_1 and P_2 for the CSP

Set of instances	# inst.	$F_{RE}(P_B)$			$F_{PR}(P_B)$			$F_{RE}(P_A)$			$F_{PR}(P_A)$		
		# term.	time	# opt	# term.	time	# opt	# term.	time	# opt	# term.	time	# opt
Waescher	17	15	72.9	12	17	3.5	2	15	100.3	**14**	16	26.8	6
Hard28	28	28	0.5	10	28	0.8	2	28	5.5	**19**	28	1.1	9
Falkenauer U	80	80	0.0	**80**	80	0.1	44	80	0.1	**80**	80	0.2	66
Falkenauer T	80	80	0.2	16	80	10.2	16	80	1.4	**78**	80	4.4	16
Schwerin 1	100	100	0.1	**100**	100	0.6	0	100	0.2	**100**	100	31.2	16
Schwerin 2	100	100	0.1	**100**	100	1.1	0	100	0.2	**100**	97	29.6	47
Scholl 1	720	720	0.0	719	720	0.1	449	720	0.0	**720**	720	0.3	523
Scholl 2	480	479	3.7	**479**	339	97.0	21	477	6.8	477	257	156.2	116
Scholl 3	10	10	0.8	**10**	6	205.7	0	10	8.8	**10**	0	300.1	0
Total (1)	1615	1612	1.9	1526	1470	30.8	534	1610	3.3	**1598**	1378	52.7	799
Inrich AA	60	60	1.4	**60**	39	108.6	7	60	10.0	**60**	48	92	38
Inrich AB	60	60	3.6	**59**	34	145.9	8	54	90.1	54	34	151.2	33
Inrich BA	60	60	1.2	59	40	109.1	9	60	7.7	**60**	45	92.3	35
Inrich BB	60	59	8.0	**59**	36	144.4	7	53	98.1	53	34	149.3	32
Total (3)	240	239	3.6	**237**	149	127.0	31	227	51.5	227	161	121.2	138
Total (1)+(3)	1855	1851	2.1	1763	1619	43.3	565	1837	9.6	**1825**	1539	61.5	937

involved in the model.

Table 4.2 shows that F_{RE} clearly outperforms F_{AF} both in terms of number of optimal solution found and in terms of required time. This can be explained by the drastic variable and constraint reduction obtained in reflect, with an average of 81.2% of arc reduction (from 64.5% for the AI 400 instances up to 98% for the Scholl 3 instances) and an average of 64.1% of constraint reduction (from 31.9% for the Scholl 1 instances up to 92.2% for the Inrich BA instances). However, even with these considerable reductions, reflect cannot handle the millions of variables and the hundreds of thousands constraints from the Irnich AB and Irnich BB instances.

Table 4.3 provides the results obtained by running the branch-and-cut-and-price by Belov and Scheithauer [30], vpsolver by Brandão and Pedroso [45], reflect, and reflect+ with a time limit of 3600 seconds. As their codes are available online, we rerun both algorithms on our machine. As required by the original codes, Cplex 12.6 was used for the branch-and-cut-and-price and Gurobi 6.5 was used for vpsolver. The 2 first columns identify the benchmark and the corresponding number of instances. Each of the 3 following set of columns associate with each code the number of instances that were solved to proven optimality and the average CPU time expressed in seconds. In addition, for reflect+,

Table 4.2: Evaluation of reflect with respect to arc-flow for the CSP

Set of instances	# inst.	arc-flow				reflect			
		# opt	time	nb. var.	nb. cons.	# opt	time	nb. var.	nb. cons.
Waescher	17	9	1780.5	174 722	9256	**17**	555.5	52 006	4257
Hard28	28	**28**	10.9	36 816	1134	**28**	19.0	10 932	635
Falkenauer U	80	**80**	0.1	3023	205	**80**	0.0	765	131
Falkenauer T	80	**80**	1.3	16 246	735	**80**	0.3	2490	332
Schwerin 1	100	**100**	0.9	11 636	733	**100**	0.3	3408	287
Schwerin 2	100	**100**	0.7	12 442	739	**100**	0.2	3664	292
Scholl 1	720	**720**	0.1	1735	166	**720**	0.0	510	113
Scholl 2	480	**480**	84.3	39 307	938	**480**	9.6	13 187	453
Scholl 3	10	**10**	324.2	1 529 969	49 268	**10**	8.6	29 938	10 923
Total (1)	1615	1607	46.2	26 853	913	**1615**	9.1	5668	367
AI 200	50	**50**	233.7	121 251	2249	**50**	45.5	42 803	1140
AI 400	50	19	2461.3	940 036	7686	**21**	2297.4	335 723	3768
ANI 200	50	35	1397.7	119 496	2245	**50**	67.2	42 021	1136
ANI 400	50	3	3474.4	935 117	7683	**10**	3083.6	334 149	3765
Total (2)	200	107	1891.8	528 975	4966	**131**	1373.4	188 674	2452
Inrich AA	60	20	307.6	4 878 777	205 208	**60**	179.8	82 069	23 436
Inrich AB	60	0	102.6	19 852 031	441 111	0	1013.9	4 493 237	191 111
Inrich BA	60	20	343.5	10 113 134	510 332	**60**	380.1	99 396	39 627
Inrich BB	60	0	2915.9	52 020 717	1 281 154	0	121.5	11 271 290	531 165
Total (3)	240	40	917.4	21 716 164	614 405	**120**	423.8	3 986 498	201 288
Total (1)+(2)+(3)	2055	1754	327.6	2 608 779	72 956	**1866**	190.3	488 393	24 035

we specify the number of times each heuristic could close an instance. Reflect+ follows Algorithm 1 in which we allowed $F_{RE}(P_B)$ (called $H1_B$ in the following) to be run for 60 seconds maximum, $F_{RE}(P_A)$ (or $H1_A$) was not called. 1200 seconds were allowed to F_{RE} + (22) + (23) (or $H2$) and the rest of the time was devoted to F_{RE} + (24) (or $H3$). Considering the limited results of the two last heuristics when too many variables and constraints were involved in the model, we added a specific rule in which, if the number of variables plus 10 times the number of constraints was greater than 1 000 000, no local time limit was imposed to $H1_B$, and in case the solution obtained is not sufficient to close the instance, $H1_A$ was also called with no local time limit.

Table 4.3 shows that reflect+ outperforms on average state-of-the-art algorithms for the CSP and the BPP. While only H1 and H2 are required to find the optimal solutions of benchmark (1) (H3 is required 7 times to prove the optimality of the 7 NIRUP instances of the benchmark), H2 appears to be particularly effective on the AI instances of benchmark (2). Finally, allowing a longer time limit for H1 and running it with both sets of patterns

Table 4.3: Comparison of reflect and reflect+ with literature algorithms for the CSP

Set of instances	# inst.	Belov		vpsolver		reflect		reflect+				
		# opt	time	# opt	time	# opt	time	# opt	time	# $H1$	# $H2$	# $H3$
Waescher	17	**17**	0.1	16	886.2	**17**	555.5	**17**	41.3	7	8	2
Hard28	28	**28**	7.5	**28**	33.0	**28**	19.0	**28**	4.2	0	23	5
Falkenauer U	80	**80**	0.0	**80**	0.1	**80**	0.0	**80**	0.1	79	1	0
Falkenauer T	80	**80**	56.9	**80**	0.4	**80**	0.3	**80**	1.0	10	70	0
Schwerin 1	100	**100**	1.0	**100**	0.3	**100**	0.3	**100**	0.1	100	0	0
Schwerin 2	100	**100**	1.3	**100**	0.3	**100**	0.2	**100**	0.1	100	0	0
Scholl 1	720	**720**	0.0	**720**	0.0	**720**	0.0	**720**	0.1	717	3	0
Scholl 2	480	**480**	0.3	479	107.7	**480**	9.6	**480**	2.8	475	5	0
Scholl 3	10	**10**	14.1	**10**	8.5	**10**	8.6	**10**	3.7	10	0	0
Total (1)	1615	**1615**	3.3	1613	42.1	**1615**	9.1	**1615**	1.5	1498	110	7
AI 200	50	**50**	90.6	**50**	105.8	**50**	45.5	**50**	8.5	1	48	1
AI 400	50	**45**	699.4	36	1430.5	21	2297.4	40	1205	0	30	10
ANI 200	50	**50**	144.2	49	119.5	**50**	67.2	**50**	49.3	0	0	50
ANI 400	50	1	3555.6	11	3170.2	10	3083.6	**17**	2703.9	0	0	17
Total (2)	200	146	1222.5	146	1206.5	131	1373.4	**157**	991.7	1	78	78
Inrich AA	60	**60**	2.8	56	453.8	**60**	179.8	**60**	11.7	60	0	0
Inrich AB	60	**60**	10.9	0	3600.0	0	1013.9	**60**	29.6	60	0	0
Inrich BA	60	**60**	2.8	57	491.6	**60**	380.1	**60**	16.4	59	1	0
Inrich BB	60	**60**	10.5	0	3600.0	0	121.5	**60**	47.5	60	0	0
Total (3)	240	**240**	6.8	113	1968.4	120	423.8	**240**	26.3	239	1	0
Total (1)+(2)+(3)	2055	2001	122.3	1872	372.6	1866	190.3	**2012**	100.7	1738	189	85

when many variables and constraints are involved seems to be worthy for benchmark (3) as reflect+ can solve all instances of the benchmark to proven optimality. When compared with vpsolver, reflect+ always finds more or the same amount of optimal solution in the same or less amount of time. When compared with the branch-and-cut-and-price by by Belov and Scheithauer [30], reflect+ seems less powerful on the AI 400 instances, a set in which the upper bound is very difficult to find. In the opposite, it seems better for the ANI 400 instances, a set in which the lower bound is very difficult to raise.

4.6.2 Results on the VSBPP

We used three VSBPP benchmark sets:

- Belov instances (4), a set of 50 instances that was proposed by Belov and Scheithauer [29]. These instances have $K = 4$, $c_k \in [5000, 10000]$, $b_k \in [625, 2500]$, $p_k = c_k$, $m = 100$, $w_i \in [1, 7500]$, and $d_i \in [1, 100]$. For some instances, some bins of the same sizes were sometimes generated, this explains why it is possible to obtain $K < 4$ and $b_k > 2500$. The same remark holds for the item types.

- Crainic instances (5), three sets instances used in Crainic et al. [87]. The first set was originally proposed in Monacci [220] and is composed of 300 instances with $K \in \{3, 5\}$, $c_k \in \{60, 80, 100, 120, 150\}$, unlimited b_k, $p_k = c_k$, $n \in \{25, 50, 100, 200, 500\}$, and $w_i \in [1, 100]$. The second set was originally proposed in Correia et al. [79] and is composed of 60 instances with $K \in \{3, 12\}$, $c_k \in [25, 300]$, limited b_k, $p_k = 100\ \sqrt{c_k}$, $n \in \{100, 200, 500, 1000\}$, and $w_i \in [1, 20]$. As the instances of both sets are given on a *bin packing* format, no specific information is available on d_i and is expected to be in the order of $n/(w_{\max} - w_{\min})$. The third set was introduced in the paper itself and is composed of 480 instances with various parameters.

- Hemmelmayr instances (6), two sets of instances used in Hemmelmayr et al.[152]. The first was originally proposed in Haouari and Serairi [148] and is composed of 150 instances with $K = 7$, $c_k \in \{70, 100, 130, 160, 190, 220, 250\}$, unlimited b_k, $p_k \in \{c_k, 10\sqrt{c_k}, 0.1\ c_k^{3/2}\}$, $n \in \{100, 200, 500, 1000, 2000\}$, and $w_i \in [1, 250]$. Again, as the instances of the set are given on a bin packing format, no specific information is available on d_i and is expected to be in the order of $n/(w_{\max} - w_{\min})$. The second set is a subset of 50 instances proposed in Monacci [220].

Table 4.4 provides the results obtained by the cutting plane algorithm by Belov and Scheithauer [29], the branch-and-price-and-cut by Alves and Valério de Carvalho [10], the variable neighbourhood search by Hemmelmayr et al. [152], arc-flow, reflect, and reflect+ with a time limit of 3600 seconds. We reran only [29] on our machine, the other results were copied (and uniformized) from the original papers. For [152], we compared the solutions they provided in their paper for each instance with the optimal solution provided by our exact approaches to obtain their number of optimal solutions. The 2 first columns identify the benchmark and the corresponding number of instances. Each of the 5 following set of columns associate with each code the number of instances that were solved to proven optimality and the average CPU time expressed in seconds. Reflect+ follows Algorithm 1 in which we allowed $H1_B$ and $H1_A$ to be run for 60 seconds maximum each. $H2$ was deactivated as it did not appear to be useful for these benchmarks, and the rest of the time was devoted to $H3$. Step 14 is modified and LB is increased to $\min \sum_{k=1}^{K} p_k\ x_k : \sum_{k=1}^{K} p_k\ x_k > LB, x_k < b_k, x_k \in \mathbb{N}\ (k = 1, \ldots, K)$. As some instances involved very few variables and constraints, we add a specific rule in which, if the number of constraints was less than 500, reflect was called instead of reflect+.

Table 4.4: Evaluation of reflect+ with respect to literature algorithms for the VSBPP

Set of instances	# inst.	Belov		Alves		Hemmelmayr		arc-flow		reflect		reflect+	
		# opt	time	# opt	time	# opt	time	# opt	time	# opt	time	# opt	time
Total (4)	50	36	1052.5	47	227.5	-	-	32	1646.7	45	613.2	**50**	3.0
Crainic 1	300	-	-	**300**	0.2	-	-	**300**	1.1	**300**	0.3	**300**	0.3
Crainic 2	60	-	-	-	-	-	-	**60**	3.4	**60**	2.2	**60**	2.5
Crainic 3	480	-	-	-	-	-	-	**480**	0.6	**480**	0.2	**480**	0.2
Total (5)	840	-	-	-	-	-	-	**840**	1	**840**	0.4	**840**	0.4
Hemmelmayr 1	150	-	-	-	-	78	13.4	**150**	2.7	**150**	1.2	**150**	1.1
Hemmelmayr 2	50	-	-	-	-	**50**	0.7	**50**	1.5	**50**	0.5	**50**	0.8
Total (6)	200	-	-	-	-	128	10.3	**200**	2.4	**200**	1	**200**	1
Total (4)+(5)+(6)	1090	-	-	-	-	-	-	1072	76.7	1085	28.6	**1090**	0.6

Table 4.4 shows that pseudo-polynomial formulations are in general very effective for most of the VSBPP instances available in the literature as their capacity is usually very low. In fact, the heuristics of reflect+ were called only for the instances of benchmark (4). $H1_A$ and $H1_B$ found 47 optimal solutions, and H3 was called the 3 remaining times to close the instance. When compared with literature algorithms, reflect+ outperforms the results of every code on every set of instances.

4.6.3 Results on the BPPIF

We used two BPPIF benchmark sets:

- fm-BPPIF (7), a set of 540 instances that was proposed by Casazza and Ceselli [57]. These instances have $c = 1000$, $n \in \{20, 50, 100, 150, 200, 250, 500, 7500, 1000\}$, and $[w_{\min}, w_{\max}] \in \{[0.1c, 0.9c], [0.5c, 0.9c], [0.1c, 0.5c]\}$. The maximal number of allowed bins was set either to $B = \lceil 0.5 * (w_{\min} + w_{\max}) * n/c \rceil$ to obtain *tights* instances, or to $B * 10/9$ to obtain *loose* instances.
- bm-BPPIF (8), a set of 540 instances that was proposed by Casazza and Ceselli [57] and is a copy of benchmark (7). Parameter B is removed and parameter F is set to 0.5 $F*$ where $F*$ is the optimal solution of the corresponding fm-BPPIF instance. In our experiments, for odd values of $F*$, we set F to $\lfloor 0.5\ F* \rfloor$, and we preliminary

solved all fm-BPPIF instances to optimality to obtain the full set of $F*$ values (the longest instance to solve to optimality required 4858.7 seconds).

After solving the linear relaxation of (4.33)-(4.37) to obtain a valid lower bound LB, we transformed the BPPIF instance into a VSBPP instances with K bins, and successively solved (4.38)-(4.45) with a value K restricted to 1, 2, and 3. If LB and the solutions given by the model do not match after $K = 3$, we solved (4.33)-(4.37).

Table 4.5 provides the results obtained by the branch-and-price algorithm of Casazza and Ceselli [57], the adaptation of arc-flow for the BPPIF, and the adaptation of reflect with a time limit of 3600 seconds. The results from the literature were copied from the original papers, and the time was adjusted to incorporate unsolved instances in the average time. The 2 first columns identify the benchmark and the corresponding number of instances. Each of the 3 following set of columns associate with each code the number of instances that were solved to proven optimality and the average CPU time expressed in seconds. In addition, for reflect, we specify the number of times each transformation to the VSBPP with a given value of K was enough to close the instance.

Table 4.5 shows that pseudo-polynomial formulations are also effective for BPPIF instances, as an adaptation of arc-flow can already produce results comparable with the literature. Using the VSBPP to obtain valid upper bounds for the BPPIF seems to be very efficient to solve the problem, as all instances but 22 are closed by our algorithm with a restricted value of $K \leq 3$. The set of instances that appear to be the most difficult for our algorithm are the fm-BPPIF tight instances with $n = 1000$ large items.

4.7 Conclusion

We studied pseudo-polynomial formulations for the one dimensional bin packing and cutting stock problems and gave a complete overview of the dominance and equivalence relations that exist among the main pattern-based and pseudo-polynomial MILP formulations that have been proposed in the literature. We also introduced reflect, a new MILP formulation that uses just half of the bin and needs significantly less constraints and variables than the classical arc-flow. We proposed heuristics and lower bounding techniques that can be used to compensate reflect weaknesses when the capacity of the instance is too hight. In addition, we showed how reflect could be modified to solve the VSBPP and the BPPIF, two relevant variants of the BPP. We tested reflect on benchmark instances of the

Table 4.5: Evaluation of reflect with respect to literature algorithms for the BPPIF

	Set of instances	# inst.	Casazza		arc-flow		reflect					
			# opt	time	# opt	time	# opt	time	# K=1	# K=2	# K=3	# arc-flow
FM	20-100 loose free	30	**30**	2.1	**30**	6.6	**30**	1.3	26	4	0	0
	20-100 loose large	30	**30**	2	**30**	6.1	**30**	1.2	0	30	0	0
	20-100 loose small	30	**30**	0.3	**30**	9.6	**30**	1.1	30	0	0	0
	20-100 tight free	30	**30**	85.7	28	328.8	**30**	28.7	0	13	11	6
	20-100 tight large	30	**30**	81.4	**30**	55.5	**30**	11.1	0	0	18	12
	20-100 tight small	30	**30**	12.4	**30**	193.9	**30**	3.4	15	14	0	1
	150-1000 loose free	60	0	3600	**60**	24.9	**60**	6.1	60	0	0	0
	150-1000 loose large	60	59	1044.6	**60**	42.7	**60**	5.4	0	60	0	0
	150-1000 loose small	60	0	3600	**60**	12.2	**60**	5.4	60	0	0	0
	150-1000 tight free	60	11	3382	11	3080.6	**60**	217	0	58	0	2
	150-1000 tight large	60	39	1885.5	39	1694.6	**59***	621.3	0	0	59	0
	150-1000 tight small	60	0	3600	27	2453.5	**60**	80.6	60	0	0	0
	Total (1)	540	289	1441.3	435	845.4	**539**	106.6	251	179	88	21
BM	20-100 loose free	30	**30**	2.8	**30**	1.2	**30**	0.6	29	1	0	0
	20-100 loose large	30	**30**	1.8	**30**	3.3	**30**	1.4	0	30	0	0
	20-100 loose small	30	**30**	18.1	**30**	35	**30**	1	30	0	0	0
	20-100 tight free	30	**30**	4.3	**30**	9.3	**30**	1.9	11	19	0	0
	20-100 tight large	30	**30**	1.5	**30**	4.1	**30**	1.5	0	30	0	0
	20-100 tight small	30	**30**	3.5	**30**	7.8	**30**	1.6	30	0	0	0
	150-1000 loose free	60	42	1591.8	**60**	6.9	**60**	3.4	60	0	0	0
	150-1000 loose large	60	50	1205.7	**60**	17.5	**60**	7.3	0	60	0	0
	150-1000 loose small	60	30	1950.5	47	1528.2	**60**	16.5	60	0	0	0
	150-1000 tight free	60	44	1891.6	**60**	140	**60**	25.7	0	60	0	0
	150-1000 tight large	60	49	1514.8	**60**	19.8	**60**	6.4	0	60	0	0
	150-1000 tight small	60	48	1453	49	1176	**60**	28.5	60	0	0	0
	Total (2)	540	443	803.3	516	324.3	**540**	10.2	280	260	0	0
	Total (1) + (2)	1080	732	1122.3	951	584.9	**1079**	58.4	531	439	88	21

*remaining instance solved in 4858.7s

three problems and achieved state of the art results, improving upon previous algorithms in the literature and finding several new proven optimal solutions.

Supplementary material 4.A Details for Lemma 1

For the sake of clarity, we provide in Algorithm 2 the procedure that we use for the decomposition into paths of a solution of the continuous relaxation of arc-flow. The algorithm receives in input a generic solution $\bar{x}_{de}$ of $L(F_{AF})$, that is, the linear programming relaxation of model (4.7)–(4.10), in which (4.10) is replaced by $x_{de} \geq 0$. Let P define a set of paths. For short, let p define both a path and the index of such path, for $p \in P$. Algorithm 2 selects an arc emanating from the source node 0 to initialize a path (step 3). Then, it iteratively extends the path until it reaches the sink node c (steps 5–8). The flow on the path is set as the minimum among the flows on the selected arcs (steps 9 and 10). The process is then iterated until all variables take value 0, and the set P is returned.

Algorithm 2 `DecomposeAF`

```
 1: **Input:** an $L(F_{AF})$ solution $\bar{x}_{de}$
 2: $P \leftarrow \emptyset$
 3: **while** $\exists$ an arc $(0, e)$ with $\bar{x}_{0e} > 0$ **do**
 4:     $p \leftarrow \emptyset$; $d \leftarrow 0$
 5:     **while** $d \neq c$ **do**
 6:         select the first arc $(d, e) \in \delta^+(d)$ with $\bar{x}_{de} > 0$ and add it to $p$
 7:         $d \leftarrow e$
 8:     **end while**
 9:     $\bar{z}_p \leftarrow \min_{(d,e) \in p}\{\bar{x}_{de}\}$
10:     **for all** $(d, e) \in p$, $\bar{x}_{de} \leftarrow \bar{x}_{de} - \bar{z}_p$
11:     $P \leftarrow P \cup \{p\}$
12: **end while**
13: **return** $P$
```

Supplementary material 4.B Proof of Lemma 2

Lemma 2. *Any solution of one-cut or of its continuous relaxation can be decomposed into a set of binary trees.*

Proof. The proof is based on the procedure that we use to decompose the solution into trees, which is given in Algorithm 3. The algorithm receives in input a generic solution $\bar{y}_{rq}$ of $L(F_{OC})$ (that is, the linear programming relaxation of model (4.4)–(4.6), in which (4.6) is replaced by $y_{pq} \geq 0$), having objective function value $\bar{z}$. Let T define a set of binary trees. Let t define both a tree and the index of such tree, for $t \in T$. Each tree is formed by a set of leaves, each having at most two children, a left one and a right one. Intuitively, children nodes are created by a cut on the piece (entire bin or residual) corresponding to their parent node. Let us use $[r, q]$ to denote a cut on a piece of length r that produces a left child of size q (and a right child of size $r - q$). Let $\mathcal{C}(\sqcup)$ denote the set of cuts used to generate tree t, for $t \in T$.

Algorithm 3 `DecomposeOC`

```
1:  Input: an $L(F_{OC})$ solution $\bar{y}_{rq}$
2:  $T \leftarrow \emptyset$
3:  while $\exists$ a cut $[c, q]$ with $\bar{y}_{cq} > 0$ do
4:      $t \leftarrow \{c\}$; $\mathcal{L} \leftarrow \{c\}$
5:      while $\exists$ a cut $[r, q]$ with $r \in \mathcal{L}$ and $\bar{y}_{rq} > 0$ do
6:          $\mathcal{L} \leftarrow \mathcal{L} \setminus \{r\}$
7:          add $q$ as left child of $r$ in $t$, and $r - q$ as right child of $r$ in $t$
8:          $\mathcal{L} \leftarrow \mathcal{L} \cup \{q\} \cup \{r - q\}$        ▷ Duplicate entries, if any, are kept
9:          $\mathcal{C}(t) \leftarrow \mathcal{C}(t) \cup \{[r, q]\}$
10:     end while
11:     $\bar{z}_t \leftarrow \min_{[r,q] \in t}\{\bar{y}_{rq}/b_{[r,q]}\}$ ▷ where $b_{[r,q]}$ is the number of times $[r, q]$ appears in $\mathcal{C}(t)$
12:     for all $[r, q] \in t$, $\bar{y}_{rq} \leftarrow \bar{y}_{rq} - \bar{z}_t * b_{[r,q]}$
13:     $T \leftarrow T \cup \{t\}$
14: end while
15: return $T$
```

Algorithm 3 selects a cut $[c, q]$ with positive $\bar{y}_{cq}$ value to initialize a tree t (step 3). Apart from the tree and the corresponding set of cuts $\mathcal{C}(t)$, the algorithm also stores a temporary list of leaves $\mathcal{L}$, which is initialized to $\{c\}$ and updated during the iterations. As long as there is a cut on a piece r that belongs to the list, then the tree and the corresponding set of cuts are enlarged (steps 5–10). The list is updated by removing the piece that was cut and inserting the two children. Note that, at step 8, the two children are always added to the list, even if their sizes are identical or are equal to the size of other entries in the list. The process of enlarging the tree terminates when no cut is found, and then the value $\bar{z}_t$ of the tree is set as the minimum among the values of the variables associated to the cuts of the tree divided by the number of times they appear in the tree. The residual variables values are updated at step 12, and the process continues until all trees have been generated.

To prove that Algorithm 3 effectively decomposes a one-cut solution into binary trees, we show that after updating the variables associated to the cuts of the tree, it is impossible to find:

$$q \in \mathcal{W} \cup \mathcal{R} \setminus \{c\} : \sum_{r \in C(q)} \bar{y}_{qr} > 0, \sum_{p \in A(q)} \bar{y}_{pq} = 0, \sum_{p \in B(q)} \bar{y}_{p+q,p} = 0 \tag{4.46}$$

Indeed, after a tree t has been created and the variables have been updated, for each $q \in \mathcal{W} \cup \mathcal{R} \setminus \{c\}$, there are two possible cases:

- $q \in \mathcal{L}$, or in other words, q belongs (at least once) to the final leafs of the tree (like, e.g., 7, 3, and 1 in the first tree). According to Step 5, this means that for such q, $\nexists \bar{y}_{qr} > 0$, and then, (4.46) does not exist for such q.
- $q \notin \mathcal{L}$, or in other words, q does not belong to any of the final leafs of the tree (like, e.g., 4 in the first tree). This means that in tree t, $\sum_{p \in A(q)} \bar{y}_{pq} + \sum_{p \in B(q)} \bar{y}_{p+q,p} = \sum_{r \in C(q)} \bar{y}_{qr}$, or in other words, that the amount of leafs where q is a child is equal to the amount of leafs where q is a father. By reminding that if inequality (4.5) is initially satisfied, then it is also satisfied even after removing an equal amount from its two sides, and thus, (4.46) does not exist for such q.

□

Supplementary material 4.C Proof of Theorem 1

Theorem 1. *F_{AF} is equivalent to F_{OC}.*

Proof. The sketch of the proof was already included in [96]. We first prove that F_{AF} is included in F_{OC}, and then that F_{OC} in included in F_{AF}.

Arc-flow is included in one-cut.

Let us consider a generic solution $\bar{x}_{de}$ to $L(F_{AF})$, that is, the linear programming relaxation of model (4.7)–(4.10) in which (4.10) is replaced by $x_{de} \geq 0$. According to Lemma 1, $\bar{x}_{de}$ can be decomposed into paths. In the following, we suppose that all paths are *left-aligned*. If that was not the case, one could easy left-align the paths by moving each arc as much as possible to the left, until no arc could be further moved. An easy way to obtain a left-aligned solution is to replace in $\mathcal{A}$ each loss arc $(d, d+1)$ with a new loss arc (d, c). These new loss arcs can only appear at the end of a path, thus imposing a left-alignment. Then, a solution with these alternative loss arcs can be easily mapped into an original $L(F_{AF})$ solution by replacing each selected (d,c) arc with a chain of $(d,d+1)$ arcs.

We use $\bar{x}_{de}$ to build a feasible and same-cost solution $\bar{y}_{pq}$ to $L(F_{OC})$ by using Algorithm 4. The intuition behind the algorithm, is that any item arc (d,e) becomes a cut on a piece of length $(c-d)$ to obtain a left piece of length $(e-d)$.

Algorithm 4 `Transform_AF_into_OC`

```
1: Input: a left aligned solution $\bar{x}_{de}$ of $L(F_{AF})$
2: for all $p \in \mathcal{R}, q \in \mathcal{W}, p > q$ do $\bar{y}_{pq} \leftarrow 0$
3: for all $(d,e) \in \mathcal{A}_{\mathrm{I}} : \bar{x}_{de} > 0, e < c$ do $\bar{y}_{c-d,e-d} \leftarrow \bar{x}_{de}$
4: return $\bar{y}$
```

Algorithm 4 first initializes all y variables to 0, and then modifies them by considering the $\bar{x}$ input values associated with item arcs. To prove the correctness of the procedure, we first show the existence of all the invoked y variables. The existence of the variables invoked at step 2 derives from the definition of the F_{OC} model itself. For step 3, because (d,e) is an item arc we know that (i) $d \in \mathcal{S}$ and $e-d \in \mathcal{W}$, and thus $c-d \in \mathcal{R}$. Consequently, $y_{c-d,e-d}$ is an F_{OC} variable providing that $c-d > e-d$, which is verified when $e < c$.

We now prove that $\bar{y}$ has the same solution cost of $\bar{x}$. By using constraints (4.8) and recalling that no arc enters the source node 0, we know that $\bar{z} = \sum_{(0,f)\in\delta^+(0)} \bar{x}_{0f}$. All paths in $\bar{x}_{de}$ are left-aligned and so they start with one of the m item arcs. Moreover, we can equivalently use w_i for $i = 1,\dots,m$ or $q \in \mathcal{W}$ to state the item width. By using these considerations, in the given order, and then applying Algorithm 4 to $\bar{x}_{0q}$, we get

$$\bar{z} = \sum_{(0,f)\in\delta^+(0)} \bar{x}_{0f} = \sum_{i=1}^{m} \bar{x}_{0w_i} = \sum_{q\in\mathcal{W}} \bar{x}_{0q} = \sum_{q\in\mathcal{W}} \bar{y}_{c-0,e-0} = \sum_{q\in\mathcal{W}} \bar{y}_{cq},$$

which is equivalent to (4.4).

To prove the feasibility of $\bar{y}$, we first note that Algorithm 4 only creates non-negative variable values, so constraints (4.6) are automatically satisfied and we only have to show that constraints (4.5) are satisfied for any $q \in \mathcal{W} \cup \mathcal{R} \setminus \{c\}$. We first focus on the case in which q is a residual width but not an item width, that is, $q \in \mathcal{R} \setminus \mathcal{W} \setminus \{c\}$. This case is easier because it implies $L_q = 0$ and $A(q) = \emptyset$, and so we just need to show that $\sum_{p\in B(q)} \bar{y}_{p+q,p} \geq \sum_{r\in C(q)} \bar{y}_{qr}$. To this aim, we first rewrite the flow conservation at intermediate nodes $e = 1, \dots, c-1$ in (4.8) as

$$\sum_{(d,e)\in\delta^-(e)} \bar{x}_{de} = \sum_{(e,f)\in\delta^+(e)} \bar{x}_{ef} = \sum_{\substack{(e,f)\in\delta^+(e),\\ f\neq c}} \bar{x}_{ef} + \sum_{(e,c)\in\delta^+(e)} \bar{x}_{ec} \geq \sum_{\substack{(e,f)\in\delta^+(e),\\ f\neq c}} \bar{x}_{ef}. \qquad (4.47)$$

After applying Algorithm 4 to the leftmost and rightmost elements, (4.47) can be rewritten by replacing $\bar{x}$ with $\bar{y}$, thus obtaining

$$\sum_{d\in\mathcal{S}, e-d\in\mathcal{W}} \bar{y}_{c-d,e-d} \quad \geq \quad \sum_{e\in\mathcal{S}, f-e\in\mathcal{W}} \bar{y}_{c-e,f-e} \quad e = 1, \dots, c-1.$$

This can be modified, by using the definitions of sets B ad C (see Section 4.2.3), as

$$\sum_{e-d\in B(c-e)} \bar{y}_{c-d,e-d} \quad \geq \quad \sum_{f-e\in C(c-e)} \bar{y}_{c-e,f-e} \quad e = 1, \dots, c-1.$$

Then, by renaming the indices as $c - e = q$, $e - d = p$, and $f - e = r$, we obtain

$$\sum_{p\in B(q)} \bar{y}_{p+q,p} \quad \geq \quad \sum_{r\in C(q)} \bar{y}_{qr} \quad q = 1, \dots, c-1,$$

which proves that (4.5) is satisfied for any $q \in \mathcal{R} \setminus \mathcal{W} \setminus \{c\}$ (as $\mathcal{R} \setminus \mathcal{W} \setminus \{c\}$ is included in $\{1, \dots, c-1\}$).

We now concentrate on the case in which q is an item width, for which we also have to take into account L_q and $A(q)$. We consider again flow conservation constraint (4.8), but this time focus on a vertex $e = c - w_i$. By replacing e with $c - w_i$ in the incoming and outgoing flow, and then splitting the outgoing flow in two parts, we get

$$\sum_{(d,c-w_i)\in\delta^-(c-w_i)} x_{d,c-w_i} = \sum_{(c-w_i,f)\in\delta^+(c-w_i)} x_{c-w_i,f} = \sum_{\substack{(c-w_i,f)\in\delta^+(c-w_i),\\ f\neq c}} x_{c-w_i,f} + \sum_{(c-w_i,c)\in\delta^+(c-w_i)} x_{c-w_i,c},$$

from which we derive

$$\sum_{(c-w_i,c)\in\delta^+(c-w_i)} x_{c-w_i,c} = \sum_{(d,c-w_i)\in\delta^-(c-w_i)} x_{d,c-w_i} - \sum_{\substack{(c-w_i,f)\in\delta^+(c-w_i),\\ f\neq c}} x_{c-w_i,f}. \qquad (4.48)$$

We now consider demand constraints (4.9) and rewrite their left hand side as

$$\sum_{(d,d+w_i)\in\mathcal{A}} x_{d,d+w_i} = \sum_{\substack{(d,d+w_i)\in\mathcal{A},\\ d+w_i\neq c}} \bar{x}_{d,d+w_i} + \sum_{\substack{(d,d+w_i)\in\mathcal{A},\\ d+w_i=c}} \bar{x}_{d,d+w_i} = \sum_{\substack{(d,d+w_i)\in\mathcal{A},\\ d+w_i\neq c}} \bar{x}_{d,d+w_i} + \sum_{(c-w_i,c)\in\delta^+(c-w_i)} x_{c-w_i,c}.$$

We embed (4.48) into this last equation, and then use it to rewrite (4.9) as

$$\sum_{\substack{(d,d+w_i)\in\mathcal{A},\\ d+w_i\neq c}} \bar{x}_{d,d+w_i} + \sum_{(d,c-w_i)\in\delta^-(c-w_i)} \bar{x}_{d,c-w_i} - \sum_{\substack{(c-w_i,f)\in\delta^+(c-w_i),\\ f\neq c}} \bar{x}_{c-w_i,f} \geq d_i. \quad (4.49)$$

We now apply the transformation involved in Algorithm 4 to each member of the left hand side of (4.49), namely: $\sum_{\substack{(d,d+w_i)\in\mathcal{A},\\ d+w_i\neq c}} \bar{x}_{d,d+w_i} = \sum_{e\in\mathcal{S},w_i\in\mathcal{W}} \bar{y}_{c-e,w_i}$; $\sum_{(d,c-w_i)\in\delta^-(c-w_i)} \bar{x}_{d,c-w_i} = \sum_{d\in\mathcal{S},c-d-w_i\in\mathcal{W}} \bar{y}_{c-d,c-w_i-d}$; and $\sum_{\substack{(c-w_i,f)\in\delta^+(c-w_i),\\ f\neq c}} \bar{x}_{c-w_i,f} = \sum_{c-w_i\in\mathcal{S},f-(c-w_i)\in\mathcal{W}} \bar{y}_{w_i,f-(c-w_i)}$. We can thus rewrite (4.49) as

$$\sum_{e\in\mathcal{S},w_i\in\mathcal{W}} \bar{y}_{c-e,w_i} + \sum_{d\in\mathcal{S},c-d-w_i\in\mathcal{W}} \bar{y}_{c-d,c-w_i-d} - \sum_{c-w_i\in\mathcal{S},f-(c-w_i)\in\mathcal{W}} \bar{y}_{w_i,f-(c-w_i)} \geq d_i. \quad (4.50)$$

We now use the definitions of sets A, B, and C, to rewrite the three components of the left hand side of (4.50), replace d_i with L_{w_i} in the right hand side, obtaining

$$\sum_{c-e\in A(w_i)} \bar{y}_{c-e,w_i} + \sum_{c-w_i-d\in B(w_i)} \bar{y}_{c-d,c-w_i-d} - \sum_{f-(c-w_i)\in C(w_i)} \bar{y}_{w_i,f-(c-w_i)} \geq L_{w_i}.$$

We use index $q \in \mathcal{W}$ instead of w_i for $i = 1, \ldots, m$ and rewrite the variables indices, so we get

$$\sum_{p\in A(q)} \bar{y}_{pq} + \sum_{p\in B(q)} \bar{y}_{p+q,p} - \sum_{r\in C(q)} \bar{y}_{qr} \geq L_q$$

which proves that (4.5) is also feasible for any $q \in \mathcal{W}$ and concludes the first part of the proof.

One-cut is included in arc-flow.

Let us consider a solution $\bar{y}_{pq}$ of $L(F_{OC})$. From Lemma 2 we know that $\bar{y}_{pq}$ can be decomposed in a set of trees. In the following, we consider the stronger fact that $\bar{y}_{pq} > 0$ can be decomposed into a set of *right-sided* trees, i.e., trees in which only right children are allowed to have their own children. This requirement is equivalent to the one that we discussed at the beginning of the first part of the proof (arc-flow solution is left aligned), and can also be made without loss of generality. Indeed, the decomposition into right-sided trees can be obtained by either (i) modifying the positions of the cuts, or (ii) enforcing an additional constraint in the model and applying a modification to Algorithm 3. Here we describe the second option, whose implementation is simpler. It consists of first adding to model (4.4)–(4.6) the inequality

$$\sum_{r \in C(q)} y_{qr} \leq \sum_{p \in B(q)} y_{p+q,p} \quad q \in \mathcal{W} \cup \mathcal{R} \setminus \{c\} \tag{4.51}$$

and then transforming step 8 of Algorithm 3 from $\mathcal{L} \leftarrow \mathcal{L} \cup \{q\} \cup \{r-q\}$ in $\mathcal{L} \leftarrow \mathcal{L} \cup \{r-q\}$. Inequality (4.51) imposes that the number of times q is recut (left-hand side) is not greater than the number of times it appears as a right child (right-hand side), thus ensuring that all recuts may be performed on a right child. Then, the modification in Algorithm 3 guarantees that no left child enters the candidate list for recutting, thus producing only right-sided trees.

We use $\bar{y}_{pq}$ to build a feasible and same-cost solution $\bar{x}_{de}$ to $L(F_{AF})$ by using Algorithm 5. The algorithm considers the alternative loss arcs (d, c) that we discussed at the beginning of the first part of the proof to have a left aligned arc-flow solution. Recall that these can be easily mapped into chains of unit width loss arcs $(d, d+1)$ if needed. The intuition behind algorithm 5 is that each cut becomes either a path that increases flow on two arcs or a cycle that decreases flow on one arc and increases it on two other arcs (two examples are provided right at the end of the proof to clarify this aspect).

Algorithm 5 `Transform_OC_into_AF`

```
1: Input: a solution $\bar{y}_{pq}$ of $L(F_{OC})$ + (4.51)
2: for all $(d, e) \in \mathcal{A}'$ do $\bar{x}_{de} \leftarrow 0$
3: for all $p \in \mathcal{R}, q \in \mathcal{W}, p > q : \bar{y}_{pq} > 0$ do
4:     $\bar{x}_{c-p,c-p+q} \leftarrow \bar{x}_{c-p,c-p+q} + \bar{y}_{pq}$
5:     $\bar{x}_{c-p+q,c} \leftarrow \bar{x}_{c-p+q,c} + \bar{y}_{pq}$
6:     if $p \neq c$ then $\bar{x}_{c-p,c} \leftarrow \bar{x}_{c-p,c} - \bar{y}_{pq}$ end-if
7: end-for
8: return $\bar{x}$
```

The algorithm initializes all $\bar{x}$ values to 0, and then increases (steps 4 and 5) or decreases (step 6) their values on the basis of the cuts selected in the solution of $L(F_{OC})$ + (4.51).

Step 4 of Algorithm 5 is invoked at most once for each pair of p and q values, thus leading to

$$\bar{x}_{c-p,c-p+q} \quad = \quad \bar{y}_{pq} \quad p \in \mathcal{R}, q \in \mathcal{W}, p > q. \tag{4.52}$$

Steps 5 and 6 may instead modify multiple times the same variable $x_{c-q,c}$ for a given value of q. By considering all generic cuts $[r,s]$ for which $r - s = q$, step 5 leads to $\bar{x}_{c-q,c} \leftarrow \sum_{r\in\mathcal{R},s\in\mathcal{W},r>s:r-s=q} \bar{y}_{rs} = \sum_{s+q\in\mathcal{R},s\in\mathcal{W}} \bar{y}_{s+q,s}$. By considering all generic cuts $[r,s]$ for which $r = q$, step 6 leads instead to $\bar{x}_{c-q,c} \leftarrow -\sum_{r\in\mathcal{W},s\in\mathcal{R},r>s:r=q} \bar{y}_{rs} = \sum_{q\in\mathcal{W},s\in\mathcal{R},q>s} \bar{y}_{qs}$. By merging these two components and recalling that $B(q) = \{p \in \mathcal{W} : p + q \in \mathcal{R}\}$ and $C(q) = \{p \in \mathcal{W} : p < q\}$ we obtain

$$\bar{x}_{c-q,c} \quad = \quad \sum_{p\in B(q)} \bar{y}_{p+q,p} - \sum_{r\in C(q)} \bar{y}_{qr} \quad q \in \mathcal{R} \setminus \{c\} \tag{4.53}$$

Equations (4.52) and (4.53) are at the basis of the correctness of Algorithm 5, that we are going to prove. First of all, we demonstrate the existence of all the x variables invoked by the algorithm by showing that they are all associated with arcs belonging to set $\mathcal{A}$. The existence of the variables at step 2 derives from the definition of the F_{AF} model itself. For step 4, because $[p,q]$ is a cut, we know that $c - p \in \mathcal{S}$, $q \in \mathcal{W}$, and $q < p$. Consequently $c - p + q \in \mathcal{S}$ and $c - p + q < c$, which certifies that $(c - p, c - p + q) \in \mathcal{A}$. A similar reasoning is used for steps 5 and 6: because $[p,q]$ is a cut, then both $c - p$ and $c - p + q$ belong to $\mathcal{S}$, and thus arcs $(c - p, c)$ and $(c - p + q, c)$ belong to $\mathcal{A}$, either in form of items arcs or in form of the long loss arcs that we introduced at the beginning of the proof.

To show that $\bar{x}$ has the same cost of $\bar{y}$, it is enough to apply (4.52) to (4.4), obtaining

$$z = \sum_{q\in\mathcal{W}} \bar{y}_{cq} = \sum_{q\in\mathcal{W}} \bar{x}_{c-c,c-c+q} = \sum_{q\in\mathcal{W}} \bar{x}_{0,q} = \sum_{(0,e)\in\delta^+(0)} \bar{x}_{0,e}$$

which is equivalent to (4.7).

We now focus on the constraints. We first show that constraints (4.8) remain satisfied during any iteration of Algorithm 5. This can be intuitively noticed by the example that we present, for the sake of clarity, right after the proof (see Figure 4.9). Formally, constraints (4.8) are clearly satisfied after the initialization step as all flows take value 0. Then, let us first consider the flow variation involved by $\bar{y}_{pq} > 0, p \neq c$. At step 4, the amount of flow leaving $c-p$ is increased by $\bar{y}_{pq}$ but is reduced by the same amount at step 6 resulting in no flow variation. The same observation can be made for the flow entering c at steps 5 and 6. For the flows entering and leaving $c-p+q$, they are increased by the same amount at steps 4 and 5. Now, let us consider the flow variations involved by $\bar{y}_{cq} > 0$. Again, the flows entering and leaving q are increased by the same amount at steps 4 and 5. At step 4, the amount of flow leaving 0 is increased by $\bar{y}_{cq}$, the same amount added to the flow entering c at step 5. Consequently, constraints (4.8) are satisfied by the built $\bar{x}$ values.

To prove that demand constraints (4.9) are satisfied, we start by rewriting (4.5) for $q \in \mathcal{W}$ as

$$\sum_{p \in A(q)} \bar{y}_{pq} + \sum_{p \in B(q)} \bar{y}_{p+q,p} - \sum_{r \in C(q)} \bar{y}_{qr} \geq L_q \quad q \in \mathcal{W},$$

which, by applying (4.52) and (4.53), can be modified as

$$\sum_{\substack{(c-p,c-p+q) \in \mathcal{A}', \\ c-p+q \neq c}} \bar{x}_{c-p,c-p+q} + \bar{x}_{c-q,c} \geq L_q \quad q \in \mathcal{W}.$$

We now set $w_i = q$ and $e = c-p$, replace L_{w_i} widt d_i, and obtain

$$\sum_{\substack{(e,e+w_i) \in \mathcal{A}, \\ e+w_i \neq c}} \bar{x}_{e,e+w_i} + \bar{x}_{c-w_i,c} = \sum_{(e,e+w_i) \in \mathcal{A}} \bar{x}_{e,e+w_i} \quad \geq \quad d_i \quad i = 1, \ldots, m.$$

For what concerns non-negativity, this may be an issue only for variables $x_{c-q,c}$, as they are the only ones affected by step 6 of Algorithm 5. We can notice, however, that the right-hand side of (4.53) is forced to be non-negative by (4.51), thus imposing $\bar{x}_{c-q,c} \geq 0$ (an example of a solution that does not satisfy (4.51) and leads to a negative $\bar{x}$ value is shown below). This shows that $\bar{x}$ is a feasible $L(F_{AF})$ solution and thus concludes the proof.

$\square$

For the sake of clarity, we conclude this section by two examples that might be useful for understanding the details of the proof. In Figure 4.9 we graphically depict the four steps performed by Algorithm 5 when applied to the $\bar{y}_{pq}$ solution of Example 2 discussed in Section 4.3. The solution consists of four cuts ($\bar{y}_{11,7} = 1$, $\bar{y}_{11,4} = 1/3$, $\bar{y}_{7,3} = 1/3$, and $\bar{y}_{4,3}$

$= 2/3$) and thus requires four iterations, shown in the figure from the top to the bottom. Suppose the algorithm selects the cuts at step 3 in the order in which we gave them. At iteration 1, it selects $[11, 7]$ and sets $\bar{x}_{0,7} = \bar{x}_{7,11} = 1$, obtaining the path shown in the top-most part of the figure. At iteration 2, it selects $[11, 4]$, thus setting $\bar{x}_{0,4} = \bar{x}_{4,11} = 1/3$ and creating a second path. At iteration 3 it processes $[7, 3]$, and, because $p = 7 \neq c$, it sets $\bar{x}_{4,7} = 1/3$, increases $\bar{x}_{7,11}$ to $4/3$, and decreases $\bar{x}_{4,11}$ to 0, thus moving the precedent flow on (4,11) to (4,7) and (7,11). At the last iteration, it selects $[4, 3]$, thus imposing $\bar{x}_{7,10} = \bar{x}_{10,11} = 2/3$ and $\bar{x}_{7,11} = 2/3$, moving part of the flow on (7,11) to (7,10) and (10,11).

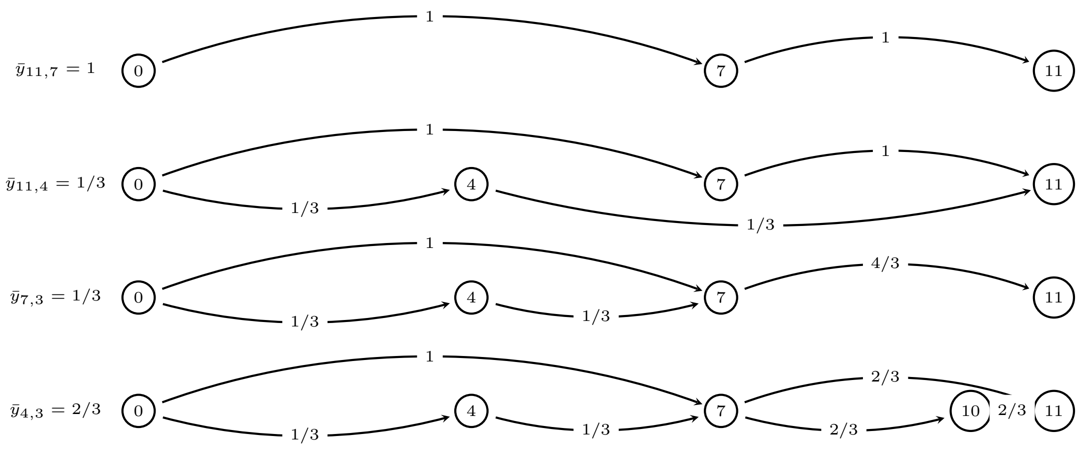


Figure 4.9: Construction of an $L(F_{AF})$ solution for Example 2 through Algorithm 5. Each iteration, from top to bottom, processes the cut associated with the $\bar{y}$ variable given on the left.

The second example that we present shows that, without additional constraints (4.51), negative flows could appear in the solution produced by Algorithm 5. Indeed, let us consider an optimal solution of the relaxed version of $L(F_{OC})$ for Example 2, having value 4/3 and consisting of $\bar{y}_{11,7} = 4/3$, $\bar{y}_{7,3} = 1/3$, and $\bar{y}_{4,3} = 2/3$. The three iterations, one per cut, produced by Algorithm 5 are graphically depicted in Figure 4.10. It can be noticed, at the last iteration, that $\bar{x}_{4,11} = -1/3$. Intuitively, this corresponds to a cycle in the flow decomposition.

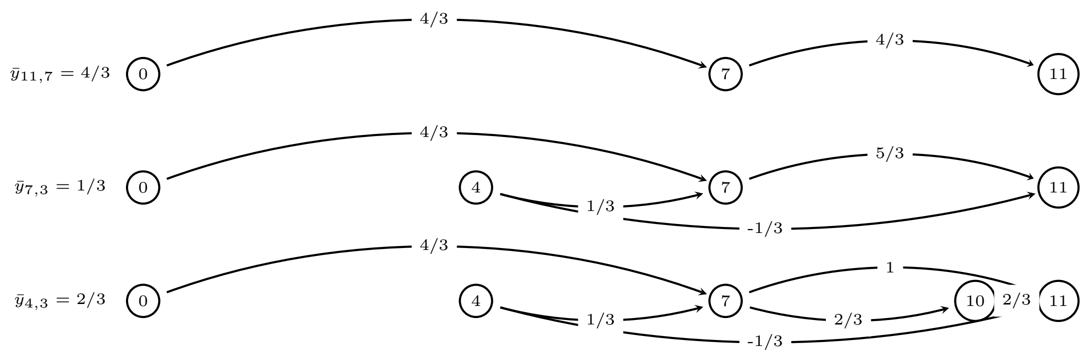


Figure 4.10: An invalid $L(F_{AF})$ solution of Example 2 with a negative flow (cycle), obtained by executing Algorithm 5 on an input $L(F_{OC})$ solution that does not satisfy (4.51).

Supplementary material 4.D Proof of Theorem 2

Theorem 2. *F_{DP} dominates F_{AF} (and hence F_{OC}).*

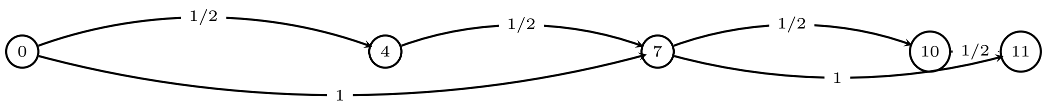
Proof. We first prove that F_{DP} is included in F_{AF}, and then give an example that shows that F_{AF} is not included in F_{DP}. For the first part, we make use of Algorithm 6. The algorithm takes in input an optimal solution $\bar{\varphi}_{j,d,j+1,e}$ of $L(F_{DP})$ and uses it to create a feasible solution $\bar{x}_{de}$ of $L(F_{DP})$. The idea, already noted in [98], is to vertically "shrink" all states with the same partial bin filling into a single node, and merge all arcs entering (resp. emanating) from the node into a unique entering (resp. emanating) arc. Consider again Example 2 of Section 4.2.2 and the optimal $L(F_{DP})$ solution depicted in Figure 4.3. By applying Algorithm 6 to it, we obtain the feasible $L(F_{AF})$ solution shown in Figure 4.11-(a).

Formally, (i) the variables associated to item arcs in F_{AF} are obtained in step 2 and become equal to the sum of the values of the merged variables from F_{DP}; (ii) the variables associated to the loss arcs $(d, d+1)$ in F_{AF} are derived in step 3 by mapping the final arcs of F_{DP} (which, we recall, connect any node (n, d) with the dummy node $(n+1, c)$). The resulting $\bar{x}_{de}$ values satisfy constraints (4.8) because $\bar{\varphi}_{j,d,j+1,e}$ satisfy (4.12), and also constraints (4.9) because $\bar{\varphi}_{j,d,j+1,e}$ satisfy (4.13). This proves that F_{DP} is included in F_{AF}.

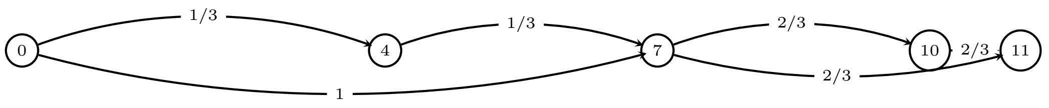
Algorithm 6 `Transform_DP_into_AF`

```
1: Input: a solution $\bar{\varphi}_{j,d,j+1,e}$ of $L(F_{DP})$
2: for all $(d,e) \in \mathcal{A}_\mathrm{I}$ do $\bar{x}_{de} \leftarrow \sum_{j=1}^{n} \bar{\varphi}_{j,d,j+1,e}$
3: for all $(d,d+1) \in \mathcal{A}_\ell$ do $\bar{x}_{d,d+1} \leftarrow \sum_{e=1}^{d} \bar{\varphi}_{n,e,n+1,c}$
4: return $\bar{x}$
```

To show instead that F_{AF} is not included in F_{DP}, it is enough to consider the same example. The optimal $L(F_{AF})$ solution, previously discussed in Section 4.3 has value 4/3. For the sake of clearness, this solution is reported in Figure 4.11-(b). The reason of this AF misbehavior is that the partial filling of value 7 can be created by both the use of item 7 and the combined used of items 4 and 3. Consequently, the path (0,4,7,10,11) cannot be obtained in F_{DP} but can be produced easily by F_{AF}.



(a) Feasible $L(F_{AF})$ solution of value 3/2 obtained by Algorithm 6 on an optimal F_{DP} solution



(b) Optimal $L(F_{AF})$ solution of value 4/3

Figure 4.11: Example 2 shows that F_{AF} is not included in F_{DP}.

□

Supplementary material 4.E Proof of Theorem 3

THEOREM 3. *F_{DP} is equivalent to F_{PR}.*

Proof. We first prove that F_{PR} is included in F_{DP}, and then that F_{DP} in included in F_{PR}. For the first part, we use the same idea than for Lemma 1 to decompose a solution of $L(F_{DP})$ into a set of paths. Then, we show that for each path in F_{DP}, there is a pattern in F_{PR}. The idea, this time, is to horizontally shrink each path p to obtain a pattern p', as shown in Figure 4.12. At each vertical stage of p corresponds a binary variable of p'. If at stage j, the arc is vertical (i.e., $\varphi_{j,d,j+1,d} = \bar{z}_p$), then the j^{th} binary variable of p' takes value 0. If instead, the arc is diagonal (i.e., $\varphi_{j,d,j+1,d+w_j} = \bar{z}_p$), then the j^{th} binary variable of p' takes value 1.

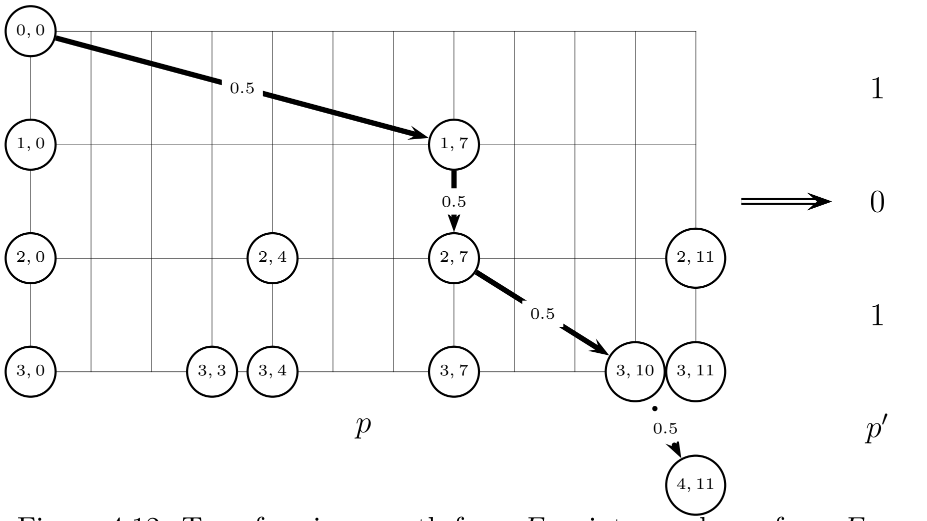


Figure 4.12: Transforming a path from F_{DP} into a column from F_{PR}

As F_{DP} is dedicated to the BPP, the transformation produces binary columns. It is possible to recreate afterwards integer columns by summing the binary variables associated with the same item width. If the demand constraints (4.13) are satisfied in the solution of $L(F_{DP})$, then they are also satisfied in the transformed patterns of $L(F_{PR})$.

The reverse process is easy: a pattern for the CSP from $L(F_{PR})$ can be first transformed into a binary pattern for the BPP, and then mapped into a path. By remarking that any path created in this way respects flow conservation constraints (4.12), and that if demand constraints are satisfied in the solution of $L(F_{PR})$ then constraints (4.13) are also satisfied, we obtain that any $L(F_{PR})$ solution can be transformed into an F_{DP} one. Additionally remark that, to strictly respect the " $=$ " sign of constraints (4.13), one should put an " $=$ " sign in the item constraints of F_{PR} as well, otherwise, the transformation may produce some surplus items. If one instead wants to keep the " $\geq$ " sign, then surplus items should be detected and removed from the columns before creating the paths. This concludes the proof.

□

Supplementary material 4.F Proof of Theorem 4

THEOREM 4. *F_{RE} models the CSP.*

Proof. Take any feasible pattern p used in a given CSP solution, order the items it contains in non increasing weight, and put them into the subset I_{c1} until the sum of the weights in the subset is greater or equal than $c/2$ or until there is no more items. The rest of the items, if any, is put in I_{c2}. By creating the arcs with Algorithm 7, we know that there exists a path p_1 that goes from 0 to $\min(c/2, c - \sum_{i \in I_{c1}} w_i)$ that takes all the items of I_{c1} whose last arc is a reflected arc (if $\sum_{i \in I_{c1}} w_i \leq c/2$, then complete the path with loss arcs until reaching $(c/2, c/2), r)$. We also know that there exists a path p_2 that goes from 0 to $\sum_{j \in I_{c2}} w_j$ and that takes all the items of I_{c2}. In addition, we know that some loss arcs link every active node to the following one, and thus, that there exists p_3, a set of loss arcs that goes from $\sum_{j \in I_{c2}} w_j$ to $\min(c/2, c - \sum_{i \in I_{c1}} w_i)$ ($p_3 = \emptyset$ if the two values are the same). Thus, there exists two colliding paths p_1 and $p_2 \cup p_3$ colliding in $\min(c/2, c - \sum_{i \in I_{c1}} w_i)$ that represent pattern p. □

Supplementary material 4.G Algorithms for reflect

In Algorithm 7, we detail how to build the set of arcs $\mathcal{A}_s$ and $\mathcal{A}_r$ used by F_{RE}. We keep in $\mathcal{V}$ the set of activated nodes that are used to build the loss arcs. We use the array M to keep track of the possible tails for the arcs. For a given item type, we also use the array H to keep track of the possible tails that were already processed, to avoid unnecessary operations. Then, for each item type i and for each demand of item i, we go through all the possible tails k that were not already processed (step 9), and create the arc $(l, l + w_j)$. If it is a standard arc (step 11), $(l, l + w_j, s)$ is stored in $\mathcal{A}_s$ (step 13) and $l + w_j$ becomes a possible tail (step 12) and an activated node (step 14). If it is a reflected arc not removed by the reduction procedure (step 15), the arc is transformed into $(l, c - (l + w_i), r)$ and is stored in $\mathcal{A}_r$ (step 16) and node $c - (l + w_i)$ is only added to the set of activated nodes (step 17). Finally, we add node $c/2$ to $\mathcal{V}$ (step 23) and create a loss arc between each pair of successive active nodes (step 24) and terminate by adding the final arc $(c/2, c/2, r)$ in $\mathcal{A}_r$ (step 26).

Algorithm 7 `Create_reflect_multigraph`

```
1: Input: $c$: bin capacity, $m$: number of item types, $w$: item widths, $d$: item demands
2: $\mathcal{A}_s \leftarrow \emptyset$ ; $\mathcal{A}_r \leftarrow \emptyset$ ; $\mathcal{V} \leftarrow \emptyset$
3: $M[0 \dots c/2] \leftarrow 0$        ▷ an array that keeps track of the possible tails
4: $M[0] \leftarrow 1$        ▷ node 0 is a possible tail
5: for $i = 1$ to $m$ do
6:     $H[0 \dots c/2] \leftarrow 0$        ▷ an array that keeps track of the tails already processed
7:     for $j = 1$ to $d_i$ do
8:         for $l = c/2 - 1$ to $0$ do
9:             if $H[l] = 0$ and $M[l] = 1$ then        ▷ if $l$ is a not yet processed tail
10:                $H[l] = 1$        ▷ $l$ is now processed
11:                if $l + w_i \leq c/2$ then        ▷ if the arc is standard
12:                    $M[l + w_i] \leftarrow 1$
13:                    $\mathcal{A}_s \leftarrow \mathcal{A}_s \cup (l, l + w_i, s)$
14:                    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(l + w_i)\}$
15:                else if $l \leq c - (l + w_i)$ then        ▷ if reflected arc + reduction
16:                    $\mathcal{A}_r \leftarrow \mathcal{A}_r \cup (l, c - (l + w_i), r)$
17:                    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(c - (l + w_i))\}$
18:                end if
19:            end if
20:        end for
21:    end for
22: end for
23: $\mathcal{V} \leftarrow \mathcal{V} \cup \{c/2\}$
24: for $i \in \mathcal{V}$ do $\mathcal{A}_s \leftarrow \mathcal{A}_s \cup \{(i, j, s)\colon\ j = \min(l \in \mathcal{V} : l > i)\}$        ▷ loss arcs
25: $\mathcal{A}_r \leftarrow \mathcal{A}_r \cup (c/2, c/2, r)$        ▷ allow paths that collide in $c/2$
26: return $\mathcal{V}, \mathcal{A}_s, \mathcal{A}_r$
```

Note that the algorithm works for even values of c. In case c is an odd number, an easy adaptation is to double the capacity and the weight of all item types. (This operation does not affect the number of normal patterns, and thus does not increase the number of variables and constraints in F_{RE}). Otherwise, an additional dummy node R has to be created and taken into explicit consideration in the algorithm.

In Algorithm 8, we detail how to reconstruct a solution after solving F_{RE}. We start by creating a path from 0 to its collapsing node d (steps 5-12) and store it in $R[d]$ if it contains a reflected arc, or in $S[d]$ otherwise (step 13). The process is iterated until no more path is found. Finally, for each colliding node d (step 15), we match the paths from $R[d]$ and $S[d]$ to reconstruct the pairs of colliding paths. Such matching is ensured by constraints (4.17) that guarantee that each selected reflected path can be matched with a selected standard path. Finally, we precise that such matching exists if all item types have $w_i < c$. In the opposite case, there could be a selected reflected path $\{(0, 0, r)\}$ corresponding to an item of size c that could not be matched with any standard path as it would represent a bin on its own.

Algorithm 8 `Reconstruct_reflect_solution`

```
1: Input: a solution $\bar{\xi}_{de\kappa}$ of $L(F_{RE})$
2: $P \leftarrow \emptyset$                                              ▷ set of colliding paths
3: $R[0 \dots /2] \leftarrow \emptyset$                                  ▷ an array of sets of reflected paths
4: $S[0 \dots /2] \leftarrow \emptyset$                                  ▷ an array of sets of non-reflected paths
5: while $\exists$ an arc $(0, e, \kappa)$ with $\bar{\xi}_{0e\kappa} > 0$ do ▷ build the paths and store them in $R$ and $S$
6:     $p \leftarrow \emptyset$; $d \leftarrow 0$
7:     while $\exists$ an arc $(d, e, \kappa)$ and $\nexists (d, e, r) \in p$ do
8:         select the first arc $(d, e, \kappa) \in \delta^+(d)$ with $\bar{x}_{de\kappa} > 0$ and add it to $p$
9:         $d \leftarrow e$
10:    end while
11:    $\bar{z}_p \leftarrow \min_{(d,e,\kappa) \in p} \{\bar{x}_{de\kappa}\}$
12:    for all $(d, e, \kappa) \in p$, $\bar{x}_{de\kappa} \leftarrow \bar{x}_{de\kappa} - \bar{z}_p$
13:    if $\exists (d, e, r) \in p$ then $R[d] \leftarrow R[d] \cup \{p\}$ else $S[d] \leftarrow S[d] \cup \{p\}$
14: end do
15: for d = 1 to c/2 do                ▷ match the pairs of colliding paths and store them in $P$
16:    while $\exists p_1 \in R[d]$ with $\bar{z}_{p1} > 0$ do
17:        $p \leftarrow \emptyset$;
18:        select the first path $p_2 \in S[d]$ with $\bar{z}_{p2} > 0$
19:        $p \leftarrow p_1 \cup p_2$
20:        $\bar{z}_p \leftarrow \min(\bar{z}_{p1}, \bar{z}_{p2})$
21:        $\bar{z}_{p1} \leftarrow \bar{z}_{p1} - \bar{z}_p$; $\bar{z}_{p2} \leftarrow \bar{z}_{p2} - \bar{z}_p$
22:        $P \leftarrow P \cup p$
23:    end do
24: end do
25: return $P$
```

In Algorithm 9, we detail how to build the set of arcs $\mathcal{A}_s$ and $\mathcal{A}_1, \dots, \mathcal{A}_K$ used by F_{RE} for the VSBPP. We keep in $\mathcal{V}$ the set of activated nodes that are used to build the loss arcs. We use the array M to keep track of the possible tails for the arcs. For a given item type, we also use the array H to keep track of the possible tails that were already processed, to avoid unnecessary operations. Then, for each item type i and for each demand of item i, we go through all the possible tails k that were not already processed (step 9), and create the arc $(l, l+w_j)$. If it is a standard arc (step 11), $(l, l+w_j, s)$ is stored in $\mathcal{A}_s$ (step 13) and $l+w_j$ becomes a possible tail (step 12) and an activated node (step 14). If it is possible to transform the arc into a valid reflected arc for bin $k (k = 1, \dots, K)$, a copy of the arc is transformed into $(l, c_k - (l+w_i), k)$ and is stored in $\mathcal{A}_k$ (step 18) and node $c_k - (l+w_i)$ is added to the set of activated nodes (step 19). Finally, we add all nodes $c_k/2 (k = 1, \dots, K)$ to $\mathcal{V}$ (step 27) and create a loss arc between each pair of successive active nodes (step 28) and terminate by adding the final arcs $(c_k/2, c_k/2, k)$ in $\mathcal{A}_k (k = 1, \dots, K)$ (step 29).

Algorithm 9 `Create_reflect_multigraph_VSBPP`

```
1: Input: $c_k$: bin capacities, $m$: number of item types, $w$: item widths, $d$: item demands
2: $\mathcal{A}_s \leftarrow \emptyset$ ; for $k = 1$ to $K$ do $\mathcal{A}_k \leftarrow \emptyset$ ; $\mathcal{V} \leftarrow \emptyset$; $c \leftarrow \max_{k=1,\dots,K}\{c_k\}$
3: $M[0 \dots c/2] \leftarrow 0$                    ▷ an array that keeps track of the possible tails
4: $M[0] \leftarrow 1$                              ▷ node 0 is a possible tail
5: for $i = 1$ to $m$ do
6:     $H[0 \dots c/2] \leftarrow 0$               ▷ an array that keeps track of the tails already processed
7:     for $j = 1$ to $d_i$ do
8:         for $l = c/2 - 1$ to $0$ do
9:             if $H[l] = 0$ and $M[l] = 1$ then     ▷ if $l$ is a not yet processed tail
10:                $H[l] = 1$                        ▷ $l$ is now processed
11:                if $l + w_i \le c/2$ then          ▷ if the arc is standard
12:                    $M[l + w_i] \leftarrow 1$
13:                    $\mathcal{A}_s \leftarrow \mathcal{A}_s \cup (l, l + w_i, s)$
14:                    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(l + w_i)\}$
15:                else
16:                    for $k = 1$ to $K$ do          ▷ for each bin type
17:                        if $l + w_i > c_k$ and $l \le c_k - (l + w_i)$ then   ▷ reflection in $k$
18:                            $\mathcal{A}_k \leftarrow \mathcal{A}_k \cup (l, c_k - (l + w_i), k)$
19:                            $\mathcal{V} \leftarrow \mathcal{V} \cup \{(c_k - (l + w_i))\}$
20:                        end if
21:                    end for
22:                end if
23:            end if
24:        end for
25:    end for
26: end for
27: for $k = 1$ to $K$ do $\mathcal{V} \leftarrow \mathcal{V} \cup \{c_k/2\}$
28: for $i \in \mathcal{V}$ do $\mathcal{A}_s \leftarrow \mathcal{A}_s \cup \{(i, j, s)$: $j = \min(l \in \mathcal{V} : l > i)\}$   ▷ loss arcs
29: for $k = 1$ to $K$ do $\mathcal{A}_k \leftarrow \mathcal{A}_k \cup (c_k/2, c_k/2, k)$   ▷ allow paths that collide in $c_k/2$
30: return $\mathcal{V}, \mathcal{A}_1, \dots, \mathcal{A}_k$, $\mathcal{A}_r$
```

Supplementary material 4.H Proof of Theorem 5

THEOREM 5. *Each valid pattern p can be represented in F_{RE} by two colliding paths whose reflected arc (d, e) has $d \leq e$.*

Proof. The proof is very similar to the previous one, and the only change comes when building the sets I_{c1} and I_{c2}. Before putting an item j in I_{c1} that would close the set, we check if $w_j/2 + \sum_{i \in I_{c1}} w_i \leq c/2$. If yes, then the reflected arc (d, e) has $d \leq e$. Otherwise, we put j in I_{c2} and repeat the process until one item closes I_{c1} with the desired reflected arc. If I_{c1} has not been closed while reaching the last item, then I_{c2} becomes the set with the reflected path, and it has the correct form. Indeed, if it was not the case, we would have (i) $w_j/2 + \sum_{i \in I_{c1}} w_i > c/2$ and (ii) $w_j/2 + \sum_{i \in I_{c2}} w_i > c/2$. By summing (i) and (ii), we would obtain $w_j + \sum_{i \in p - \{j\}} w_i > c$, which is impossible as p is a valid pattern. $\square$

Chapter 5

Logic Based Benders' Decomposition for Orthogonal Stock Cutting Problems

[1]

We consider in this chapter the problem of packing a set of rectangular items into a strip of fixed width, without overlapping, using minimum height. Items must be packed with their edges parallel to those of the strip, but rotation by 90° is allowed. The problem is usually solved through branch-and-bound algorithms. We propose an alternative method, based on Benders' decomposition. The master problem is solved through a new ILP model based on the arc flow formulation, while constraint programming is used to solve the slave problem. The resulting method is hybridized with a state-of-the-art branch-and-bound algorithm. Computational experiments on classical benchmarks from the literature show the effectiveness of the proposed approach. We additionally show that the algorithm can be successfully used to solve relevant related problems, like rectangle packing and pallet loading.

Keywords: Orthogonal stock cutting problem, Logic based Benders' decomposition, Rectangle packing, Pallet loading.

5.1 Introduction

Given n rectangular *items* of integer width w_j and height h_j ($j = 1, \ldots, n$) and a *strip* of fixed width W, the *orthogonal Stock Cutting Problem* (SCP) consists in packing all the

[1]The results of this chapter appears in: M. Delorme, M. Iori, and S. Martello, Logic Based Benders' Decomposition for Orthogonal Stock Cutting Problems, *Computers & Operations Research*, 78:290-298, 2017 [100].

items into the strip without overlapping and using the minimum strip height. Items must be packed with their edges parallel to those of the strip, but rotation by 90° is allowed. The *oriented* counterpart of the SCP, in which rotation is not allowed, is known in the literature as the *Strip Packing Problem* (SPP, see, e.g., Martello et al. [212]). Both the SCP and the SPP are important because they have many real world applications, especially in wood, paper, glass, and metal industries (see, e.g., Lodi et al. [193]).

It is not difficult to see that the SCP is $\mathcal{NP}$-hard in the strong sense. Consider indeed the famous (one-dimensional) *bin packing problem* (BPP): partition n elements, having values v_j $(j = 1, \ldots, n)$ into the minimum number of subsets so that the sum of the values in each subset does not exceed a given capacity V (see Delorme et al. [98] for a recent exhaustive survey). Any BPP instance can be transformed into an equivalent SCP instance by setting $W = V$ and, for $j = 1, \ldots, n$, $w_j = v_j$ and $h_j = W + 1$. Its solution will consist of the minimum number of "shelves" of height $W + 1$, and hence it will provide the optimal solution to the BPP instance. The BPP is known to be $\mathcal{NP}$-hard in the strong sense, so the same holds for the SCP.

According to the classification introduced by Lodi et al. [196], the SCP and the SPP can be characterized as 2SP|R|F and 2SP|O|F, respectively, while in the general cutting and packing typology by Wäscher et al. [289] both problems are categorized as *two-dimensional open dimension problem*.

In this chapter we present an exact algorithm for the SCP. In Section 5.2 we review successful approaches that have been proposed in the literature for the SCP, the SPP, and related problems. In Section 5.3 we provide a mathematical model for the SCP. Preprocessing techniques and initial lower and upper bound computations are discussed in Section 5.4. The proposed algorithm is given in Sections 5.5, 5.6, 5.7, and 5.8. The outcome of extensive computational experiments is presented in Section 5.9, where we also discuss some relevant variants of the problem.

5.2 Literature review

Most of the techniques developed in the literature to solve the SCP and the SPP are combinatorial branch-and-bound algorithms. One of the first branch-and-bound approaches for the SPP was proposed by Martello et al. [212] in the early noughties. The algorithm makes use of preprocessing techniques, dominance criteria, and a powerful lower bound based on a problem oriented continuous relaxation. In the same period, Lesh et al. [185] proposed a branch-and-bound algorithm specifically tailored for *perfect-packing* cases, in which the optimal solution has no loss space, i.e., $\sum_{j=1}^{n} w_j h_j = WH$, where H denotes the height of the optimal solution. The algorithm is based on powerful cuts that fathom nodes for which the current partial packing cannot be filled by the remaining items without inserting a 'hole'. Later on, Alvarez-Valdes et al. [7], as well as Boschetti and Montaletti [43] obtained better branch-and-bound algorithms by improving the dominance criteria and the bounding techniques proposed in [212].

In recent years, new types of exact methods were proposed by Westerlund et al. [292] and by Castro and Oliveira [58], who tried *Mixed Integer Linear Programming* (MILP) models to solve more general classes of problems, that include the SPP. The latter also made computational experiments on SPP instances, but the results were not encouraging, as the proposed approaches generally perform worse than the 'old' Martello et al. [212] branch-and-bound algorithm. Côté et al. [83] found better results by using a Benders' decomposition in which the master problem is a MILP model and the slave is solved through branch-and-bound.

To the best of our knowledge, Jakobs [162] was the first to propose a metaheuristic (genetic) algorithm for the SPP. Since then, many metaheuristics have been proposed, e.g., by Iori et al. [160] to cite a well-known approach. As the focus of this chapter is on exact solutions, we do not give an exhaustive list of heuristic and metaheuristic algorithms: the interested reader can refer to the recent articles by Özcan et al. [227] and by Thomas and Chaudhari [272] for updated reference lists.

Coming to the SCP, the literature particularly focused on heuristics and metaheuristics rather than on exact methods: see, e.g., Burke et al. [51] for a classical approach, or Wei et al. [291] for an extensive literature review. To the best of our knowledge, the only exact approaches for the SCP were proposed by Kenmochi et al. [172] and by Arahori et al. [13]. Both approaches are based on branch-and-bound techniques. The former algorithm transforms a general instance into a perfect-packing one by adding small items of size 1×1. It then applies a branch-and-bound method that fathoms nodes through powerful cuts derived from those proposed in [185] (see above). The latter algorithm improves the enumeration scheme by adding innovative cuts and strong fathoming criteria. According to the computational experiments presented in [13] this can be considered the state of the art of the exact algorithms for the SCP. Note that these two algorithms are also used for the SPP, and that both present very competitive results on this problem variant too.

In the last decades, several problems related to the SCP were also studied. We cite in particular the problems of *Packing Squares into a Square* (PSS) and of *Packing Rectangles into a Square with Rotation* (PRSR), which ask for the minimum area square required to pack a given set of items. As we will see, the approach we propose can be easily applied to solve them. In the early nineties, Leung et al. [185] proved that the PSS is strongly $\mathcal{NP}$-hard, hence the same holds for PRSR. The two problems have been studied by Picouleau [229], who gave some simple heuristics for the PSS, by Caprara et al. [55], who proposed lower bounds and determined their worst-case performance, and by Correa [78], who presented a polynomial-time approximation scheme for the PRSR. To the best of our knowledge, the only exact approach for the PSS and the PRSR is the one proposed by Martello and Monaci [211].

Another related problem to which our approach can be extended is the *Pallet Loading Problem* (PLP), which consists in packing the maximum number of identical rectangles into a given larger rectangle. This problem, whose complexity status is currently unknown, was introduced in the late sixties by Barnett and Kynch [22] and then extensively studied by many researchers, who proposed both exact methods (see, e.g., Dowsland [109], Alvarez-Valdes et al. [5], or Ahn et al. [2]) and powerful heuristics (see, e.g., Lins et al. [190] or Birgin et al. [41]). The number of references being huge, we refer the interested reader to the recent, very complete survey by Silva et al. [260].

The exact approach we propose is based on the logic based Benders' decomposition, that was formally developed by Hooker [155], and applied with success by Hooker and Ottosson [157] to 0-1 programming and by Hooker [156] to planning and scheduling problems. The approach was later specialized to mixed integer programming by Codato and Fischetti [69] who introduced the so-called combinatorial Benders' cuts. Such cuts were successfully used by Côté et al. [83] for solving the SPP. In the *logic based Benders' decomposition*, the classical master problem is frequently solved as a MILP and the slave through constraint programming (although other solution techniques can be used). Constraint programming techniques were used by Clautiaux et al. [68] to solve the recognition version of the SPP.

5.3 Mathematical model

We assume the existence of a coordinate system with origin in the bottom-left corner of the strip, having integer coordinates on the horizontal (width) and the vertical (height) axis. For any integer coordinate i on the width axis, we call the unit-width vertical space above interval $[i, i+1)$ a *column*. We say that an item j *engages* a column q whenever j is packed so as to occupy a portion of column q. We will use the following notation

- $\mathcal{N} = \{1, 2, \ldots, n, n+1, \ldots, 2n\}$ = set of the given n items ($j = 1, 2, \ldots, n$) followed by a copy of each of them rotated by 90°, i.e., such that $w_j = h_{j-n}$ and $h_j = w_{j-n}$ ($j = n+1, n+2, \ldots, 2n$);
- $\mathcal{W} = \{0, 1, \ldots, W-1\}$ = set of all integer width coordinates (abscissae) where the bottom left corner of an item can be packed;
- $\mathcal{W}(j, q) = \{q - w_j + 1, \ldots, q\}$ = set of integer abscissae a such that, if item j is packed with its bottom-left corner at a (at any ordinate), then it engages column q.

From now on, when no confusion arises, we will use the term 'item' to denote either an input item or a rotated copy. By introducing decision variables

- x_{jp} = binary variable taking the value 1 if the bottom-left corner of item j is packed at abscissa p, and the value 0 otherwise ($j \in \mathcal{N}, p \in \mathcal{W}$);
- y_j = ordinate at which the bottom edge of item j is packed ($j \in \mathcal{N}$);
- z = overall height of the packing,

the SCP can be modeled as:

$$\min \quad z \tag{5.1}$$

$$\sum_{p \in \mathcal{W}} x_{jp} + x_{(n+j)p} = 1 \qquad j = 1, 2, \ldots, n, \tag{5.2}$$

$$y_j + h_j \le z \qquad j \in \mathcal{N}, \tag{5.3}$$

$$\text{nonoverlap} \left\{ [y_j, y_j + h_j], j \in \mathcal{N} : \sum_{p \in \mathcal{W}(j,q)} x_{jp} = 1 \right\} \qquad q \in \mathcal{W}, \tag{5.4}$$

$$x_{jp} \in \{0, 1\} \qquad j \in \mathcal{N}, p \in \mathcal{W}, \tag{5.5}$$

$$y_j \ge 0 \qquad j \in \mathcal{N}, \text{ integer.} \tag{5.6}$$

Constraints (5.2) ensure that each item is packed exactly once, either rotated or not. Constraints (5.3) impose that no item exceeds the height of the strip. Logical constraints (5.4) (of "constraint programming" flavor) prevent items from overlapping by imposing, for each column q, that the vertical intervals $[y_j, y_j + h_j)$ associated with all items j that engage column q do not overlap.

Constraints (5.4) can equivalently be expressed through linear inequalities, as done by Baldacci and Boschetti [20] and by Castro and Oliveira [58]. However computational experiments (see, e.g., [58]) show that the direct use of the resulting mathematical model with an MILP solver is not very effective. In the next sections we show that decomposition techniques produce instead reduced models that can be solved more easily.

5.4 Preprocessing

We adapted a number of techniques from the literature to preprocess SCP instances through reduction, heuristic initialization, and lower bound computations.

A size reduction of strip and item width is attempted through an adaptation to the SCP of two techniques proposed by Boschetti and Montaletti [43] for the SPP:

(a) determine, through dynamic programming, the maximum width $W' \leq W$ that can be obtained by packing side by side any subset of the given items: if $W' < W$, then the strip width can be set to W';

(b) for each item $j \in \mathcal{N}$ compute, through dynamic programming, the maximum width w'_j that a subset of items can take when packed side by side with j: if $w_j + w'_j < W$, then the item width can be increased to $W - w'_j$. Note that, after this process, the width (resp. height) of an item $j > n$ is no longer necessarily equal to the height (resp. width) of item $j - n$ (see Section 5.3).

A third reduction technique, also presented in [43] but developed by Martello et al. [212] for the SPP, is based on a dominance criterion that packs at the bottom of the strip some large items and all the small items that would fit side by side with the large ones, provided such a packing is possible. However, its extension to the SCP appears to have little usefulness, due to difficulty in handling the possible rotation of the considered large items.

As mentioned in Section 5.2, several heuristics exist for the SCP. We adopted the *best-fit heuristic* by Burke et al. [51], which is relatively simple and has a good average performance. We implemented the three versions proposed in [51], that differ from each other in the position where the next item is packed (*Leftmost*, *Next to Tallest Neighbor*, *Next to Shortest Neighbor*).

Concerning lower bounds, we computed a value LB as the best (higher) between

(i) the simple bound $L_1 = \max\left(\lceil\sum_{j=1}^{n} w_j h_j / W\rceil, \max_{j=1,\ldots,n}\{\min(w_j, h_j)\}\right)$, and

(ii) L_2 = rounded up solution value of the continuous relaxation of the *Integer Linear Programming* (ILP) model that will be introduced in Section 5.6.

Once LB has been computed, we execute a *truncated* (heuristic) version of the branch-and-bound algorithm proposed by Kenmochi et al. [172], which looks for a feasible solution of value LB: we selected the so called *G-staircase placement* with *DP cuts*, as its performance on particular types of instance (where the unused space is very small) was shown to be very good. Such strategy maintains, at each decision node, a staircase-shape envelope that separates the two regions where the next items may or may not be placed. We limited the exploration of the branch-decision tree by: (i) heuristically killing decision nodes that are unlikely to lead to an optimal solution, and (ii) cutting out "small" portions of the strip region available to the next packings with the objective of maintaining a low number of steps in the staircase.

5.5 Decomposition algorithm

The decomposition introduced by Benders [34] solves a difficult problem through the iterative solution of two subproblems: the *master problem* and the *slave problem.* The master is generally a relaxation of the given problem, while the slave either provides an overall optimal solution or generates cuts to be added to the master at the next iteration. The algorithm we propose is derived from relatively recent developments of this technique: the logic based Benders' decomposition (Hooker [155]) and the combinatorial Benders' cuts (Codato and Fischetti [69]). The latter method was used by Côté et al. [83] for the SPP.

Our master problem is an extension of the *One-Dimensional Contiguous Bin-Packing Problem* (1CBP), a relaxation that was introduced by Martello et al. [212] for the SPP. For the SPP, the 1CBP is obtained by horizontally 'cutting' each two-dimensional $w_j \times h_j$ item into h_j unit height *slices* of length w_j, and the objective is to pack all resulting slices into the minimum number of one-dimensional bins of capacity W so that slices belonging to the same item are packed into contiguous bins.

The master problem we developed solves a variant of the 1CBP in which each item j is vertically cut into w_j unit width slices of height h_j. It then looks for a feasible solution of threshold value LB (the capacity of the one-dimensional bin) that contiguously packs the slices into at most W bins. In addition, it is necessary to take into account the possibility of rotating the items, and hence both copies of each two-dimensional item (see Section 5.3) are cut into slices, and we impose that the solution packs only one of the two sets of slices. Figure 5.1(a) provides a pictorial representation of a master problem solution.

In Section 5.6 we describe the approach we developed for solving the master problem. Once the master has been solved, the slave must check the feasibility of the resulting solution for the original SCP by looking for a rearrangement of the slices that reconstructs the original two-dimensional items without exceeding the height of the strip produced by the master. Figure 5.1(b) shows an SCP solution corresponding to the master solution of Figure 5.1(a). In Section 5.7 we discuss the approach we adopted for its solution.

The overall approach we propose starts with preprocessing and sets a possible strip height at a threshold provided by a lower bound. At each iteration, it looks for a feasible solution of value equal to the current threshold through the above master-slave method. If it can prove that no such solution exists, then the threshold is conveniently increased. The method can be summarized as follows.

Algorithm SINGLE:

1. reduce the instance (see Section 5.4);

2. execute the best-fit heuristics to obtain an upper bound UB, and compute lower bound

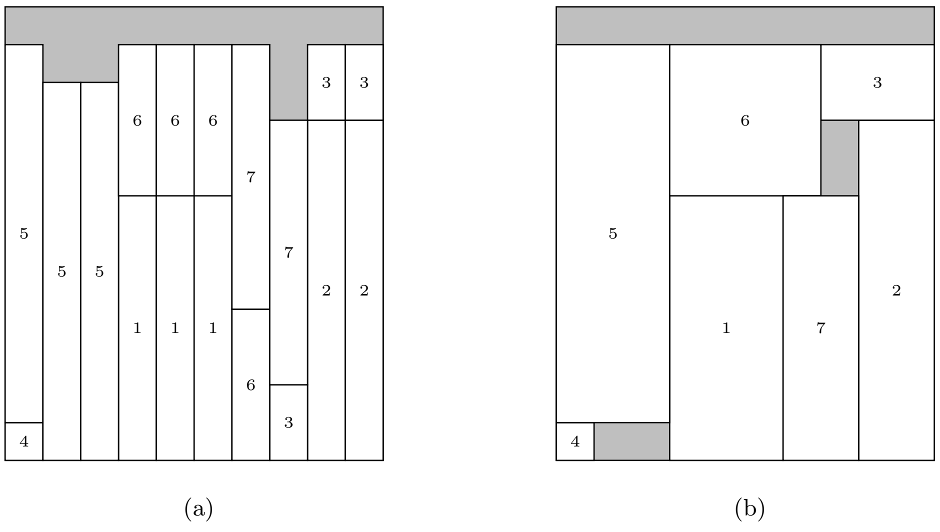


Figure 5.1: (a) master problem solution; (b) corresponding SCP solution

```
   $LB$ (see Section 5.4);
3. execute the truncated branch-and-bound heuristic (see Section 5.4);
4. **if** a solution of value $LB$ has been obtained **then** $UB := LB$ and **terminate**;
   **while** $UB > LB$ **do**
5.    exactly solve the master problem with threshold height $LB$ (see Section 5.4);
6.    **if** no solution of value $LB$ is found **then** remove all cuts, increase $LB$, and **continue**;
7.    exactly solve the slave problem;
8.    **if** a solution of value $LB$ is obtained **then** $UB := LB$ and **terminate**;
9.    add improved Benders' cuts to the master problem (see Section 5.7)
   **end while.**
```

In many cases, the value of UB is not much higher than that of LB, and hence increasing LB by one at Step 6 is a reasonable choice. (In our computational experiments, we rarely found instances with $z > LB + 1$.) For different cases, a binary search between LB and UB could be preferable.

5.6 Master problem

As mentioned in Section 5.5, the aim of the master problem is to find a feasible solution of given value LB (threshold) to an adaptation of the one-dimensional bin packing problem with contiguity constraints. In [83], the master arising from the SPP was solved using two exact algorithms: a combinatorial branch-and-bound and, when it fails due to time limit, the Benders' decomposition of a mathematical model given by the oriented version of (5.1), (5.2), and (5.5), with an additional constraint imposing that, for any column, the sum of the heights of the items engaging it does not exceed z.

Our master was solved through an ILP model of an adaptation of the 1CBP introduced in [212] for the SPP (see Section 5.5). The type of modeling we adopted has its roots in the ARCFLOW model proposed by Valério de Carvalho [278] for the one-dimensional cutting stock problem. (Note that, in spite of its similar name, this problem is totally different from the two-dimensional problem we are considering.) We make use of a directed graph with $W + 1$ vertices $0, 1, \dots, W$, and arc set $A = A_1 \cup \dots \cup A_{2n} \cup A_0$ where

$$A_j = \{(d, e) : 0 \le d < e \le W \text{ and } e - d = w_j\} \ (j = 1, \dots, 2n) \tag{5.7}$$

and $A_0 = \{(d, d+1) : 0 \le d < W\}$. In other words, there is an arc between two vertices d and e if there is an item (either an input item or a rotated copy) whose width is equal to $e - d$. In addition, there is a set A_0 of *loss arcs*, corresponding to identical dummy items having unit width and height, used for ensuring flow conservation, as it will be clear from the model.

Consider the following numerical example: $n = 3$, $W = 5$, $w_1 = 5$, $h_1 = 1$, $w_2 = h_2 = 3$, $w_3 = h_3 = 2$: Figure 5.2 shows the resulting graph (disregard by the moment the values on the arcs, which give the item heights). The topmost arc corresponds to input item 1, non rotated (A_1). Then we have the three arcs corresponding to input item 2 ($A_2 \equiv A_5$) and the four arcs corresponding to input item 3 ($A_3 \equiv A_6$). Finally , the loss arcs (A_0, dotted) and, on the bottom, the five arcs corresponding to the rotated copy of item 1 (A_4). (Note that, formally, one should also have a second, useless, copy of the arc sets corresponding to input items 2 and 3, for which, however, rotation does not make sense.)

Let us introduce variables $\bar{x}_{jde}$ ($j = 0, \ldots, 2n$, $d = 0, \ldots, W-1$, $e = 1, \ldots, W$) to represent the number of times arc $(d, e) \in A_j$ is selected. For $j = 1, 2, \ldots, 2n$, binary variable $\bar{x}_{jde}$ takes the value 1 iff item j is selected and packed with its bottom left corner at abscissa d, engaging columns $d, d+1, \ldots, e-1$. For $j = 0$, $\bar{x}_{jde}$ is a non-negative integer variable giving the number of 1×1 dummy items packed at abscissa d. Let us denote by $\delta_j^-(e)$ (resp. $\delta_j^+(e)$) the set of arcs of A_j entering (resp. emanating from) vertex e. The

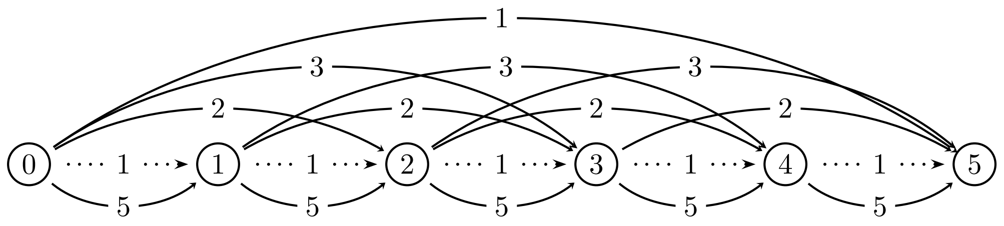


Figure 5.2: Arcs generated for the example instance

master problem is then to find a feasible solution to

$$\sum_{j=0}^{2n}\left(-\sum_{(d,e)\in\delta_j^-(e)} h_j\bar{x}_{jde}+\sum_{(e,f)\in\delta_j^+(e)} h_j\bar{x}_{jef}\right)=\begin{cases} LB & \text{if } e=0;\\ -LB & \text{if } e=W;\\ 0 & \text{otherwise,}\end{cases} \quad e=0,1,\dots,W, \tag{5.8}$$

$$\sum_{(d,e)\in A_j}\bar{x}_{jde}+\sum_{(d,e)\in A_{n+j}}\bar{x}_{(n+j)de}=1 \qquad j=1,2,\dots,n, \tag{5.9}$$

$$\bar{x}_{jde}\in\{0,1\} \qquad j=1,2\dots,2n;(d,e)\in A_j, \tag{5.10}$$

$$\bar{x}_{0de}\geq 0, \text{integer} \qquad (d,e)\in A_0. \tag{5.11}$$

Constraints (5.8) impose: (i) the solution value LB to the flow emanating from node 0 and to that entering node W (hence ensuring that every item is entirely packed within the strip), and (ii) the flow conservation at nodes $1,\dots,W-1$. Constraints (5.9) impose that, for each item j, either the original item or its rotated version is packed. The model has $W+n+1$ constraints and $O(nW)$ variables (as, from (5.7), each arc set A_j has no more than W arcs).

In Figure 5.2, the values on the arcs represent the height of the corresponding items, i.e., in the model, the flow that circulates along the arc every time it is selected. For the previous numerical example, Figure 5.3 shows the arcs selected in a feasible solution of threshold value 3, while Figure 5.4 provides a possible graphical representation of such solution. The non-zero $\bar{x}_{jde}$ values are

- $\bar{x}_{2,0,3}=1$, i.e., item 2 has its bottom left corner packed at abscissa 0, it engages columns $0,1,2$, occupying three height units in each of them;
- $\bar{x}_{3,3,5}=1$, i.e., item 3 has its bottom left corner packed at abscissa 3, it engages columns $3,4$, occupying two height units in each of them;
- $\bar{x}_{4,0,5}=1$, i.e., item 4 (the rotated copy of item 1) has its bottom left corner packed at abscissa 0, it engages columns $0,1,2,3,4$, occupying one height unit in each of them;

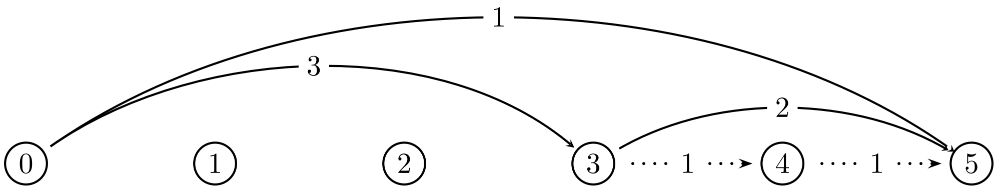


Figure 5.3: Set of arcs selected in the optimal solution of the master for the example instance

- $\bar{x}_{0,3,4} = \bar{x}_{0,4,5} = 1$, i.e., loss arcs engage one height unit (waste) in columns 3 and 4 (to satisfy the flow conservation constraints).

5.7 Slave problem and cut generation

Let $[\bar{x}^s_{jde}]$ be the solution to the current master problem, and z^s (equal to the current threshold LB) its value. The solution identifies a subset $\mathcal{N}^s \subset \mathcal{N}$ containing one copy (either rotated or not) of each of the n input items, together with the corresponding abscissa, and hence the columns engaged by the item. The slave problem, *y-check* in the following, is then to decide if there exists an ordinate for each item $j \in \mathcal{N}^s$ such that all items are packed without overlapping and without exceeding the height of the strip. This problem is known to be $\mathcal{NP}$-hard, as shown by Côté et al. [83].

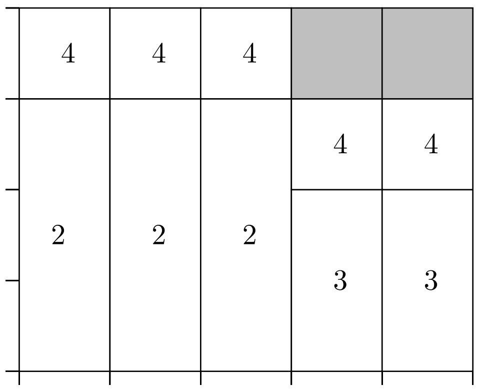


Figure 5.4: Graphical representation of an optimal master solution for the example instance

The y-check problem arising for the SPP was attacked in [83] through a specialized branch-and-bound algorithm. We preferred to explore the possibility of obtaining an effective overall algorithm by solving the slave through a simpler, non-tailored constraint programming approach. Remind that, for each item j, w_j and h_j give its width and height in the selected orientation. The slave problem is then to find ordinates y_j $(j \in \mathcal{N}^s)$ satisfying (see (5.1)-(5.6))

$$y_j + h_j \le z^s \qquad j \in \mathcal{N}^s, \tag{5.12}$$

$$\text{nonoverlap}\left\{ [y_j, y_j + h_j], j \in \mathcal{N}^s : \sum_{p \in \mathcal{W}(j,q)} \bar{x}^s_{jp(p+w_j)} = 1 \right\} q \in \mathcal{W}, \tag{5.13}$$

$$y_j \ge 0, \ \text{integer} \qquad j \in \mathcal{N}^s. \tag{5.14}$$

Constraints (5.12) impose that no item exceeds the height of the strip, while constraints (5.13) impose that no two items overlap. If the slave problem returns a solution satisfying (5.12)-(5.14), then we know that the original SCP instance has been optimally solved.

If instead the slave can prove that no feasible solution to (5.12)-(5.14) exists, we generate a set of lifted combinatorial Benders' cuts, following the method developed in [83] for the SPP:

(i) heuristically find a (possibly small) subset $\widetilde{\mathcal{N}}^s \subseteq \mathcal{N}^s$ of items such that any solution including them at their current abscissa is infeasible for the slave;

(ii) solve an LP to identify, for each item $j \in \widetilde{\mathcal{N}}^s$, an interval $[l^s_j, r^s_j]$ that includes its current abscissa, and has the following property: If all the items $j \in \widetilde{\mathcal{N}}^s$ are in the master solution, each with the abscissa in $[l^s_j, r^s_j]$, then the same (infeasible) slave problem will be obtained;

(iii) add a cut to prohibit this, i.e.,

$$\sum_{j \in \widetilde{\mathcal{N}}^s} \sum_{(d,e) \in A_j, l^s_j \le d \le r^s_j} \bar{x}_{jde} \le |\widetilde{\mathcal{N}}^s| - 1. \tag{5.15}$$

Observe that cuts (5.15) are only valid for a given threshold value $z^s = LB$: a certain master solution $[\bar{x}^s_{jde}]$ can be infeasible for LB, but feasible, at a subsequent iteration, for a higher threshold value. This corrects an imprecision in the overall descriptive model of

Section 2.3 in [83], that was pointed out by Hashimoto et al. [150].

5.8 The case of identical item copies

The approach we have introduced so far is obviously valid if the input instance includes a number of identical items. For such cases however, a more effective algorithm, called MULTI in the following, can be obtained through simple modifications of SINGLE:

- at Step **1**, the item width increase (b) of Section 5.4 is deactivated in order to avoid the creation of dissimilarities among copies of identical items;
- at Step **5**, we can group together identical items: let m denote the number of items that are different from each other and b_j the number of copies of item j ($j = 1, 2, \ldots, m$). We then perform a modified version of the master (5.8)-(5.11), in which n is replaced by m, constraints (5.9) become

  $$\sum_{(d,e)\in A_j} \bar{x}_{jde} + \sum_{(d,e)\in A_{m+j}} \bar{x}_{(m+j)de} = b_j \quad j = 1, 2, \ldots, m, \tag{5.16}$$

  while (5.10) and (5.11) merge to

  $$\bar{x}_{jde} \geq 0, \text{integer} \quad j = 0, 1, \ldots, 2m; (d, e) \in A_j. \tag{5.17}$$

  For $j = 1, 2, \ldots, 2m$, $\bar{x}_{jde}$ is now a non-negative integer variable giving the number of copies of item j that are selected and packed with their bottom left corner at abscissa d, engaging columns $d, d+1, \ldots, e-1$. The meaning of $\bar{x}_{0de}$ does not change. Once the master problem has been solved, we map its solution into the original instance;
- at Step **7**, for pairs of identical items (j, k) $(j, k \in \mathcal{N}^s,\ j < k)$ assigned to the same abscissa, it is imposed that $y_j < y_k$, in order to avoid symmetries in the slave problem;
- at Step **9**, as constraint (5.15) does not extend to general integer variables, when the slave proves that (5.12)-(5.14) has no solution, instead of adding a cut, we kill the decision node producing $\bar{x}_{jde}$ in the master enumeration tree.

Our overall approach, referred to as DIM in the following, executes algorithm SINGLE of Section 5.5 when all items are different, and algorithm MULTI otherwise.

5.9 Computational experiments

In order to test the effectiveness of the proposed approach we performed extensive computational experiments on classical SCP instances from the literature. As we will see, our approach can be easily modified to handle relevant variants of the problem. We thus also tested it on literature instances of the square and rectangle packing problems PSS and PRSR, as well as of the pallet loading problem PLP (see Section 5.2).

We used CPLEX 12.6.0 for all steps of the algorithm: master, slave, and cuts. The logical constraints used in the slave were `IloNoOverlap` (which models constraints (5.13)) and in addition, for MULTI, `IloEndBeforeStart` (to avoid symmetries). In the implementation of SINGLE, the slave problem was invoked within the `lazy callback` procedure, while in that of MULTI it was invoked within the `incumbent callback` procedure. Both procedures are available in the branch-and-cut framework, so, as soon as the master produces a decision node with integer solution, the slave checks it for feasibility: if the check fails, SINGLE adds new cuts (5.15) to the master, while MULTI just kills the corresponding decision node. This implementation of the logic based Benders' decomposition is sometimes called *branch-and-check* (see Thorsteinsson [273]).

The experiments were performed on an Intel Xeon E3-1220 3.10 GHz with 8 GB RAM, equipped with four cores. In order to have a fair comparison with other algorithms and machines, we always used a single core. The time limit was set to 3 600 seconds per instance, with a limit of 10 seconds for the execution of the truncated heuristic of Section 5.4.

The performance of DIM was compared with that of other algorithms, in most cases using the results published by their authors. As the involved computers used have different speeds, we evaluated the CPU performances using the indicators given by PassMark© Software (see `https://www.cpubenchmark.net/`). The indicator for our computer is 6 106. Concerning G-Staircase, the best algorithm among those presented by Kenmochi et al. [172], the experiments were re-run using a working copy of the original code, kindly provided by the authors. (These experiments confirmed a good reliability of the PassMark indicators.)

Furthermore, different authors adopted different ways for handling the cases of time limit in the evaluation of the average CPU time: some included the time limits in the computations, some did not. In order to allow comparisons, our tables provide, for each algorithm, the average CPU times (in seconds) relative to the instances solved to proven optimality, and their number (# opt). The highest number of solved instances is highlighted in bold.

All the instances we used for our experiments can be downloaded from the library of codes and instances of the University of Bologna Operations Research Group, `http://or.dei.unibo.it/library`. In the following, we examine the outcome of the computational experiments on the different instance classes.

5.9.1 SCP instances

This is the main benchmark, as it refers to instances of the problem DIM is tailored for. We used the following classical two dimensional instances from the literature:

- NGCUT: a set of 12 knapsack instances proposed by Beasley [26];
- CGCUT: a set of 3 knapsack instances proposed by Christofides and Whitlock [64];
- GCUT: a set of 13 knapsack instances proposed by Beasley [25];
- BENG: a set of 10 packing instances proposed by Bengtsson [35];
- HT: a set of 9 SCP instances proposed by Hopper and Turton [158];
- BKW: a set of 13 SCP instances proposed by Burke et al. [51];
- CLASS01, ..., CLASS10: ten sets of 50 bin packing instances each (six proposed by Berkey and Wang [38], and four later proposed by Martello and Vigo [216]). Each class is composed by five groups of 10 instances each, with n = 20, 40, 60, 80, and 100, respectively.

For all cases, but HT and BKW, the strip width W was set to the width of the original two-dimensional container (knapsack or bin).

The performance of Algorithm DIM was compared with that of the algorithms by Kenmochi et al. [172] and by Arahori et al. [13]. The experiments reported in the latter paper were made on a Pentium 4 3.0GHz, with a time limit of 3 600 seconds. The performance indicators of this machine is 357, i.e., the entries in Table 5.1 for the algorithm in [13] should be multiplied by 0.058. For the instances for which a comparison with both other SCP algorithms can be done, it turns out that DIM always solved at least all the instances solved by the best between the algorithms in [172] and [13], sometimes with higher, sometimes with smaller CPU times. Similar results were obtained for instances CLASS*, that could only be compared with the working code provided by the authors of [172]. It can be observed that, for both algorithms, CLASS01 and CLASS02 are relatively easy, CLASS03 and CLASS04 are difficult, while CLASS05, CLASS06, CLASS07, CLASS08, and CLASS10 are extremely hard. The most relevant difference concerns CLASS09, for which DIM appears especially powerful. This could be explained by the fact that the instances of such class have a number of items with large width and height: their packing frequently creates large "holes" in the strip, and hence lower bounds are not tight, while the ARCFLOW model performs well because the number of arcs is small.

By comparing our experiments with those in [83], we can observe that allowing item rotation considerably increases the difficulty of the problem. For example, the instances of CLASS07 (which were all solved in the oriented case) have many oblong horizontal items: this allows a powerful initial reduction through dominance criteria when the items are oriented (which makes the instance quite easy to solve), but no reduction at all when they can be rotated.

Worth is mentioning that we solved for the first time instance GCUT02: The optimal solution is reported in Figure 5.5, where item numbers and (possibly rotated) sizes are provided within the rectangles. The solution was obtained in 108 CPU seconds.

Table 5.1: SCP instances. CPU times to be multiplied by 0.058 for [13]

Name	# inst.	Arahori et. al [13]		Kenmochi et. al [172]		Algorithm DIM	
		# opt.	Avg. time (s)	# opt.	Avg. time (s)	# opt.	Avg. time (s)
NGCUT	12	**12**	0.4	11	1.0	**12**	25.8
CGCUT	3	**2**	0.0	**2**	0.2	**2**	0.2
GCUT1-4	4	1	0.8	1	141.6	**2**	59.2
GCUT5-13	9	-	-	2	425.7	**5**	35.9
BENG	10	**10**	0.1	**10**	0.2	**10**	0.2
HT	9	**9**	0.0	**9**	0.1	**9**	0.2
BKW1-12	12	-	-	**9**	6.7	**9**	1.0
BKW13	1	-	-	0	-	0	-
CLASS01	50	-	-	44	2	**50**	0.4
CLASS02	50	-	-	**50**	0	**50**	0.3
CLASS03	50	-	-	2	82.9	**9**	846.2
CLASS04	50	-	-	18	35.6	**19**	145.3
CLASS05	50	-	-	0	-	**1**	1289.4
CLASS06	50	-	-	0	-	0	-
CLASS07	50	-	-	0	-	0	-
CLASS08	50	-	-	0	-	0	-
CLASS09	50	-	-	0	-	**45**	293.3
CLASS10	50	-	-	0	-	**2**	1400.4

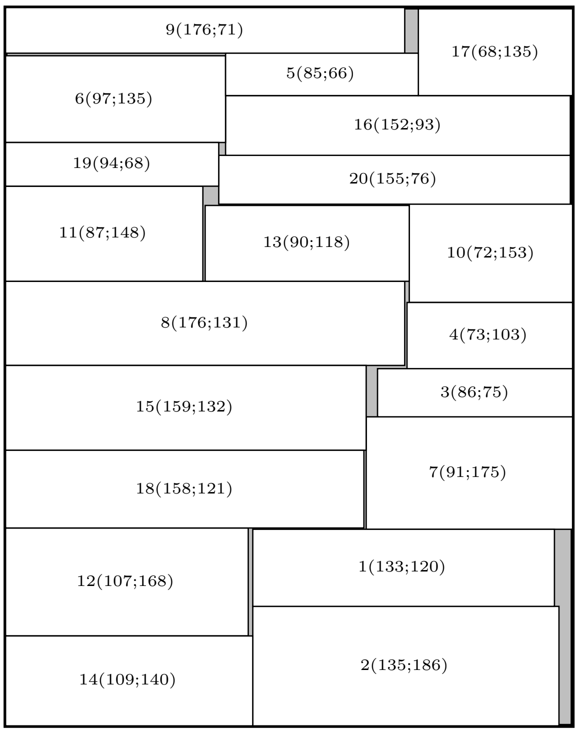


Figure 5.5: Optimal solution found for GCUT02 ($W = 250$, $z_{opt} = 1118$)

5.9.2 Rectangle packings

In this section we deal with the rectangle packing problems mentioned in Section 5.2. Given n rectangular items of integer width w_j and height h_j ($j = 1, \dots, n$), problem PRSR asks for the minimum area square needed to pack all the items without overlapping. The special case in which $w_j = h_j$ for all j is denoted as PSS. Both problems can be solved through Algorithm DIM as follows. We set $W = LB = \left\lceil \sqrt{\sum_{j=1}^{n} w_j h_j} \right\rceil$, and we execute DIM. While no feasible solution is found, we increase both W and LB by one unit, and iterate. We evaluated this approach on classical instances from the literature, and compared our results with those obtained by Martello and Monaci [211] through a specialized algorithm that attempts squares of different size through binary search, namely:

- NGCUT, CGCUT, GCUT, BENG: see Section 5.9.1;
- GARD: PSS instances with $w_j = h_j = j$ for $j = 1, \dots, n$, proposed by Gardner [129];

- KORF: PRSR instances with $w_j = j$, $h_j = j + 1$ for $j = 1, \ldots, n$, proposed by Korf et al. [181].
- RND_S: 200 PSS instances proposed by Martello and Monaci [211];
- RND_R: 200 PRSR instances proposed by Martello and Monaci [211].

Both RND_S and RND_R consist of four groups of 50 instances each, with n = 5, 10, 15, and 20, respectively.

The experiments in [211] were made on an Intel i5-750 CPU (2.67 GHz), using IBM-ILOG Cplex 12.5.1 and Gurobi 5.6, with a time limit of 3 600 seconds per instance. The performance indicator for such computer is 3 732, so the entries given in Table 5.2 for this algorithm should be multiplied by 0.611. The results show that Algorithm DIM always solved at least all the instances solved in [211], in several cases with smaller CPU times. In particular, all BENG instances were solved to proven optimality, and 5 more GARD instances were solved with respect to [211]. (Note however that 6 additional instances were solved by Korf et al. [181] through a highly specialized algorithm.) Worth is mentioning that instances RND_* with $n \geq 15$ confirm to be very difficult to solve exactly. We tested DIM with a larger time limit on the only RND_R 10 instance it could not solve: a proven optimal solution was found in 3 918 seconds.

Table 5.2: Rectangle packing instances. CPU times to be multiplied by 0.611 for [211]

Name	# inst.	Martello and Monaci [211]		Algorithm DIM	
		# opt.	Avg. time (s)	# opt.	Avg. time (s)
NGCUT	12	10	357.1	**12**	67.1
BENG	10	3	0.0	**10**	0.2
CGCUT	3	0	-	**1**	734.6
GCUT	13	**3**	560	**3**	742.7
KORF	40	17	73.6	**24**	2.2
GARD	40	16	2.2	**21**	191.1
RND_S 05	50	**50**	0.0	**50**	0.3
RND_S 10	50	**50**	341.2	**50**	81.8
RND_S 15	50	**5**	186.9	**5**	894.3
RND_S 20	50	0	-	0	-
RND_R 05	50	**50**	0.0	**50**	0.4
RND_R 10	50	39	384.2	**49**	464.2
RND_R 15	50	0	-	0	-
RND_R 20	50	0	-	0	-

5.9.3 Pallet loading

Our last set of experiments was performed on instances of the pallet loading problem (PLP). Given a rectangle of width W and height H, the problem is to pack into it, without overlapping, the maximum number of identical rectangular items of integer width w_1 and height h_1. We attacked the problem through Algorithm DIM as follows. We initialize n to $\left\lfloor \frac{WH}{w_1 h_1} \right\rfloor$ (an obvious upper bound on the solution value), we define $w_j = w_1$, $h_j = h_1$ for $j = 2, \ldots, n$, we set $LB = H$, and we execute DIM. While no feasible solution is found, we decrease n by one, and iterate.

The PLP has been intensively studied since the sixties, and many specialized algorithms have been proposed for its solution. The comprehensive recent survey by Silva et al. [260] has a bibliography of over fifty references, and examines computational experiments from the literature involving some three million instances. Additional experiments are reported by Ahn et al. [2]. (The two articles appeared almost at the same time, so they do not cite each other.).

Although the MULTI version of our approach exploits the identical item property, it is not guided in any way by the very strong property that *all* items are identical, nor it uses the powerful heuristics and upper bounds that such property exploits. Our objective here was not to beat very specialized algorithms, but to see if we could solve to proven optimality some previously unsolved, or very difficult to solve, instances from the literature. We thus tested MULTI on

- GROUP1, ..., GROUP5: five sets, each containing between 10 and 21 instances, used by Ahn et al. [2];
- HARD II: a set of 241 instances that were identified by Alvarez-Valdes et al. [5] as the most difficult ones among the 40 609 "Cover II" instances they tested. (All these instances have been solved to proven optimality (see Birgin et al. [41]);
- HARD III: a subset of 970 instances for which optimality is hard to prove, among the 98016 initially proposed in set "Cover III" by Alvarez-Valdeset al. [6]. Attempts to exactly solve such instances, identified by Birgin et al. [41], is currently underway (see the web page `http://lagrange.ime.usp.br/~lobato/packing/cover3.php` they maintain);
- SAMPLE59: a set of 59 instances selected by Silva et al. [260] as particularly significant, as they have been used as benchmark instances in at least two numerical experiments from the literature.

For all instances but most of those in HARD III, the optimal solution is known. The available information about the solution times is not uniform. For GROUP*, Ahn et al. [2]

provide the average CPU times for their exact solution, obtained on an Intel I5-750 CPU, 3 GB, with performance indicator 4 277 (i.e., their times should be multiplied by 0.7). For HARD II, Alvarez-Valdes et al. [5] provide the median CPU time obtained by using CPLEX 7.0 on a PC Pentium III 850 MHz, CPLEX 7.0. We found no precise indicator for such computer: on the basis of indicators for similar machines, it should be very low (below 300), so their times should be multiplied by 0.05. For HARD III and SAMPLE59, no complete information on the solution times is available. In Table 5.3 we provide the comparable information for our approach.

The results show a good performance of DIM on all GROUP* instances but GROUP4 (probably because such instances are characterized by very large W and H values). The algorithm was also successful for HARD II instances, although no significant comparison with [5] can be done because of the big difference between the two CPLEX versions used. We solved to proven optimality the great majority of HARD III and SAMPLE59 instances within reasonable CPU times. The only unsolved SAMPLE59 instance is #58. Worth is mentioning that we solved 28 out of the 29 instances that have been indicated by Silva et al. [260] as those on which exact algorithms should be tested. Such instances (in parentheses the CPU times) are: #18 (7.4 s), #40 (0.6 s), #42 to #45 (0.3 s, 15.1 s, 1.5 s, 80.5 s), #26 to #34 (each in less than 0.7 s), #46 to #51 (each in less than 1.0 s), #52 (68.0 s), #53 (0.2 s), #54 (236.8 s), #55 (697.2 s), #56 (2.1 s), #57 (0.7 s), #58 (unsolved after 3 600 s), and #59 (5.5 s).

The results obtained on HARD III are particularly relevant, as optimality was still not proven for most (683) of these 970 instances. We were able to solve exactly 849 instances. In all such cases, it turned out that the lower bound found by the most powerful PLP heuristics was the optimal solution value, enforcing the conjecture of Birgin et al. [41] that their recursive partitioning approach always finds the optimal solution.

5.10 Conclusion

We have presented a logic based Benders' decomposition approach for the orthogonal stock cutting problem. We initially consider as the master a relaxation of the problem, that consists of a one-dimensional bin packing problem with contiguity constraints. We solve it through an ILP model based on the ARCFLOW formulation. The slave then checks, through constraint programming, if the solution obtained can lead to a feasible solution for the original problem. If the attempt is unsuccessful, we prevent the master to replicate the

Table 5.3: Pallet loading instances. CPU times to be multiplied by 0.700 for [2], and by 0.05 for [5]

Name	# inst.	Ahn et. al [2]		Alvarez-Valdes et. al [5]		Algorithm DIM		
		# opt.	Avg. time (s)	# opt.	Med. time (s)	# opt.	Avg. time (s)	Med. time (s)
G1	17	**17**	0.0			**17**	0.3	0.3
G2	21	20	2.3			**21**	4.6	0.4
G3	17	**17**	2.9			**17**	8.3	0.4
G4	15	**14**	530.5			9	11.3	39.9
G5	10	8	278.8			**10**	20.1	7.3
HARD II	241			204	506.9	**237**	17.9	1.3
HARD III	970					**849**	282.8	0.3
SAMPLE59	59					**58**	19.5	0.2

current solution, either by adding cuts or by killing the corresponding decision node, and iterate. Computational experiments on classical benchmarks from the literature show that the algorithm compares favorably with state-of-the art approaches. In particular, it solved for the first time instance GCUT02. We also tested an adaptation of the algorithm to other relevant problems. For rectangle packing problems, the algorithm compared favorably with a recent specialized algorithm. It also showed a good performance on hard instances of the pallet loading problem, solving 849 out of 970 instances for most of which the known solutions were not proved to be optimal.

Chapter 6

A Training Software for Orthogonal Packing Problems

[1]

We present an open source architecture for the interactive solution of packing problems in two dimensions. Although primarily developed for helping engineering students to understand the algorithmic approaches to the solution of difficult combinatorial optimization problems, thanks to its visual tools, the application can be useful to practitioners and developers. We give intuitive and formal definitions of the problems at hand, discuss two natural heuristic approaches, provide technical information on the application, and report the results of classroom experimental testings.

Keywords: Training software, Combinatorial optimization, Orthogonal packing, Visualization, Classroom experiments.

6.1 Introduction

In the teaching of combinatorial optimization algorithms it is helpful that students can use tools to easily understand the features and the difficulty of specific optimization problems. In this chapter we illustrate `TwoBinGame`, an open source visual application for interactively "playing" with (i.e., trying to solve) two-dimensional packing problems. The application was developed at the Universities of Bologna and of Modena and Reggio Emilia with the primary scope of guiding engineering students, but it can also be useful to practitioners and developers to visualize, test, and evaluate possible exact or heuristic algorithms. In `TwoBinGame`, the user operates in a computer generated environment where it is possible to interact and look for solutions by manipulating virtual objects. The

[1]The results of this chapter appears in: G. Costa, M. Delorme, M. Iori, E. Malaguti, and S. Martello, Training Software for Orthogonal Packing Problems, *Technical Report OR-17-4*, 2017 [82].

application was developed in Scala. We refer the reader to `http://scala-lang.org/documentation/books.html` for an overview of recent books on the Scala language.

Two-dimensional packing problems arise in a variety of industrial applications, when it is requested to allocate a given set of rectangular objects (*items*) to rectangular standardized stock units so as to minimize the waste area. In wood or glass industries, large rectangular sheets of material (*bins*) are cut to obtain given sets of rectangular elements. In warehouses, goods have to be allocated to shelves. When paging journals, it is necessary to place articles, photographs, and advertisements in the various pages. In all such cases, the standardized stock units can be seen as "large" rectangles to which smaller rectangles have to be allocated without overlapping. In other industrial applications, such as, e.g., paper or cloth production, the standardized stock unit consists of a roll of material (*strip*) from which one has to obtain the desired rectangular items by minimizing the used roll length. In both cases, two main variants occur in practice: either the items to be packed have a fixed orientation (e.g., when the material is corrugated or decorated), or they can be rotated (usually by 90 degrees). The interactive application we describe is capable of handling the resulting four variants (fixed length stock units or infinite length rolls; oriented or non oriented items).

Besides their industrial relevance, two-dimensional packing problems have considerable theoretical interest in the field of algorithmic combinatorial optimization. Already in 1965, Gilmore and Gomory [136] studied a two-dimensional packing problem and presented a column generation approach for its optimal solution. The analysis of the vast literature produced in the following 50 years is beyond the scope of this chapter. We refer the interested reader, e.g., to the book by Dyckhoff and Finke [113], to the more recent surveys by Lodi et al. [193, 195], and to the typology proposed by Wäscher et al. [289].

In order to experiment the developed application, we constructed a set of two-dimensional packing instances, and we asked a set of engineering students to test their skills by using the application to find good-quality feasible solutions with a prefixed time limit. Their solutions were compared to optimal and approximate solutions produced by ad-hoc algorithms from the literature.

The literature that presents interactive systems to study and solve decision problems is very varied and multidisciplinary. Although a complete survey is out of the scope of this chapter, some interesting contributions are briefly discussed in the following.

A first branch of this literature focused on the way interactive systems can be used in teaching. An early discussion was given in Asfahl et al. [16], who emphasized the fact that computer training programs can help capturing the attention of the audience and teaching non-conventional subjects. A few years later, Llaugel and Confesor [191] presented a computerized interactive program to teach quality control and quality improvement to undergraduate students. The program was based on the simulation of the process of filling medicine bottles, where over filling and under filling have a cost, and was assigned to groups of students who competed with each other to get lowest cost solutions. Crumpton and Harden [88] discussed the outcome of a test conducted on 20 students, who were asked to interact with a virtual reality tool simulating a *pick and pack problem* in the cereal industry: cereal boxes were proceeding down a conveyor and the operator was required to grab them, orient them, and then pack them into a larger box. The results were discussed mainly from an ergonomics point of view. Bodin and Gass [42] discussed key aspects in the teaching of the *analytic hierarchy process*. They developed a series of educational tests by using the Expert Choice Software and assigned them as classroom exercises to groups of students. Several pedagogical insights were discussed for the attempted tests. More recently, Costa et al. [80, 81] discussed Java tools developed for the teaching of graph theory, including applications to solve a number of optimization problems such as, e.g., shortest spanning trees, shortest paths, and maximum flows.

Another branch of this literature focused on the comparison between computerized and human behavior in the solution of optimization problems. Large attention was devoted to the well-known *Traveling Salesman Problem* (TSP), which requires to find a minimal cost hamiltonian cycle in a weighted graph. Mac Gregor and Ormerod [208] showed that humans are efficient in solving Euclidean TSP instances when compared to basic heuristics. They also discussed the practical difficulty of TSP instances as a function of the number of non-boundary points. A related discussion can be found in Chapter 4 of the TSP book by Applegate et al. [12]. A review of this area of research is provided by MacGregor and Chu [207]. Recently, Miyata et al. [218] studied the performance of young children on the TSP using a city-block metric. They showed that children tended to use strategies such as traveling straight to the farthest goal first, whereas adults relied more on nearest neighbor attempts.

In the next section we give a formal definition of the problems at hand. In Section 6.3 we provide technical information on the developed visual application, and in Section 6.4 we report the results of our experimental testing. Conclusions follow in Section 6.5.

6.2 Orthogonal packing problems

Let n be the number of items to pack, and denote as w_j and h_j the width and height of item j ($j = 1, 2, \dots, n$). When packing in a *strip*, the traditional representation is to see it as a *vertical* band, having fixed width and infinite height. In order to obtain a better display on a monitor, we decided instead of adopting an *horizontal* view, i.e., our strip has fixed height and infinite width.

Let H be the height of the bin or strip, and W be either the width of the bin or any upper bound on the maximum strip width. We consider two optimization problems, each in two variants. In the *strip* case, the objective is to pack all the n items by minimizing the width at which the strip is used. In the *bin* case, the objective is to pack a subset of items having the largest total area. In both cases the items are either *oriented* (they cannot be rotated) or they can be rotated by 90° degrees. We denote the resulting four variants as:

- [**SO:**] the stock unit is a strip and the items are oriented;
- [**SR:**] the stock unit is a strip and the items can be rotated by 90°;
- [**BO:**] the stock unit is a bin and the items are oriented;
- [**BR:**] the stock unit is a bin and the items can be rotated by 90°.

All problems we consider are strongly $\mathcal{NP}$-hard and can be modeled in different ways. In the following we adopt, for the sake of clarity, the modeling approach originally developed by Beasley [26], where the variables represent the coordinates at which the items are packed in the bin/strip. As it will be clear later, such variables correspond to the decisions that the user has to take when using our visual tool. We assume in the following that all numerical data are positive integers. Consider a Cartesian system restricted to non-negative integer coordinates with origin $(0, 0)$ in the bottom-left corner of the bin/strip.

We first consider problem SO. The following *Integer Linear Programming* (ILP) model makes use of a pseudo-polynomial number of binary decision variables

$$x_{pq}^{j} = \begin{cases} 1 & \text{if item } j \text{ is packed with its bottom-left corner at } (p, q); \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

for $j = 1, \dots, n$, $p \in W_j$, $q \in H_j$, where $W_j = \{0, 1, \dots, W - w_j\}$ and $H_j = \{0, 1, \dots, H - h_j\}$ denote all positions where the bottom-left corner of item j may be placed. The ILP model is:

$$\min\ z \tag{6.2}$$

$$\sum_{p\in W_j}\sum_{q\in H_j} x^j_{pq} = 1 \qquad (j=1,\dots,n) \tag{6.3}$$

$$\sum_{j=1}^{n}\sum_{\substack{p=r-w_j+1\\ p\in W_j}}^{r}\sum_{\substack{q=s-h_j+1\\ q\in H_j}}^{s} x^j_{pq} \le 1 \qquad (r=0,\dots,W-1; s=0,\dots,H-1) \tag{6.4}$$

$$\sum_{p\in W_j}\sum_{q\in H_j}(p+w_j)x^j_{pq} \le z \qquad (j=1,\dots,n) \tag{6.5}$$

$$x^j_{pq} \in \{0,1\} \qquad (j=1,\dots,n; p\in W_j; q\in H_j). \tag{6.6}$$

3 The objective function (6.2) minimizes the width z at which the strip is used. Equations (6.3) impose that each item is packed in exactly one position. Inequalities (6.4) impose that at most one item occupies any unit square of the strip. Constraints (6.5) set the value of the objective function. Note that, for each item j, definitions (6.6) only consider variables corresponding to feasible positions for the bottom-left corner of the item, so no packed item can exceed the height of the strip.

In order to model problem SR, we add to (6.1) a twin set of variables,

$$y^j_{pq} = \begin{cases} 1 & \text{if item } j \text{ is packed, rotated, with its (new) bottom-left corner at } (p,q); \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

for $j=1,\dots,n$, $p\in\overline{W}_j$, $q\in\overline{H}_j$, where $\overline{W}_j=\{0,1,\dots,W-h_j\}$ and $\overline{H}_j=\{0,1,\dots,H-w_j\}$ denote all positions where the bottom-left corner of the rotated item may be placed. The resulting ILP model for SR has the same objective function as SO, while constraints become

$$\sum_{p\in W_j}\sum_{q\in H_j} x^j_{pq} + \sum_{p\in\overline{W}_j}\sum_{q\in\overline{H}_j} y^j_{pq} = 1 \qquad (j=1,\dots,n) \tag{6.8}$$

$$\sum_{j=1}^{n}\left(\sum_{\substack{p=r-w_j+1\\ p\in W_j}}^{r}\sum_{\substack{q=s-h_j+1\\ q\in H_j}}^{s} x^j_{pq} + \sum_{\substack{p=r-h_j+1\\ p\in\overline{W}_j}}^{r}\sum_{\substack{q=s-w_j+1\\ q\in\overline{H}_j}}^{s} y^j_{pq}\right) \le 1 (r=0,..,W-1; s=0,..,H-1) \tag{6.9}$$

$$\sum_{p\in W_j}\sum_{q\in H_j}(p+w_j)\,x^j_{pq} + \sum_{p\in\overline{W}_j}\sum_{q\in\overline{H}_j}(p+h_j)\,y^j_{pq} \le z \qquad (j=1,\dots,n) \tag{6.10}$$

$$x^j_{pq} \in \{0,1\} \qquad (j=1,\dots,n; p \in W_j; q \in H_j) \qquad (6.11)$$

$$y^j_{pq} \in \{0,1\} \qquad (j=1,\dots,n; p \in \overline{H}_j; q \in \overline{W}_j) \qquad (6.12)$$

5 to impose feasibility with respect to the two possible orientations.

An ILP model for problem BO (pack oriented items in a bin) can be immediately derived from (6.2)-(6.6) by: (i) eliminating constraints (6.5), as definitions (6.6) guarantee that no packed item can exceed the borders of the bin; (ii) replacing objective function (6.2) with

$$\max \sum_{j=1}^{n} w_j\, h_j \Big(\sum_{p \in W_j} \sum_{q \in H_j} x^j_{pq} \Big), \qquad (6.13)$$

that maximizes the packed area; (iii) replacing the '=' sign with '$\leq$' in (6.3), as not all items must be packed.

Finally it is easily seen that the non-oriented bin packing version BR can be modeled, by introducing the twin variables (6.7), as

$$\max \sum_{j=1}^{n} w_j\, h_j \Big(\sum_{p \in W_j} \sum_{q \in H_j} x^j_{pq} + \sum_{p \in \overline{W}_j} \sum_{q \in \overline{H}_j} y^j_{pq} \Big) \qquad (6.14)$$

$$\sum_{p \in W_j} \sum_{q \in H_j} x^j_{pq} + \sum_{p \in \overline{W}_j} \sum_{q \in \overline{H}_j} y^j_{pq} \leq 1 \qquad (j = 1,\dots,n) \qquad (6.15)$$

$$(6.9), (6.10), (6.11), (6.12).$$

As already mentioned, the purpose of this chapter is not to present the state-of-the-art of algorithms for the exact or approximate solution of two-dimensional packing problems. In order to be self-contained, we give however a brief description of two classical and intuitive heuristic algorithms that we used to evaluate the solutions produced by the students who took part in the classroom tests. (The exact solutions were obtained through the exact approaches proposed by Côté et al. [83] and Delorme et al. [100].)

The classical *Bottom-Left* algorithm was introduced by Baker et al. [18] for problem SO, in the (equivalent) version in which the strip is vertical (see Section 6.2). It preliminary sorts the items according to a prefixed policy (non increasing width, or non increasing height, or non increasing area), and packs one item at a time, in the lowest possible position, left justified. Its worst-case performance is 3, i.e., it is guaranteed to produce a vertical strip whose hight does not exceed by more than three times the hight of the optimal solution. The algorithm can be implemented so as to run in $O(n^2)$ time (see Chazelle [61]). In our case (horizontal strip), the item is packed in the leftmost possible position, top justified.

As the classroom tests were performed both on problems SO and SR, we also implemented a simple variation that preliminarily sorts the items according to a given policy (non-increasing max(width,height) or non-increasing area) and, at every iteration, packs the current item in the leftmost-top position: if both orientations are feasible, the item is packed by selecting the smallest side as the width.

Another stream of heuristics (*Best-fit algorithms*, see Burke et al. [51]) first finds, in the current packing, *holes* (empty orthogonal spaces) at which the next item may be packed (initially, the bottom of the strip/bin is the unique hole). It selects the bottommost hole and inserts in it the largest item that fits. If the item width is smaller than that of the hole, it is packed according to a prefixed policy (leftmost, or close to the tallest neighbor, or close to the smallest neighbor). If no item can be packed in the selected hole, the hole is "closed" (filled with empty space). When rotation is allowed, the algorithm considers both orientations when checking if an item fits in a hole, and selects the largest item that fits, selecting the highest one in case of tie.

For both families of algorithms, our approach performs a separate execution for each of the three prefixed policies, and returns the best solution. For the cases with rotation, a so called *tower reduction* post process is executed, that tries to rotate, if possible, the items whose top edges touch the top of the strip, in order to reduce the strip height.

An instance of any problem variant can obviously include a number of identical items. In order to obtain a compact definition and visual rendering, our representation of an instance handles it without duplicating the items, but defining the number of copies of each item *type*.

6.3 Software

In this section, we detail `TwoBinPack`, the open source Scala architecture that was developed to host `TwoBinGame`. Scala is a general purpose programming language that combines ideas from functional programming and object-oriented programming. It runs on the Java platform and its distribution is released under a BSD licence. Full documentation can be found at `http://scala-lang.org/documentation`.

The application we describe is released under the GPLv3 license. A self-contained version is available for free download, as a compressed file, from `http://www.or.deis.unibo.it/staff_pages/martello/Tools/T.html`. Instructions, additional information, and (future) enhanced versions can be found at `http://gianlucacosta.info/TwoBinPack/`. A visual tutorial can be seen online at `https://youtu.be/SS6mJxugyxc`.

The architecture includes three main components:

1. `TwoBinManager`, that manages the problem instances (creation, modification, removal, import, export) and the solutions (import, visualization);
2. `TwoBinGame`, that loads the instances created by `TwoBinManager` and allows the user to interactively solve them;
3. `TwoBinKernel`, a Scala library referenced by the two previous components.

In addition, `TwoBinPack` is based on four modules of the general-purpose library Helios (see `https://www.facebook.com/pages/Helios/206962992779275`): Helios-core, Helios-fx, Helios-jpa, and Helios-reflection. The overall architecture is summarized in Figure 6.1.

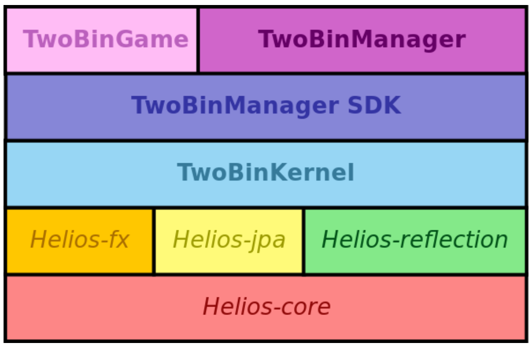


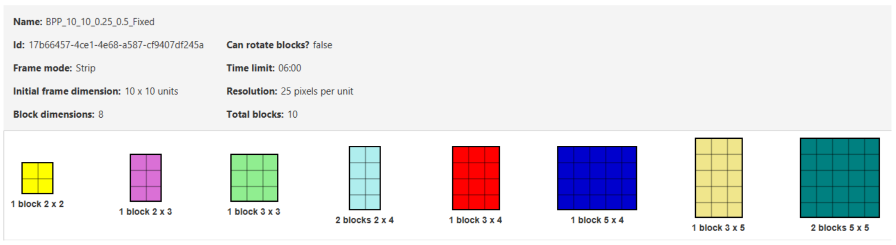
Figure 6.1: The architecture stack.

`TwoBinKernel` is the core of the `TwoBinPack` architecture. It provides the ScalaFX rendering components used to model the two-dimensional packing problems, as well as the tools required to obtain a user-friendly interface in terms of, e.g, dimensions, frames, coordinate system, templates, problem, and user solution. `TwoBinKernel` is also an open source Scala library, released under the GPLv3 license, available for the creation of new applications (see `https://github.com/giancosta86/TwoBinKernel`). In the next sections we provide some details on the two other components of the architecture.

TwoBinManager

`TwoBinManager` is a ScalaFX application designed to manage instances, bundles, and solutions through a local HyperSQL database residing in the user's home directory.

An instance corresponds to a certain problem variant, it allows rotation or not, it has a total number of items (called *blocks* in the application), a number of item types, and the amount of items of each type. In addition it has a time limit assigned to the user to find a solution. Figure 6.2 shows a strip packing instance with 10 items of 8 item types, for which rotation is not allowed and 6 minutes are given to find a solution.


Name: BPP_10_10_0.25_0.5_Fixed

Id: 17b66457-4ce1-4e68-a587-cf9407df245a **Can rotate blocks?** false

Frame mode: Strip **Time limit:** 06:00

Initial frame dimension: 10 x 10 units **Resolution:** 25 pixels per unit

Block dimensions: 8 **Total blocks:** 10

Figure 6.2: A strip packing instance.

An instance can be generated within the program, or it can be read from a standard text file or from a *bundle* file (a set of one or more instances) previously generated by `TwoBinManager`. Some parameters of the instance can be modified in `TwoBinManager` (e.g., the time limit).

TwoBinGame

`TwoBinGame` is a ScalaFX application enabling users to interactively solve two-dimensional packing problems. It reads the bundles created by `TwoBinManager` and returns a file that contains the best solution found by the user for each instance in the bundle and the time required to obtain it. Figure 6.3 shows the solution given by a student for the instance shown in Figure 6.2, that packs the items into a strip of width 14, found in 3 minutes and 17 seconds. Upon reading a bundle file, the user tries to solve, one after the other, all the instances in the bundle. Once the time limit is expired (or if the user decides to pass to the next instance), `TwoBinGame` stores the best solution found for the current instance. When all instances of the bundle have been tried, the user can save the obtained results. `TwoBinGame` provides two different ways for building a solution: usual drag and drop, or a

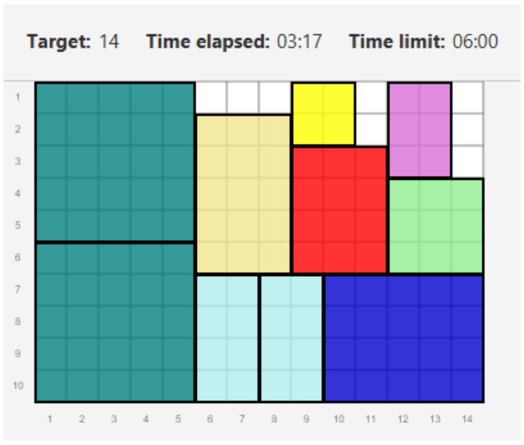
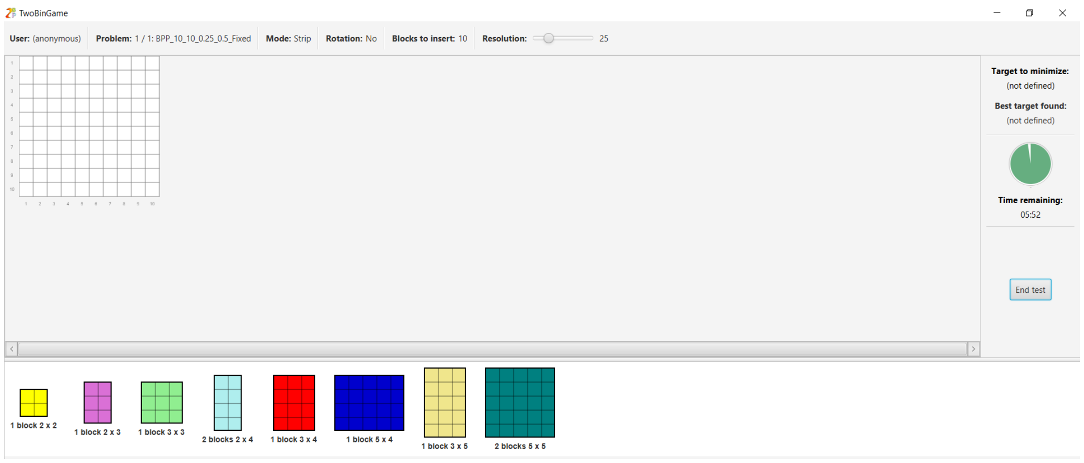


Figure 6.3: A solution for the instance of Figure 6.2.

mouse-wheel and click approach (for a faster interaction).

Figure 6.4 shows the visual interface of `TwoBinGame` for the instance of Figure 6.2.



TwoBinGame

User: (anonymous) Problem: 1 / 1: BPP_10_10_0.25_0.5_Fixed Mode: Strip Rotation: No Blocks to insert: 10 Resolution: 25

Target to minimize: (not defined)

Best target found: (not defined)

Time remaining: 05:52

1 block 2 x 2 1 block 2 x 3 1 block 3 x 3 2 blocks 2 x 4 1 block 3 x 4 1 block 5 x 4 1 block 3 x 5 2 blocks 5 x 5

Figure 6.4: The user view for the instance of Figure 6.2.

6.4 Experiments

We used `TwoBinGame` to perform a series of classroom tests on the SO and SR variants with students of Engineering (Degrees in Management Engineering) of the Universities of

Bologna and of Modena and Reggio Emilia. Classroom tests were optional and competitive. Each student that accepted to participate received a small bonus in her/his final mark. The students achieving the best solutions received a larger bonus. In the next section we describe in details the setup of the tests, whereas in Section 6.4.2 we discuss the results that were obtained.

6.4.1 Setup

We decided to focus on SO and SR random instances having the following characteristics:

1. $n \in \{10, 13, 17, 20\}$;
2. $H \in \{10, 15, 20\}$;
3. w_j and h_j values uniformly randomly distributed in $[H/4, H/2], [1, 2H/3]$;
4. rotation either allowed or forbidden.

For each quadruplet (n, H, range, rotation) one instance was generated, producing in total 48 instances. After a manual check, we removed the instances that could be trivially solved, and generated others with the same parameters. We fairly distributed the 48 instances into 8 bundles of 6 *standard* instances each. For statistical purposes, we added to each bundle an additional instance, having either medium or high difficulty. The one of medium difficulty had 13 items to be packed into a strip of height 15 without rotation, while the difficult one had 20 items to be packed into a strip of height 20 allowing rotation. According to the presumed difficulty, each instance was allowed a time limit of x minutes, with $x \in \{4, 5, 6, 7, 8\}$.

Each student was assigned a bundle. Each test lasted about one hour: 20 initial minutes were used to instruct the students on how to download the software, solve a toy instance, and learn on a standard instance with no time limit. The remaining 40 minutes were used to solve the seven instances in the assigned bundle. At the end of the test, each student sent by email the file containing the best solution obtained for each instance.

Four classroom tests were performed. At the University of Bologna, 65 students of the second year of the Bachelor Degree performed the test in the university lab, while 18 students from the same class performed the test at home using their own PCs. In the latter case, links and instructions on how to download `TwoBinGame`, as well as the bundle number, were communicated to the students by email. At the University of Modena and Reggio Emilia, two tests were performed in the university lab: the former one involved 72 students of the first year of the Master Degree, while the latter involved 58 students of the third year of the Bachelor Degree. Overall, 213 students performed the test: 195 in the lab and 18 online.

The outcome of the tests showed no remarkable difference in the performance of students of different courses so, in the next section, we evaluate the results through aggregate information.

6.4.2 Results

We report in the following the outcome of 201 tests out of the 213 that were performed. The results of the remaining 12 tests were disregarded because either incomplete information was provided via email or the student did not reach a minimum of 5 feasible solutions out of 7. We evaluate in Tables 6.1–6.5 the $201 \times 6 = 1206$ solutions of the six standard instances in the bundles, while in Table 6.6 and Figure 6.5 we comment on the 201 solutions of the additional, more difficult, instances.

The tables aggregate the instances into subgroups, one per line, according to different characteristics. Let z_{opt} be the optimal solution value. The tables provide, for each line, the number of tests performed on the corresponding instances and, respectively for the humans and the heuristics,

- average percentage of optimal solutions (avg. perc. opt.);
- average absolute gap (avg. abs. gap) between solution value and optimal value;
- average relative gap (avg. rel. gap), computed as (solution value - $z_{opt})/z_{opt}$.

The last line of each table provides the overall average values.

Table 6.1 evaluates the solution quality when varying the number n of items. As it could be expected, the students found good quality solutions for instances with a small number of items and worse solutions when this number was larger. A similar behavior cannot be clearly established for the heuristics. Overall, the students beat the heuristics in finding proven optimal solutions (22.5% vs 18%), but they resulted slightly worse in terms of average solution quality (1.83 vs 1.65, and 6.1% vs 5.7%).

Table 6.2 ranks the solutions according to the range of the optimal solution value (strip length). The students performed very well on the 168 tests made on instances for

Table 6.1: Evaluation by varying n

n	# tests	avg. perc. opt.		avg. abs. gap		avg. rel. gap	
		human	heuristic	human	heuristic	human	heuristic
10	304	28%	9.2%	1.35	1.41	7%	7.5%
13	299	25.4%	29.1%	1.62	1.48	5.9%	5%
17	299	19.1%	11%	2.12	2.05	6.4%	6.5%
20	304	17.4%	22.7%	2.23	1.66	5.2%	3.9%
overall	1206	22.5%	18%	1.83	1.65	6.1%	5.7%

which $z_{opt} \in [5; 14]$, optimally solving almost 40% of them. Their performance decreased consistently when the range increased, and no instance with $z_{opt} \geq 55$ could be solved to optimality. A similar behavior can be noticed for the heuristics, confirming that the range of the optimal solution value has considerable impact on the difficulty of an instance. It is interesting to observe that the students clearly beat the heuristics in finding optimal solutions, probably because the visualization gives good opportunities for a clever post-processing of near-optimal solutions.

Table 6.2: Evaluation by varying the optimum solution range

z_{opt} range	# tests	avg. perc. opt.		avg. abs. gap		avg. rel. gap	
		human	heuristic	human	heuristic	human	heuristic
$[5; 14]$	168	39.3%	35.1%	0.73	0.74	5.6%	5.7%
$[15; 24]$	289	32.2%	23.5%	1.15	1.03	5.9%	5.5%
$[25; 34]$	462	20.8%	16.7%	1.58	1.73	5.5%	6%
$[35; 44]$	124	9.7%	10.5%	2.96	2.19	7.5%	5.3%
$[45; 54]$	90	4.4%	0%	4.16	3.11	8.3%	6.2%
$[55; 64]$	73	0%	0%	4.25	3.05	7.3%	5.3%
overall	1206	22.5%	18%	1.83	1.65	6.1%	5.7%

Table 6.3 takes the variation of the strip height into account. Both students and heuristics found better solutions for instances with small strip height. The strong impact of the strip height on the instance difficulty is also shown by the increase in the absolute and relative gaps when H increases. The reason for this behavior is probably that a small strip height gives few possibilities for the vertical placement of an item.

Table 6.4 shows the results for the two ranges adopted for the items dimensions: in the former range the items have comparable dimensions, while in the latter they are quite

Table 6.3: Evaluation by varying the strip height H

H	# tests	avg. perc. opt.		avg. abs. gap		avg. rel. gap	
		human	heuristic	human	heuristic	human	heuristic
10	402	39.8%	39.3%	0.78	0.66	4.2%	3.8%
15	402	21.4%	14.7%	1.56	1.64	6.2%	6.7%
20	402	6.2%	0%	3.22	2.65	8%	6.7%
overall	1206	22.5%	18%	1.83	1.65	6.1%	5.7%

dissimilar from each other. This parameter only marginally affected the performance of the students who, however, performed slightly better for the wider range. The fact that such range appears to produce easier instances is confirmed by the heuristics, whose performance is clearly better for it.

Table 6.4: Evaluation by varying the item dimensions' range

item range	# tests	avg. perc. opt.		avg. abs. gap		avg. rel. gap	
		human	heuristic	human	heuristic	human	heuristic
$[H/4; H/2]$	603	21.4%	14.6%	1.97	2.01	6.2%	6.5%
$[1; 2H/3]$	603	23.5%	21.4%	1.69	1.29	6%	4.9%
overall	1206	22.5%	18%	1.83	1.65	6.1%	5.7%

Table 6.5 concerns the possibility of rotating the items (by 90 degrees). On average, allowing rotation helps the students in finding optimal or good-quality solutions. A similar behavior cannot be observed for the heuristics: when rotation is allowed they find less optimal solutions, but at the same time they provide better absolute and relative gaps. This could be produced by the tower-reduction post processing: when rotation is allowed, the algorithm repositions some long and thin items packed at the top of the strip, which helps in reducing the gap but not in reaching optimality.

Table 6.6 presents results for the two additional instances that were included in the bundles. The instance having a supposed medium difficulty was attempted 71 times by the students, whereas the difficult one was attempted 130 times. Just one of these attempts resulted in an optimal solution (for the medium difficulty instance). The table shows the impact of the allowed time limit (from 4 to 7 minutes). For the medium difficulty instance, the time appears to be a relevant factor to decrease the gaps. For the difficult instance,

Table 6.5: Evaluation by allowing/disregarding rotation

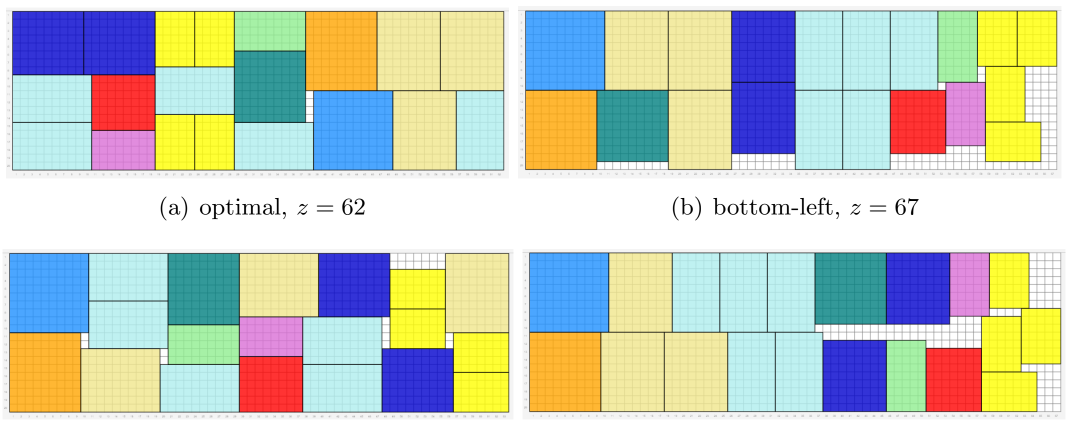
rotation	# tests	avg. perc. opt.		avg. abs. gap		avg. rel. gap	
		human	heuristic	human	heuristic	human	heuristic
not allowed	603	19.2%	18.4%	1.85	1.8	6.8%	6.6%
allowed	603	25.7%	17.6%	1.81	1.51	5.4%	4.9%
overall	1206	22.5%	18%	1.83	1.65	6.1%	5.7%

instead, it does not allow to produce better solutions, probably because the instance was "too difficult".

Table 6.6: Solution quality for the difficult shared instances by varying time

time limit	Medium instance			Difficult instance		
	# tests	avg. abs. gap	avg. rel. gap	# tests	avg. abs. gap	avg. rel. gap
4	16	2.9	9.8%	33	5.0	7.3%
5	19	2.5	8.7%	32	3.7	5.5%
6	15	1.9	6.6%	31	4.9	7.1%
7	21	1.9	6.7%	34	4.9	7.2%
overall	71	2.3	7.9%	130	4.6	6.8%

The medium difficulty instance had $z_{opt} = 26$, which was found just by one student (in 4:44 minutes), whereas the heuristics found a strip of length 28. The difficult instance had $z_{opt} = 62$: the best student found a solution with $z = 63$ in 7:02 minutes, while the heuristic solutions (both the one found by bottom-left and the one found by best-fit) had $z = 67$. The four solutions obtained for this instance are shown in Figure 6.5, which gives some insight in the packing processes. The best student solution is very good: some small waste portions of the strip are accepted even at an early stage of the packing, but this results in a small final waste of just 24 units out of 1240. The possibility of rotating the items has been conveniently exploited (see the light blue items). The heuristic solutions are instead quite bad. Best-fit managed to produce a very dense packing at the beginning of the strip, but this resulted in a large waste towards the end. Bottom-left had a similar behavior. Note that both heuristics left for the final portion of the strip the yellow, red, and purple items. In particular, the yellow items appear to be difficult to pack, and the optimal solution is the only one that managed to conveniently pack them together with a light blue item.



(a) optimal, $z = 62$

(b) bottom-left, $z = 67$

(c) best human, $z = 63$

(d) best-fit, $z = 67$

Figure 6.5: Solutions of the difficult instance

6.5 Conclusions

We have presented `TwoBinGame`, an open source visual application for interactively solving two-dimensional packing problems. The application was developed for guiding engineering students to understand how difficult combinatorial optimization problems can be solved through heuristic approaches. Although mainly conceived for didactical purposes, the application can be conveniently used in professional contexts. Practitioners can find it useful to build good quality solutions for real world two-dimensional packing instances arising, e.g., in the glass, steel or paper industry. On the other hand, `TwoBinGame` can help developers in the design of effective algorithms for the solution of these problems. We have additionally reported on classroom experiments that proved to be useful in testing students' skills versus exact and heuristic approaches.

Chapter 7

Mathematical Models and Decomposition Algorithms for Makespan Minimization in Plastic Rolls Production

[1]

In this chapter, we study an optimization problem that originates from the packaging industry, and in particular in the process of blown film extrusion, where a plastic film is used to produce rolls of different dimensions and colors. The film can be cut along its width, thus producing multiple rolls in parallel, and set-up times must be considered when changing from one color to another.

The optimization problem that we face is to produce a given set of rolls on a number of identical parallel machines by minimizing the makespan. The problem combines together cutting and scheduling decisions, and is of high complexity. For its solution we propose mathematical models and heuristic algorithms that involve a non-trivial decomposition method. By means of extensive computational experiments we show that proven optimality can be achieved only on small instances, whereas for larger instances good quality solutions can be obtained especially by the use of an iterated local search algorithm.

Keywords: Plastic Rolls Production, Blown Film Extrusion, Optimization, Mixed Integer Linear Programming, Iterated Local Search.

[1]The results of this chapter appears in: V. Nesello, M. Delorme, M. Iori, and A. Subramanian, Mathematical Models and Decomposition Algorithms for Makespan Minimization in Plastic Rolls Production, *Journal of the Operational Research Society*, to appear [223].

7.1 Introduction

In the packaging industry, plastic films are commonly obtained by blown film extrusion. A tube of polymer is inflated to form a thin film bubble. The bubble is collapsed to obtain a flat film, which is then used to produce the required items. A common process is to first use the film to produce plastic rolls, and then use the rolls later on the production process to obtain smaller items. The rolls have a certain width that depends on the extruder machine at use, and a (possibly very large) length that depends from the adopted operating time. To obtain the desired roll dimensions, the film can be cut both along its width and along its length. The cutting along the width is obtained by one or more devices called *slits* and allows to produce multiple rolls in parallel on the same machine. We refer to, e.g., Cantor [53] for a wide description of the blown film extrusion characteristics.

In this chapter we focus on a problem that originates from an industry producing plastic bags for the South American market. Multiple identical extruded machines, each equipped with a limited number of slits, are available to produce in parallel a set of required plastic rolls. Each roll has a specific color, thickness, width, and length. The extruder machines make use of raw polymer, thus a colorant is required for the plastic bags to have the desired color. This implies that the rolls processed at the same time on the same machine have the same color and thickness, although they might have different width and length. The width of the film can be adjusted at any time so as to waste neither time nor plastic. A setup time is incurred when switching from one color to another, while changing the thickness is instantaneous. The aim is to produce the required set of rolls by minimizing the *makespan*, i.e., the completion time of the last item.

Note that the cutting process can be seen as a *three-stage two-dimensional guillotine cutting* (see, e.g., Silva et al. [259]): the first-stage of vertical cuts separates *blocks* of rolls having different colors; the second-stage of horizontal cuts separates *shelves* of rolls produced in parallel; the third-stage of vertical cuts separates a roll from the subsequent one thus obtaining the required products.

To better understand this complex problem, a simple example involving 12 items of 3 different colors and a unique thickness is depicted in Figure 7.1. The numbers on the arcs represent the setup times between pairs of colors. The widths of the items are proportional to the vertical dimensions of the depicted rectangles, whereas lengths are proportional to the horizontal dimensions (roll lengths have been scaled down to better fit the graphical representation). An optimal solution with two machines, both equipped with one slit and having maximum width 5, is provided in Figure 7.2. Machine 1 processes a first block containing only item 1 for 5 time units, then incurs in a setup of 4 units, and finally processes a second block containing items from 5 to 8 for 11 time units. The slit is used only in the second block to separate the shelf containing item 8 from the shelf containing the remaining items. Note that no plastic is wasted during the processing of the second block. The width of the film can indeed be reduced by lowering down by one unit the upper part when processing items 8 and 6, and by two units when processing 8 and 7. Similarly, the bottom can be lift up by one unit when processing only 7. A similar process is performed by machine 2, that processes two blocks, both made by two shelves. The slit is used in the first block to separate item 4 from 2 and 3, and in the second block to separate items 9 and 10 from 11 and 12. The resulting makespan is of 20 time units.

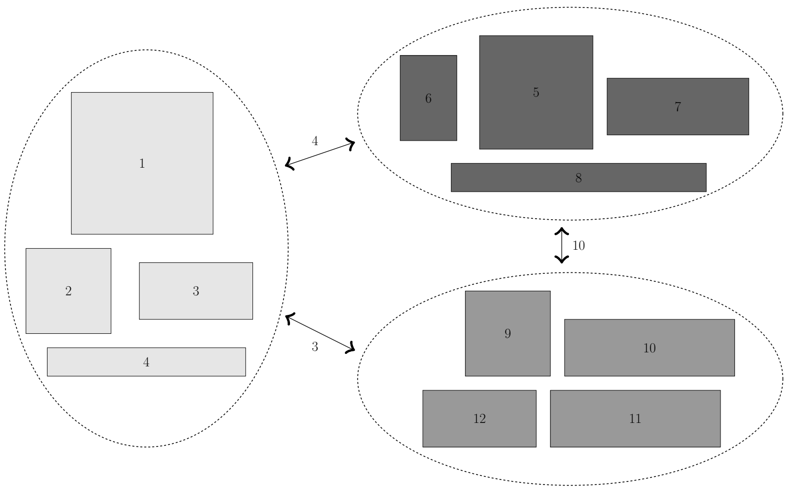


Figure 7.1: A simple PRPP instance.

The resulting optimization problem, that we hereafter call *plastic rolls production problem* (PRPP), embeds a cutting component (the 3-stage two-dimensional guillotine cutting) into a classical scheduling problem of makespan minimization with setup times on parallel machines. Problems arising in the so-called *cutting and packing* research area are frequently encountered in practical industrial applications (see, e.g., Kallrath et al. [165] and the special issue edited by Bennell et al. [36]). Cutting and packing problems are frequently combined with problems from other optimization areas so as to better model real-world situations. Iori et al. [161] combined two-dimensional loading problem and vehicle-routing, Gramani et al. [140] and Silva et al. [258] integrated lot-sizing with cutting stock prob-

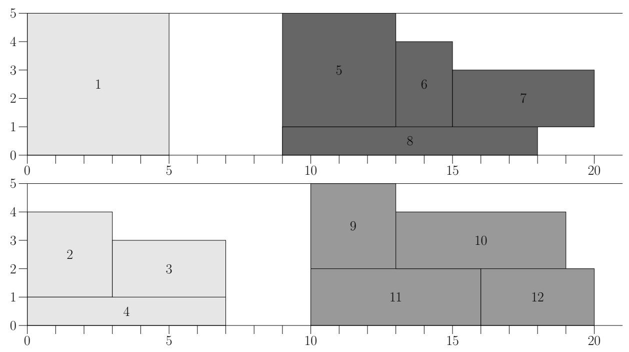


Figure 7.2: An optimal solution for the instance in Figure 7.1.

lems, and Hu et al. [159] considered two-dimensional spatial resource constraints in project scheduling, just to cite some examples.

To the best of our knowledge, the problem that is most similar to the PRPP is the three-stage two-dimensional guillotine cutting problem studied by Vanderbeck [285]. In this problem, rectangular bins of fixed width and length are cut into blocks, then blocks are cut into shelves, and shelves are cut into items. Fixed setup times occur between blocks, and the objective is mainly waste minimization. The problem was solved by a decomposition based on the recursive use of a column generation approach. The PRPP is however different from the problem in [285] because it has a limit on the number of shelves per block, it has sequence-dependent setup times, it involves rolls instead of bins of fixed length, and aims at minimizing makespan instead of waste material.

The aim of this charpter is to formally introduce the PRPP, present exact and heuristic methods for its optimization, and perform an extensive computational evaluation of the proposed solution strategies. In details, we first model the PRPP using a compact *mixed integer linear programming* (MILP) formulation, i.e., a formulation that has a polynomial number of variables and constraints. We then decompose the problem by separating the cutting component from the scheduling one, and use this to derive valid lower and upper bounds on the optimal makespan. Notably, these bounds are obtained by solving three different MILP models, one involving a non-trivial pseudo-polynomial *arc-flow* formulation, another one requiring a tailored branch-and-cut implementation and the last one based on the generalized assignment problem. These techniques work well for small- and medium-size instances, thus, to efficiently address large-size instances, we developed a metaheuristic based on an *iterated local search* framework.

The remainder of the chapter is organized as follows: (i) the PRPP is formally introduced and the relevant literature is discussed; (ii) the compact MILP formulation is presented; (iii) the decomposition approach and the lower bounds are explained; (iv) the upper bounding methods are described; (v) the proposed techniques are computationally evaluated; and (vi) conclusions are drawn in the last section.

7.2 Problem Description

The PRPP requires to produce n plastic rolls, called *items* for short in the following, on m identical parallel machines. Each item j has width w_j, length l_j, and belongs to a class γ_j, for $j = 1, \dots, n$. Without loss of generality we suppose that the processing time of an item j is equal to its length l_j. Two items belong to the same class if they have the same color and thickness, thus allowing both of them to be processed at the same machine simultaneously. A setup cost s_{ij} occurs when a machine processes an item j after having processed an item i. A machine cannot process items during setup. Each machine has a maximum width W and is equipped with σ slits. A *block* is a subset of items of the same class, whereas a *shelf* is a sequence of items packed at the same width in a block. A machine can process items in parallel by using a three-stage two-dimensional guillotine cutting process: the first-stage vertical cuts separate blocks, the second-stage horizontal cuts separate shelves, and the third-stage vertical cuts separate items. The width of a shelf is given by the wider item in that shelf. The total width of the shelves produced in parallel on a machine cannot exceed W, whereas their total number cannot exceed $\sigma + 1$. The processing time of a shelf is given by the sum of the lengths of the items in that shelf, and the processing time of a block is the maximum processing time of the shelves in that block. The objective of the PRPP is to feasibly schedule the items into the machines by minimizing the makespan.

We consider that a fixed identical setup cost occurs on each machine at the beginning of the activities. As this cost is the same for all the machines it has no impact on the optimal solution, thus we simply disregard it from our study. Moreover, our models and algorithms could be easily adapted to include a starting setup cost that depends from an initial configuration on each machine.

The PRPP is composed by a two-dimensional cutting component (creating the blocks) and a scheduling component (assigning the blocks to the machines with sequence-dependent setup times). Because of its nature, it generalizes a number of quite different combinatorial optimization problems, as noticed in Table 7.1. If a single machine with 0 slits is available and all items have the same color, then the problem is trivial because there are no setup costs and items must be processed one after the other, so the makespan is simply the sum of all processing times. If a single machine with 0 slits is available and the items have different colors, then the PRPP reduces to the well-known *traveling salesman problem* (TSP), see, e.g., Applegate et al. [12], because items cannot be produced in parallel and thus minimizing the makespan is equivalent to finding the permutation of the items that leads to the lowest setup cost. If all items have the same color, and multiple machines are available but none of them has slits, then the PRPP is equivalent to the *identical parallel machine scheduling problem* ($\mathrm{P}||\mathrm{C}_{\max}$), see, e.g., DellAmico et al. [94]. In this case, indeed, no setup cost is paid, items cannot be produced in parallel on a machine, and thus the problem is to distribute the items among the machines to obtain the best makespan. If all items have the same color and just one machine is available, then the PRPP becomes a *three-stage cardinality-constrained two-dimensional strip packing problem* (3SC2SPP), where the cardinality limit is imposed on the number of shelves. The strip packing problem (SPP) is to pack a set of rectangles into a strip of fixed width and minimum length, without overlapping and without rotation (see, e.g., Côté et al. [83]). Several SPP variants including multi-stage guillotine cuts have been studied in the literature (see, e.g., Silva et al. [259] and Mrad [221] among others), but, to the best of our knowledge, the 3SC2SPP is a new problem.

With the exception of the trivial case, all the discussed problem variants are NP-hard, so the same holds for the PRPP.

Table 7.1: Relevant PRPP variants under different input parameters.

colors	machines	slits	problem
1	1	0	trivial
2 or more	1	0	traveling salesman problem
1	2 or more	0	identical parallel machines scheduling problem
1	1	1 or more	three-stage two-dim. card.-constr. SPP
2 or more	2 or more	1 or more	plastic rolls production problem (this chapter)

7.3 A Compact Mathematical Formulation

In this section we propose a compact model for the PRPP that generalizes the one presented by Lodi and Monaci [198] for the two-stage two-dimensional knapsack problem by considering three-stage cutting and setup costs. In the following, let items be sorted by non-increasing width, breaking ties by non-increasing length. Let us say that an item *initializes* a shelf if it is the lowest-index item in that shelf, and that it initializes a block if it is the lowest-index item in that block.

Now, let us first model the cutting part of the PRPP by using the decision variable

- $x_{jik} = \begin{cases} 1 & \text{if item } j \text{ is in shelf } i \text{ of block } k; \\ 0 & \text{otherwise;} \end{cases}$ for $1 \leq k \leq i \leq j \leq n$, $\gamma_k = \gamma_i = \gamma_j$.

Variable x has a threefold meaning: if $x_{kkk} = 1$, then item k initializes block k, meaning that item k has the greatest width of all items in block k; if $x_{iik} = 1$, $k < i$, then item i initializes shelf i in the block initialized by a previous item k, meaning that item i has the greatest width of all items in shelf i, but there is at least one item with greater width that initializes another shelf in block k; if $x_{jik} = 1$, $k < i < j$, then item j is packed into the shelf initialized by a previous item i in the block initialized by a previous item k, meaning that item j is placed in a shelf containing at least one item of greater width. This is necessary to enforce items with lower width to be packed to the right of the larger ones, therefore meeting the three-stage two-dimensional guillotine cutting configuration of the problem. For example, in Figure 7.2, the block containing items 2, 3, and 4 is the block number 2, because the item 2 is the one with greatest width. Hence, we say that item 2 initializes the block and shelf of the same number, and we have $x_{222} = 1$. Item 3 is packed on the shelf initialized by item 2, so $x_{322} = 1$, and item 4 initializes the shelf number 4, so $x_{442} = 1$. The other variables that take value 1 are x_{111}, x_{555}, x_{655}, x_{755}, x_{885}, x_{999}, $x_{10,9,9}$, $x_{11,11,9}$, and $x_{12,11,9}$.

Let us call for short "block k" the block initialized by item k and "shelf i" the shelf initialized by item i. To model the scheduling part of the PRPP let us introduce two dummy items, 0 and $n+1$, having 0 length and no setup costs from and to other items. Item 0 represents the beginning of the activities and item $n+1$ represents the end. Let us also denote $N = \{1, 2, \ldots, n\}$ and $N' = \{0, 1, \ldots, n, n+1\}$. We make use of the following decision variables:

- $\xi_{hk} = \begin{cases} 1 & \text{if block } h \text{ is followed by block } k; \\ 0 & \text{otherwise;} \end{cases}$ for $h, k \in N'$, $h \neq n+1$, $k \neq 0$, $h \neq k$;
- α_k : setup required for block k, for $k \in N$;
- β_k : processing time of block k, for $k \in N$;
- ϑ_{hk} : time flow between blocks h and k, for $h, k \in N'$, $h \neq n+1$, $k \neq 0$, $h \neq k$;
- z : makespan.

In practice, the ξ_{hk} variables keep track of the sequences of blocks in each machine, whereas the other variables are used to compute the completion times of the jobs and the makespan. For the example depicted in Figure 7.2, we have: $\beta_1 = 5$, $\beta_2 = 7$, $\beta_5 = 11$, $\beta_9 = 10$, $\xi_{01} = 1$, $\xi_{15} = 1$, $\xi_{5,13} = 1$, $\xi_{02} = 1$, $\xi_{29} = 1$, $\xi_{9,13} = 1$, $\alpha_5 = 4$, $\alpha_9 = 3$, $\vartheta_{15} = 5$, $\vartheta_{5,13} = 20$, $\vartheta_{29} = 7$, $\vartheta_{9,13} = 20$, and $z = 20$.

The PRPP can then be formulated as the following MILP model:

$$\min \quad z \tag{7.1}$$

subject to

$$\sum_{i=1}^{j} \sum_{k=1}^{i} x_{jik} = 1 \qquad j \in N \tag{7.2}$$

$$\sum_{i=k}^{n} w_i x_{iik} \leq W x_{kkk} \qquad k \in N \tag{7.3}$$

$$\sum_{i=k}^{n} x_{iik} \leq \sigma + 1 \qquad k \in N \tag{7.4}$$

$$x_{jik} \leq x_{iik} \qquad j, i, k \in N, k \leq i \leq j \tag{7.5}$$

$$\sum_{h \in N' \setminus \{k, n+1\}} \xi_{hk} = x_{kkk} \qquad k \in N \tag{7.6}$$

$$\alpha_k \geq \sum_{h \in N' \setminus \{k, n+1\}} s_{hk} \xi_{hk} \qquad k \in N \tag{7.7}$$

$$\beta_k \geq \sum_{j=i}^{n} l_j x_{jik} \qquad i, k \in N, k \leq i \tag{7.8}$$

$$\sum_{h\in N'\setminus\{0,k\}} \xi_{kh} - \sum_{h\in N'\setminus\{k,n+1\}} \xi_{hk} = \begin{cases} m & \text{if } k=0; \\ -m & \text{if } k=n+1; \\ 0 & \text{otherwise} \end{cases} \qquad k \in N' \tag{7.9}$$

$$\sum_{h\in N'\setminus\{0,k\}} \vartheta_{kh} - \sum_{h\in N'\setminus\{k,n+1\}} \vartheta_{hk} = \begin{cases} 0 & \text{if } k=0; \\ \alpha_k+\beta_k & \text{otherwise} \end{cases} \qquad k \in N'\setminus\{n+1\} \tag{7.10}$$

$$z \geq \vartheta_{h,n+1} \qquad h \in N'\setminus\{n+1\} \tag{7.11}$$

$$\alpha_k, \beta_k \geq 0 \qquad k \in N \tag{7.12}$$

$$0 \leq \vartheta_{hk} \leq U\xi_{hk} \qquad h,k \in N', h \neq n+1, k \neq 0, h \neq k \tag{7.13}$$

$$\xi_{hk} \in \{0,1\} \qquad h,k \in N', h \neq n+1, k \neq 0, h \neq k \tag{7.14}$$

$$x_{jik} \in \{0,1\} \qquad j,i,k \in N, k \leq i \leq j, \gamma_k = \gamma_i = \gamma_j \tag{7.15}$$

The objective function (7.1) minimizes the makespan. Constraints (7.2) ensure that every item is scheduled. Constraints (7.3) state that the maximum width of each block is not exceeded. Constraints (7.4) impose that the number of shelves in a block is not greater than the maximum number of slits plus one. Constraints (7.5) state that an item j can assigned to a shelf i only if the shelf has been initialized by i. Constraints (7.6) state that if a block k is used, then exactly one ξ_{hk} variable incoming into k should be used. Constraints (7.7) and (7.8) compute, respectively, the setup time and the processing time required for each block k, if any. Note that the processing time of the block is given by the maximum of the sums of the processing times on each shelf. Constraints (7.9) ensure that exactly m sequences are created and that these sequences are connected. Constraints (7.10) use the variables ϑ_{hk} as a commodity to compute the increasing time along the sequences. This has the effect of disregarding subtours, and also allows to compute, through constraint (7.11), the value of the makespan. Finally, constraints (7.12)-(7.15) impose the bounds on the variables, with U being a valid upper bound value on the makespan.

7.4 Lower Bounds based on a Decomposition Method

Model (7.1)-(7.15) has the merit of unambiguously describe the PRPP, but, as later shown, it has a poor computational performance. In this section and in the following one we study a natural decomposition of the PRPP into its two main components, namely, cutting and scheduling, and show how this can lead to the computation of lower and upper bounds on the optimal solution value. Figure 7.3 presents a flowchart of our decomposition. The *cutting component* (CC) solves a pure cutting problem, producing a set of possible cutting patterns and a set of selected blocks. The patterns are passed to a *scheduling component* (SC), that computes a color sequence for a pure scheduling problem and obtains a valid lower bound. Two algorithms, namely, an iterated local search and an approach based on a generalized assignment problem, use the selected blocks (the latter also uses the color sequence) to produce a feasible solution of good quality.

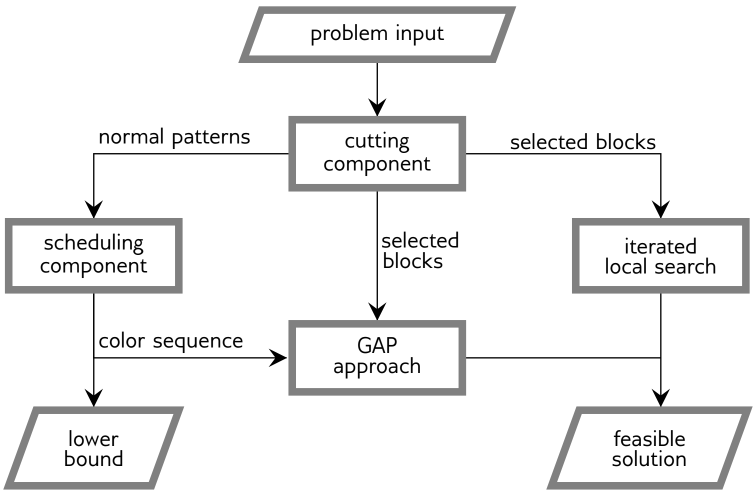


Figure 7.3: A flowchart of the complete algorithm.

7.4.1 Cutting component (CC)

Let S be the set of all item classes, and $N_s = \{j \in N : \gamma_j = s\}$ be the subset of items whose class is s, for $s \in S$. As described in the problem description Section, the problem of producing all items belonging to the same class from a film of minimum length is the 3SC2SPP. In this section we propose a method to solve the 3SC2SPP that is based on the arc-flow formulation introduced by Valério de Carvalho [278] for the one-dimensional cutting stock problem, and later extended to the two-stage two-dimensional case by Macedo et al. [206].

We need some additional notation to describe our formulation. First of all, let us define, for each $s \in S$, m_s^* as the number of different item widths and $\mathcal{W}_s^* = \{w_1^*, w_2^*, \dots, w_{m_s^*}^*\}$ the corresponding width set. Additionally, we compute the set $\mathcal{L}_s$ of all possible combinations of item lengths as

$$\mathcal{L}_s = \left\{ x = \sum\nolimits_{j \in T} l_j : 0 \leq x \leq \Lambda_s, T \subseteq N_s \right\}, \tag{7.16}$$

with Λ_s being an upper bound on the maximum length of a block of items of class s. Using the definitions by Herz [153] and Christofides and Whitlock [64], $\mathcal{L}_s$ can be either called the set of canonical dissections or of normal patterns. Here we use the term normal patterns. The computation of $\mathcal{L}_s$ may be obtained by invoking a standard dynamic programming procedure. We use $\mathcal{L}_s$ to determine the possible positions of the first-stage cuts.

To model the second-stage cuts, we make use of a graph $G'_s = (V'_s, A'_s)$. The vertex set is $V'_s = \{(a,b)\colon a = 0,1,\dots,W; b = 0,1,\dots,\sigma+1\}$. The arc set A'_s is composed by so-called *item arcs* and *loss arcs*: items arcs represent a second-stage cut corresponding to an item and their set is $\{((d,u),(e,u+1))\colon 0 \leq u \leq \sigma; 0 \leq d < e \leq W \text{ and } e - d \in \mathcal{W}_s^*\}$; loss arcs represent unused portions of the film and their set is $\{((a,b),(W,\sigma+1))\colon (a,b) \in V'_s\}$. In addition, let $A'_{j^*s} = \{a = ((d,u),(e,v)) \in A'_s : e - d = w_{j^*}^*\}$, for $w_{j^*}^* \in \mathcal{W}_s^*$. Let also $\delta^+(e,u)$, respectively $\delta^-(e,u)$, be the subset of arcs of A'_s that leaves, respectively enters, a node (e,u).

Third-stage cuts induced on a shelf by a width $w_{j^*}^* \in \mathcal{W}_s^*$, for $j^* = 1,\dots,m_s^*$, are modeled by the use of a multi-graph $G''_{j^*s} = (V''_s, A''_{j^*s})$, where the vertex set is $V''_s = \{0,1,\dots,\Lambda_s\}$ and the arc set is composed by a subset of item arcs and two subsets of loss arcs as $A''_{j^*s} = \{(k,d,e)\colon 0 \leq d < e \leq \Lambda_s \text{ and } \exists\ k \in N_s\colon e - d = l_k \text{ and } w_k \leq w_{j^*}^*\} \cup \{(0,d,\Lambda_s) : d = 0,1,\dots,\Lambda_s - 1\} \cup \{(0,d,d+1) : d = 0,1,\dots,\Lambda_s - 1\}$. Moreover, for any $j^* = 1,\dots,m_s^*$ we define $\delta_{j^*}^+(e)$, respectively $\delta_{j^*}^-(e)$, as the subset of arcs of A''_{j^*s} that leaves, respectively enters, a node e.

Let us introduce the following decision variables:

- φ_p = number of times a block of length $p \in \mathcal{L}_s$ is chosen (first-stage vertical cuts);
- φ'_{pa} = number of times arc $a \in A'_s$ is chosen as a second-stage horizontal cut on a block of length $p \in \mathcal{L}_s$;
- φ''_{j^*a} = number of times arc $a \in A''_{j^*s}$ is used as a third-stage vertical cut on a shelf of width $w_{j^*}^* \in \mathcal{W}_s^*$.

The film of minimum length required to produce all items of class $s \in S$ can be obtained by solving the following MILP model:

$$\min \quad z_s^{CC} = \sum_{p \in \mathcal{L}_s} p\varphi_p \tag{7.17}$$

subject to

$$\sum_{a \in \delta^+(e,u)} \varphi'_{pa} - \sum_{a \in \delta^-(e,u)} \varphi'_{pa} = \begin{cases} \varphi_p & \text{if } (e,u) = (0,0); \\ -\varphi_p & \text{if } (e,u) = (W, \sigma+1); \\ 0 & \text{otherwise,} \end{cases} \qquad (e,u) \in V'_s, p \in \mathcal{L}_s \tag{7.18}$$

$$\sum_{a \in \delta^+_{j^*}(e)} \varphi''_{j^*a} - \sum_{a \in \delta^-_{j^*}(e)} \varphi''_{j^*a} = \begin{cases} \sum\limits_{p \in \mathcal{L}} \sum\limits_{a \in A'_{j^*s}} \varphi'_{pa} & \text{if } e = 0; \\ -\sum\limits_{p \in \mathcal{L}} \sum\limits_{a \in A'_{j^*s}} \varphi'_{pa} & \text{if } e = \Lambda_s; \\ 0 & \text{otherwise,} \end{cases} \qquad j^* = 1, \ldots, m_s^* \tag{7.19}$$

$$\sum_{j^*=1,\ldots,m_s^*, w_{j^*}^* \geq w_k} \sum_{a \in A''_{j^*s}} \varphi''_{j^*a} = 1 \qquad k \in N_s \tag{7.20}$$

$$\varphi''_{j^*(0p\Lambda_s)} = \sum_{a \in A'_{j^*s}} \varphi'_{pa} \qquad j^* = 1, \ldots, m_s^*, \; p \in \mathcal{L}_s \tag{7.21}$$

$$\varphi_p \geq 0, \text{integer} \qquad p \in \mathcal{L}_s \tag{7.22}$$

$$\varphi'_{pa} \geq 0, \text{integer} \qquad p \in \mathcal{L}_s, a \in A'_s \tag{7.23}$$

$$\varphi''_{j^*a} \geq 0, \text{integer} \qquad j^* = 1, \ldots, m_s^*, \; a \in A''_{j^*s} \tag{7.24}$$

The objective function (7.17) minimizes the total used length. Constraints (7.18) impose flow conservation among the φ' variables, and also link together φ' with φ by stating that shelves can be created only in those blocks p that have a positive φ_p value. Similarly, constraints (7.19) ensure flow conservation among the third-stage cuts and allow to produce items only in shelves that have been created by second-stage cuts. Constraints (7.20) ensure that all items are produced, while constraints (7.21) force an empty space from p to Λ_s for shelves produced in block p.

For example, to model the solution depicted in the left-most block of the bottom machine in Figure 7.2, supposing $\Lambda_s = 9$ and $\sigma = 3$, the following variables would take value 1: φ_7, $\varphi'_{7((0,0),(1,1))}$, $\varphi'_{7((1,1),(4,2))}$, $\varphi'_{7((4,2),(5,4))}$, $\varphi''_{1(4,0,7)}$, $\varphi''_{1(0,7,9)}$, $\varphi''_{4(2,0,3)}$, $\varphi''_{4(3,3,7)}$, and $\varphi''_{4(0,7,9)}$. For the right-most block in the same machine, the following variables would instead take value 1: φ_{10}, $\varphi'_{10((0,0),(2,1))}$, $\varphi'_{10((2,1),(5,2))}$, $\varphi'_{10((5,2),(5,4))}$, $\varphi''_{2(11,0,6)}$, $\varphi''_{2(12,6,10)}$, $\varphi''_{2(0,10,19)}$, $\varphi''_{3(9,0,3)}$, $\varphi''_{3(10,3,9)}$, $\varphi''_{3(0,9,10)}$, and $\varphi''_{3(0,10,19)}$.

Model (7.17)–(7.24) may have a slow convergence to an optimal solution because it may contain a large number of variables: $O(\Lambda_s)$ for the first set of cuts, $O(\Lambda_s m^*_s \sigma)$ for the second, and $O(\Lambda_s m^*_s n)$ for the third. We use a heuristic to limit the value of Λ_s and some preprocessing techniques to improve its computational performance.

In terms of preprocessing, we adopted the two following techniques:

- for each item $j \in N_s$ we compute, through dynamic programming, the maximum width $w'_j \leq W - w_j$ that can be taken by a subset of items packed side by side with j. If $w_j + w'_j < W$, then the width of item j is increased to $w_j = W - w'_j$.
- let p be the item with smallest width and q the item with second smallest width. If there is an item j ($j \neq p \neq q$), such that, $w_j + w_p \leq W$, $w_j + w_q > W$, and $l_j \geq l_p$, then we pack items j and p alone in a single block of length l_j.

A consequence of the first preprocessing is that if $w'_j = 0$ (that is, no item can be packed side by side with j) then w_j is set to W and j is packed alone in a block of length l_j.

Limiting the value taken by Λ_s may decrease consistently the number of variables. We pursue this by means of a two-step algorithm. By remarking that any upper bound for z^{CC}_s is also a valid upper bound for Λ_s, we first solve model (7.17)–(7.24) heuristically, by limiting Λ_s to a small value (in our implementation we chose $\Lambda_s = 1.5 \times \max_{j \in N_s}\{l_j\}$). If the solution obtained, say, $\bar{z}^{CC}_s$, satisfies $\bar{z}^{CC}_s \leq \Lambda_s$, then we terminate with a proof of optimality. If instead $\bar{z}^{CC}_s > \Lambda_s$, we set $\Lambda_s = \bar{z}^{CC}_s$ and solve the model once more. A solution obtained for a given Λ_s can be easily mapped into a solution for another $\Lambda'_s > \Lambda_s$, thus, the solution obtained at the end of first step is given as a “warm start” to the solver at the beginning of the second step.

7.4.2 Scheduling component (SC)

The second component of our decomposition approach is a scheduling problem that takes as input the information provided by solution of the cutting component. Recall that S is the set of item classes (defined by thickness and color). Let us now define by C the set of colors, and by $S_c \subseteq S$ the subset of classes whose color is c, for $c \in C$. Model (7.17)-(7.24) is invoked for each class $s \in S$ to determine the minimum length film necessary to produce all items in that class. Let $\bar{z}_s^{CC}$ be the optimal solution value of the model and let

$$b_c = \sum\nolimits_{s \in S_c} \bar{z}_s^{CC} \tag{7.25}$$

be the minimum film length required to produce all items of color c. Let us also determine all possible positions for a first-stage cut on items of color c by computing

$$\mathcal{L}_c = \left\{ x = \sum\nolimits_{j \in T} l_j : 0 \le x \le b_c, T \subseteq \bigcup\nolimits_{s \in S_c} \{N_s\} \right\}. \tag{7.26}$$

Production typically happens on more than one machine, and in such a case the length b_c will be split among the machines. The values used to split b_c can be limited to those in (7.26). This consideration is at the basis of a MILP model that we developed to solve the SC.

To this purpose, we build a graph $\widetilde{G} = (\widetilde{C}, \widetilde{A})$. The vertex set is $\widetilde{C} = C \cup \{0\}$, where 0 is a dummy vertex that represents the beginning and the end of the activities. The arc set $\widetilde{A}$ connects each pair of vertices in $\widetilde{C}$. Let s_{cd} be the value of the setup length when changing production from color c to color d (recall that no setup occurs when keeping the same color and just changing the thickness). We aim at creating a working sequence for each machine $p \in M$, specifying the order in which colors are processed on that machine, and their corresponding quantity.

Let us first introduce a variable z^{SC} indicating the value of the optimal makespan. Let us also introduce two sets of three-index binary variables:

- $y_{cdp} = \begin{cases} 1 & \text{if vertex } c \text{ is followed by vertex } d \text{ on machine } p; \\ 0 & \text{otherwise;} \end{cases}$ for $c, d \in \widetilde{C}$, $p \in M$;
- $\omega_{c\ell p} = \begin{cases} 1 & \text{if } \ell \text{ units of film of color } c \text{ are used on machine } p; \\ 0 & \text{otherwise;} \end{cases}$ for $c \in C$, $\ell \in \mathcal{L}_c$, $p \in M$.

The scheduling component can be modeled as the following MILP model:

$$\min \quad z^{SC} \tag{7.27}$$

subject to

$$z^{SC} \ge \sum_{c \in \widetilde{C}} \sum_{d \in \widetilde{C}} s_{cd} y_{cdp} + \sum_{c \in C} \sum_{\ell \in \mathcal{L}_c} \ell \omega_{c\ell p} \qquad p \in M \tag{7.28}$$

$$\sum_{c \in \widetilde{C}} y_{0cp} = 1 \qquad p \in M \qquad (7.29)$$

$$\sum_{c \in \widetilde{C}} y_{c0p} = 1 \qquad p \in M \qquad (7.30)$$

$$\sum_{p \in M} \sum_{\ell \in \mathcal{L}_c} \ell \omega_{c\ell p} = b_c \qquad c \in C \qquad (7.31)$$

$$\sum_{d \in \widetilde{C}} y_{dcp} = \sum_{d \in \widetilde{C}} y_{cdp} \qquad c \in \widetilde{C}, p \in M \qquad (7.32)$$

$$\sum_{\ell \in \mathcal{L}_c} \omega_{c\ell p} \leq \sum_{d \in \widetilde{C}} y_{dcp} \qquad c \in C, p \in M \qquad (7.33)$$

$$\sum_{c \in T} \sum_{d \in T} y_{cdp} \leq |T| - 1 \qquad T \subset C, |T| \geq 1, p \in M \qquad (7.34)$$

$$y_{cdp} \in \{0, 1\} \qquad c, d \in \widetilde{C}, p \in M \qquad (7.35)$$

$$\omega_{c\ell p} \in \{0, 1\} \qquad c \in C, \ell \in \mathcal{L}_c, p \in M \qquad (7.36)$$

The objective function (7.27) minimizes the makespan of the schedule. This is forced to be not lower than the total workload (including setup and production times) on each machine p by constraints (7.28). Constraints (7.29) and (7.30) ensure, respectively, that a single path starts and ends at vertex 0 for each machine. Note that y_{00p}=1 would correspond to an empty path for machine p. Constraints (7.31) guarantee that b_c units are processed in total for each color c. Constraints (7.32) impose flow conservation for each path on each vertex. Constraints (7.33) impose that if color c is processed on machine p then the path adopted for p should enter vertex c. Constraints (7.34) are the classical subtour elimination constraints and are used to impose the connectivity of the solution.

The optimal solution value for z_{SC} represents a lower bound on the optimal PRPP solution value. The values taken by the y and ω variables are used to derive also a valid upper bound, as explained in the following.

7.5 Upper Bounding Procedures

The solution of the CC consists of a series of selected blocks. These can be used to produce all the items, but , in order to produce a feasible PRPP solution, they must be allocated to the machines by taking into account set-up times and makespan minimization. In this section we propose two upper bounding procedures that make use of this idea, that is, they take in input the blocks generated by the CC and then focus on the best way to schedule them on the machines.

7.5.1 A Heuristic Based on a Generalized Assignment Problem

The solution of the CC consists of the blocks defined by the selected $\bar{\varphi}_p$ variables (selected first-stage vertical cuts of length p), whose total length is equal to $\bar{z}_s^{CC}$ as stated in (7.17). The set of selected blocks for a color c is thus given by the union of the selected blocks for all $s \in S_c$, and their total length is b_c as stated in (7.25).

The solution of the SC does not directly consider the item lengths, but partitions b_c into a set of film segments whose length is determined by the selected $\bar{\omega}_{c\ell p}$ variables, each corresponding to a segment of length ℓ (refer also to (7.31)) .

If we manage to allocate the selected blocks from the CC into the film segments produced by the SC, then we would produce a proven optimal solution (being feasible for both cutting and scheduling and having cost equal to the lower bound z^{SC}). If this is not possible, we can at least use the allocation of minimum excess, which produces a heuristic solution. This idea is at the basis of our first upper bounding procedure.

To simplify notation, let B be the set of blocks selected by the cutting component, l'_j the length of each block $j \in B$, and c'_j the color of each block $j \in B$. Let F be the set of film segments produced by the scheduling component, l''_i the length of each segment $i \in F$ and c''_i the color of each segment $i \in F$. Let also m_i be the index of the machine processing block $i \in F$ anf ζ_p the total time (working and setup) of machine $p \in M$ in the solution of the SC.

By introducing the following decision variables

- $x_{ij} = 1$ if block $j \in B$ is assigned to segment $i \in F$, 0 otherwise;
- $s_i =$ value of the slack of segment $i \in F$;
- $z =$ makespan,

the problem of allocating blocks to segments can be modeled as the following MILP:

$$\min \quad z \tag{7.37}$$

subject to

$$\sum_{i \in F} x_{ij} = 1 \qquad j \in B \tag{7.38}$$

$$\sum_{j \in B, c'_j = c''_i} l'_j x_{ij} \leq s_i + l''_i \qquad i \in F \tag{7.39}$$

$$z \geq \sum_{i \in F, m_i = p} s_i + \zeta_p \qquad p \in M \tag{7.40}$$

$$x_{ij} \in \{0, 1\} \qquad i \in F, j \in B \tag{7.41}$$

The objective function (7.37) minimizes the makespan. Constraints (7.38) state that every block must be assigned exactly once. Constraints (7.39) force the sum of the slack variable s_i and the length l''_i of the segment $i \in F$ to be not smaller than the sum of the lengths l'_j of the blocks assigned to that segment. Constraints (7.40) compute the makespan by forcing z to be greater than or equal to the original working time of a machine p plus the total slack assigned to that machine.

Model (7.37)–(7.41) is reminiscent of the well-known *generalized assignment problem* (GAP), so our first upper bounding procedure is called *GAP based approach* (GAPBA) in the following.

7.5.2 An Iterated Local Search Algorithm

Our second upper bounding procedure relies on scheduling the set B of blocks generated by the CC on m identical machines (see the previous Section for a formal definition of B). The objective is to minimize the makespan, i.e., the maximum completion time of a block, but, as the blocks may be of different colors, sequence-dependent setup times must be taken into account. According to the three-field notation proposed by Graham et al. [139], this $\mathcal{NP}$-hard problem can be denoted as $P|s_{hk}|C_{\max}$.

The algorithm that we use to solve the $P|s_{hk}|C_{\max}$ is an adapted version of the ILS-RVND heuristic by Subramanian [268], originally designed to solve vehicle routing problems (VRPs). The most common objective function in VRPs is to minimize the total tour length, which is equivalent, on scheduling problems, to minimize the total time spent by the machines to process all jobs. However, the objective function of the $P|s_{hk}|C_{\max}$ minimizes the makespan, which is equivalent to minimizing the longest route length on VRPs. The adaptations that we implemented to take care of this difference essentially consist in modifying the way the objective function is computed throughout the algorithm. The input data for the ILS-RVND is basically a matrix that stores the time spent by a machine to process a block k immediately after a block h, which is computed by summing up the processing time of block k plus the setup time between h and k. The values for the processing times are derived from the length of the blocks generated by the CC and the setup times come directly from the PRPP input data, more precisely, from the setup times between different colors.

In short, ILS-RVND is a multi-start heuristic that combines *iterated local search* (ILS), see, e.g., Lourenço et al. [201], with *randomized variable neighborhood descent* (RVND), see, e.g., Mladenović and Hansen[219] and Subramanian et al. [269]. The algorithm alternates between local search and perturbation procedures, where the latter modifies a local optimal solution by randomly moving or swapping items between different machines. Initial solutions are generated using a greedy randomized algorithm. A detailed and comprehensive description of ILS-RVND can be found in Subramanian [268].

The local search is performed in two different levels: (i) inter-machine, that is, moves involving different machines; (ii) intra-machine, i.e., moves involving a single machine. In what follows, we describe each of the neighborhood structures used in ILS-RVND. They are exhaustively examined in a random order using the best improvement strategy.

Inter-machine neighborhoods:

- `Insertion inter(1,0)`: a block is removed and inserted in another machine;
- `Insertion inter(2,0)`: two adjacent blocks are removed and inserted in another machine;
- `Swap inter(1,1)`: permutation of two blocks assigned to different machines;
- `Swap inter(2,1)`: permutation of a block in a machine with two adjacent blocks in another machine;
- `Swap inter(2,2)`: permutation of two pairs of adjacent blocks in two different machines;
- `Cross`: the sequences of two distinct machines are split into two, creating two initial and two final subsequences. The initial subsequences are interchanged to build two new sequences (each containing an initial and a final subsequence provided by two distinct machines).

Intra-machine neighborhoods:

- `Swap`: permutation of two blocks in the same machine;
- `1-block insertion`: a block is removed from its current position and inserted in another position in the same machine;

- `2-block insertion`: two adjacent blocks are removed and inserted in another position in the same machine;
- `3-block insertion`: three adjacent blocks are removed and inserted in another position in the same machine.

7.6 Computational Experiments

The algorithms were coded in C++ and the experiments were conducted on a single core of an Intel Core i7 processor with 3.4 Ghz and 16 GB of RAM, running Ubuntu 12.04. All formulations were solved using CPLEX 12.6. A time limit of 3600 seconds and a memory limit of 10 GB were imposed for the compact formulation. For the CC, we set a time limit of 1200 seconds because only one instance could not be solved to optimality within such limit (and the remaining instances could not be solved even allowing a much larger time). The ILS-RVND algorithm was executed 10 times for each instance by adopting the same parameter values as in the original work Subramanian [268]. As for the SC, we first imposed a time limit of 3600 seconds but we later verified that this value was overestimated. Figure 7.4 depicts the value of the average gap considering all instances (described in details in the next section) between the lower bound after a particular runtime and the lower bound found after 3600 seconds. We can observe that the average initial gap is already small (0.17%) and after 300 seconds it reduces to 0.12%. The improvement obtained from that point on is not significant. Therefore, we decided to adopt 300 seconds as time limit for the SC, as it seems that this setting offers good compromise between time spent and lower bound quality.

7.6.1 Instances

Two sets of instances have been created on the basis of observations of processes in the industry producing plastic bags that was at the origin of our research. The values of the parameters were generated using uniform distribution considering the minimum and maximum values observed in practice. The first set contains 50 small instances with number of items ranging from 10 to 50, while the second one is composed of larger instances with number of items ranging from 100 to 250. For each value of n (independently of the set), two different numbers of machines were selected and two groups of instances were created, each group containing 5 randomly generated instances. Table 7.2 shows the parameters adopted

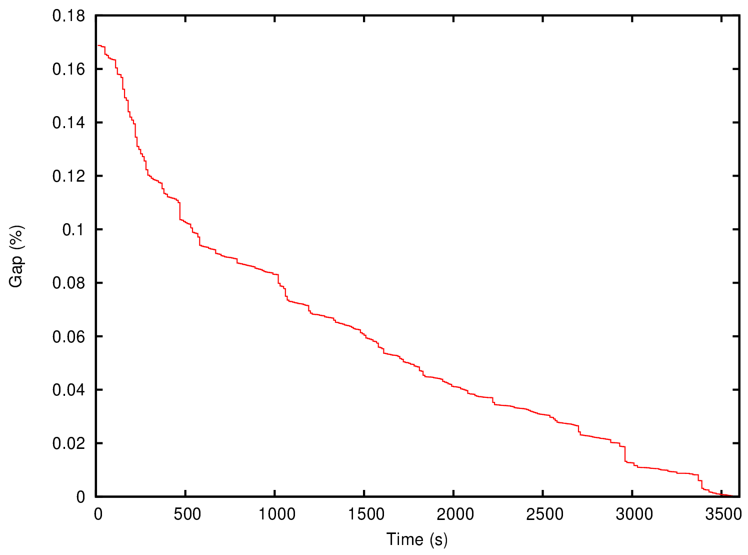


Figure 7.4: Evolution of the lower bound produced by the SC.

for each group of instances. The number of colors (#colors), the number of classes (#γ), and the number of slits (σ) were generated by using a random integer uniform distribution within the indicated intervals. For all instances, the value of W was set to 180, whereas the values of w, l, and s were uniformly randomly generated as integer values in the intervals [30,180], [10,200], and [10,30], respectively, with s being restricted to take values that are multiples of 5.

Table 7.2: Instance parameters

Small instances					Large instances				
n	m	#colors	#γ	σ	n	m	#colors	#γ	σ
10	2	[2,3]	[#colors,3]	[1,2]	100	4	[4,6]	[#colors,10]	[1,4]
	3	[2,3]	[#colors,3]	[1,2]		6	[4,6]	[#colors,10]	[1,4]
20	2	[2,3]	[#colors,4]	[1,2]	150	6	[4,8]	[#colors,14]	[1,4]
	3	[2,3]	[#colors,4]	[1,2]		8	[4,8]	[#colors,14]	[1,4]
30	3	[2,4]	[#colors,6]	[1,3]	200	6	[4,8]	[#colors,14]	[1,5]
	4	[2,4]	[#colors,6]	[1,3]		8	[4,8]	[#colors,14]	[1,5]
40	3	[2,4]	[#colors,8]	[1,3]	250	8	[4,10]	[#colors,18]	[1,5]
	4	[2,4]	[#colors,8]	[1,3]		10	[4,10]	[#colors,18]	[1,5]
50	4	[2,5]	[#colors,10]	[1,4]					
	5	[2,5]	[#colors,10]	[1,4]					

7.6.2 Algorithm performance

With respect to the small instances, we report the results for all algorithms proposed in this chapter, whereas for the larger instances, we only present the results found by the lower bounding procedure (cutting plus scheduling component) and by the ILS-RVND heuristic. In the latter case it is prohibitively expensive to use the compact formulation, and the GAPBA approach produces poor upper bounds because it relies on the output of the SC, which is executed for a short time period, thus generating low quality sequences.

In the tables presented hereafter, #inst represents the number of instances of a group, LB and UB correspond to the lower and upper bound, respectively, #opt denotes the number of optimal solutions found, time (s) indicates the CPU time in seconds, BKLB represents the best known lower bound and gap (%) is the percentage gap between the UB found by a given method and BKLB, that is: $100(UB - BKLB)/UB$.

Table 7.3 presents the aggregate results obtained by the compact formulation on the small instances. The formulation finds proven optimal solutions for 13 instances with up to 20 items, but cannot prove optimality for any of the larger instances. The average CPU time for the 10-item instances is acceptable, but it increases rapidly with the number of items. Nonetheless, the average gaps between the UBs an the BKLBs are of high quality, even for the 50-item instances. Furthermore, we notice that the average gaps tend to increase with the number of machines.

Table 7.3: Aggregate results for the compact formulation

n	m	#inst	avg. BKLB	Compact formulation				
				avg. UB	avg. LB	gap (%)	#opt	time (s)
10	2	5	571.2	571.2	571.2	0.0	5	18.3
	3	5	287.8	287.8	287.8	0.0	5	32.5
20	2	5	1001.0	1001.0	993.5	0.0	1	2895.4
	3	5	595.6	596.4	585.6	0.1	2	2961.7
30	3	5	826.1	827.6	819.5	0.2	0	3488.8
	4	5	640.2	643.8	602.4	0.6	0	3153.6
40	3	5	1027.6	1035.2	834.2	0.8	0	3595.5
	4	5	972.2	982.6	806.6	1.1	0	3595.8
50	4	5	1088.8	1092.4	682.9	0.4	0	3596.4
	5	5	806.6	823.4	441.2	2.1	0	3595.9
avg/total		50	781.7	786.1	662.5	0.5	13	2693.4

Table 7.4 presents, for the small instances, the average results obtained in terms of

lower bound by the CC followed by the SC (CS-LB), and in terms of upper bounds by GAPBA and ILS-RVND. For CS-LB we report the average CPU time spent by the CC (time_{CC}) and by the SC (time_{SC}), and the average lower bound produced for each group of instances. For GAPBA we provide the average values of CPU time, gap, and upper bound, as well as the number of proven optimal solutions found (#opt). As for ILS-RVND, we compute for each instance the average time, the best and average gaps, and the average UB (by considering the 10 runs). Then in the table we report the means of these values considering the 5 instances per line, as well as the number of proven optimal solutions.

Table 7.4: Results obtained by CS-LB, GAPBA, and ILS-RVND for the small instances

			CS-LB			GAPBA				ILS-RVND				
n	m	#inst	time_{CC} (s)	time_{SC} (s)	avg. LB	time (s)	gap (%)	avg. UB	#opt	avg. time (s)	best gap (%)	avg. gap (%)	avg. UB	#opt
10	2	5	< 0.1	< 0.1	571.2	< 0.1	0.0	571.2	5	0.1	0.1	0.0	571.2	5
	3	5	< 0.1	0.2	287.4	< 0.1	1.4	292.0	3	0.1	0.8	0.9	290.4	3
20	2	5	< 0.1	22.5	1001.0	< 0.1	0.0	1001.0	5	0.1	0.1	0.0	1001.0	5
	3	5	0.1	3.9	595.6	< 0.1	0.9	601.0	3	0.1	0.9	1.0	601.0	3
30	3	5	0.7	58.8	826.1	< 0.1	0.2	827.4	2	0.1	0.2	0.2	827.4	2
	4	5	1.2	146.4	640.2	< 0.1	0.5	643.2	2	0.1	0.5	0.5	643.2	2
40	3	5	1.6	155.8	1027.6	< 0.1	0.5	1032.8	2	0.1	0.4	0.4	1032.0	2
	4	5	0.2	299.3	972.2	< 0.1	0.9	981.2	0	0.2	0.4	0.4	976.9	0
50	4	5	1.9	178.6	1088.8	< 0.1	0.2	1091.4	3	0.2	0.2	0.2	1090.8	3
	5	5	10.2	299.6	806.6	0.1	0.9	814.2	0	0.2	0.7	0.7	812.6	0
avg/total		50	1.5	116.5	781.7	< 0.1	0.6	785.5	25	0.1	0.4	0.4	784.6	25

The combination of the two components clearly runs faster than the compact model and the former finds much better LBs than the latter, except for some 10-item instances and very few 20-item instances. We can also observe that scheduling is much more time consuming than cutting. GAPBA and ILS-RVND have an equivalent performance and both methods were capable of finding 25 proven optimal solutions, meaning that CS-LB was equal to the best UB found by GAPBA and ILS-RVND in half of the total number of instances.

GAPBA and ILS-RVND are very competitive in terms of CPU time and solution quality, the former is slightly faster, while the later provides solutions with better gaps. Both find the same number of optimal solutions and the average gap difference is of only 0.2%. Regarding the scalability of the methods with respect to the instance size, they both appear to be much better suited for practical applications, as shown by the experiments.

Figure 7.5 illustrates the average gap obtained by the compact formulation, GAPBA, and ILS-RVND for the small instances. The formulation found, on average, the best gaps for the instances with $n \leq 20$, but it is outperformed by the other two approaches as the size of the instances increase. Moreover, the solutions found by ILS-RVND appear to be systematically better than or equal to those found by GAPBA, with just a very limited increase in the required computational effort. Overall, the proposed decomposition manages to find good quality solutions and small average gaps within a much smaller computational effort that the one required by the compact formulation.

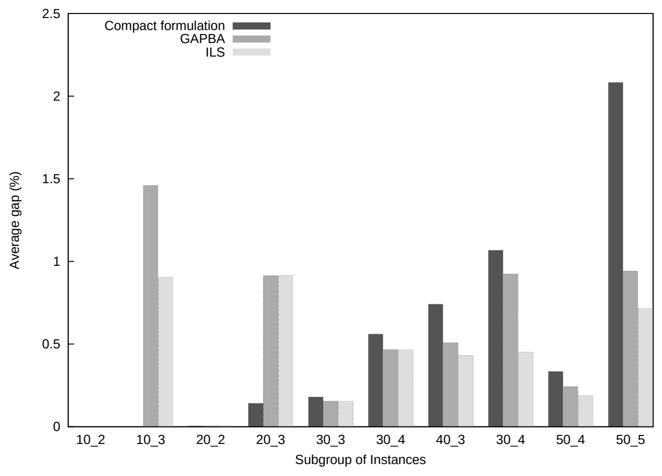


Figure 7.5: Comparison of the upper bounds on some small instances

Table 7.5 shows the aggregate results obtained by CS-LB and ILS-RVND for the large instances involving 100 to 250 items. We do not prove the optimality for any instance, but we report significantly small gaps for instances with 100 items. The gap is considerably small for instances with up to 150 items, but for the larger instances (200 and 250 items) it tends to increase considerably, especially when the number of machines is large. Nevertheless, this does not necessarily imply that the UBs found by ILS-RVND are of poor quality, since one cannot ensure that the CS-LBs are of high quality. In addition, the difficulty in producing high quality blocks by the CC may affect the performance of ILS-RVND in finding high quality solutions.

7.7 Concluding Remarks

In this chapter we introduced the Plastic Rolls Production Problem (PRPP), which integrates cutting and scheduling decisions, thus generalizing several well-known optimization problems. We proposed different approaches for obtaining lower and upper bounds for

Table 7.5: Results obtained by DM and ILS-RVND for the large instances

			CS-LB			ILS-RVND			
n	m	#inst	time$_{\text{CC}}$ (s)	time$_{\text{SC}}$ (s)	avg. LB	avg. time (s)	best gap (%)	avg. gap (%)	avg. UB
100	4	5	18.4	300.0	2056.8	1.4	0.4	0.4	2065.8
	6	5	4.2	300.0	1440.7	1.8	0.7	0.8	1452.7
150	6	5	188.4	300.0	1909.6	4.0	4.5	4.6	1990.4
	8	5	123.0	300.0	1615.9	5.4	0.8	0.9	1631.2
200	6	5	225.5	300.0	2774.2	13.9	3.5	3.5	2857.4
	8	5	496.0	300.0	1449.1	3.6	17.5	17.6	1771.7
250	8	5	360.2	300.0	1590.1	6.5	9.4	9.5	2260.4
	10	5	634.3	300.0	2040.3	13.2	20.8	20.9	1965.8
avg/total		40	289.6	300.0	1859.6	6.3	7.2	7.3	1999.4

the PRPP. The first one is a compact mathematical formulation that works quite well for instances involving up to 20 items. The second approach is based on a two-phase decomposition method designed to generate improved lower bounds, especially for instances with more than 20 items, and a heuristic information on the way to cut items into blocks. These blocks are later used in a generalized assignment problem based approach (GAPBA), as well as in a iterated local search (ILS-RVND) heuristic, to produce feasible solutions. The compact formulation found high quality lower and upper bounds for very small instances, but failed in producing good solutions for larger instances. The decomposition method found high quality lower bounds for instances with up to 100 items, although its performance seems to degradate for larger instances. Both GAPBA and ILS-RVND generated high quality solutions for instances with up to 50 items, but only the latter managed to produce good solutions for larger instances with up to 250 items.

As for future work, one can improve the quality of the performance of the two-phase decomposition by implementing a combinatorial branch-and-bound approach for solving the cutting component, as well as a column generation based algorithm for solving the scheduling component. Additionally, improved upper bounds could be obtained by proposing alternative ways of generating blocks for the ILS-RVND heuristic. The study of different problem variants, with alternative objective functions or machine characteristics, is also of interest.

Chapter 8

Integer Linear and Constraint Programming for Project Scheduling Problems

[1]

In project scheduling problems, a set of tasks has to be scheduled so that the total makespan of the project is minimized. In this chapter, we study three kinds of project scheduling problems: the resource constrained project scheduling problem, for which we propose improvements to existing time-indexed formulations; the discrete time-cost tradeoff problem for which we introduce a new MILP model; and the multi-mode resource-constrained project scheduling problem for which we propose an hybridized algorithm. For each of the problems, we compare the proposed algorithms with classical MILP formulations and constraint programming approaches from the literature.

Keywords: Resource constrained project scheduling problem, Discrete time-cost tradeoff problem, Multi-mode resource-constrained project scheduling problem, Exact algorithms.

8.1 Introduction

Given a set of p renewable resources with availability b_k $(k = 1, \dots, p)$, a set p' of non-renewable resources with availability b'_k $(k = 1, \dots, p')$, a set of n tasks with m_i execution modes $(i = 1, \dots, n)$, each consisting in a combination of duration d_{ij} $(i = 1, \dots, n; j = 1, \dots, m_i)$, renewable resources consumption r_{ijk} $(i = 1, \dots, n; j = 1, \dots, m_i; k = 1, \dots, p)$, and non-renewable resources consumption r'_{ijk} $(i = 1, \dots, n; j = 1, \dots, m_i; k = 1, \dots, p')$, and a list of precedence constraints H, the *Multi-Mode Resource-Constrained Project*

[1]This chapter gathers the results of preliminary research conducted by J.F. Côté, M. Delorme, M. Iori, and S. Martello.

Scheduling Problem (MMRCPSP) consists in finding the execution mode and the starting time for each task so that the resource availabilities and the precedence relations are satisfied, and the project makespan is minimized.

The MMRCPSP is a very general problem that can also be used to model some other, less constrained, problems. For example, when no non-renewable resources are considered and only one mode per task is available, the problem is called the *Resource-Constrained Project Scheduling Problem* (RCPSP). When no renewable resource is considered, and only one non-renewable resource is available, the problem is called the *Discrete Time-Cost Tradeoff Problem* (DTCTP). These problems are important because they have many real world applications, especially in material and human resource management (see, e.g., Artiguez et al. [14]).

The RCPSP is $\mathcal{NP}$-hard in the strong sense. Indeed, consider the *bin packing problem* (BPP), in which we are given a set of n items of weight w_i $(i = 1, \ldots, n)$ to pack into the minimum number of bins with capacity c (see, e.g., Delorme et al. [98] for a recent survey). Any BPP instance can be seen as an RCPSP by setting $p = 1$, $b_1 = c$, $H = \emptyset$ and, for $i = 1, \ldots, n$, $d_{i1} = 1$ and $r_{i11} = w_i$. The minimum makespan found by the RCPSP would be the minimum number of bins required for the BPP. As the BPP is known to be $\mathcal{NP}$-hard in the strong sense (see, e.g., Garey and Johnson [130]), the same holds for the RCPSP, and thus, for the MMRCPSP. As far as the DTCTP is concerned, it was shown to be $\mathcal{NP}$-hard in the strong sense as well by De et al. [90].

In this chapter, we strengthen existing time indexed formulations for the RCPSP, we introduce a new *mixed integer linear programming* (MILP) formulation for the DTCTP, and we propose a hybridized algorithm for the MMRCPSP. In Section 8.2, we review the existing literature for the three problems while we detail some mathematical formulations in Section 8.3. The proposed algorithms are given in Section 8.4 and the outcome of extensive computational experiments is presented in Section 8.5.

8.2 Literature review

The literature on project scheduling problems is very dense and a complete review of it is beyond the scope of this study. We mention, however, relevant surveys and interesting approaches that have been proposed for the RCPSP, the DTCTP, and the MMRCPSP.

Among the three problems, the RCPSP is probably the one that was the most studied in the literature. A recent survey proposed by Artiguez et al. [14] reviewed around 300 references on the RCPSP and its extensions. The *Discrete-time* (DT) formulation, probably the first mathematical model for the RCPSP, was proposed almost 50 years ago by Pritsker et al. [232]. The *Disaggregated Discrete-Time* (DDT) formulation was proposed in the late Eighties by Christofides et al. [63]. Other MILP models were proposed later on by Alvarez-Valdes and Tamarit [8], Mingozzi et al. [217], Artigues et al. [15], and Koné et al. [178], but they appear to have mainly theoretical interests as the results they exhibit when solved through MILP solvers tend to be worse than those obtained by DT and DDT (see Koné et al. [178]). Among the exact approaches that were tried to solve the RCPSP, we cite branch-and-bound algorithms (see, e.g., Demeulemeester and Herroelen [102, 103], Brucker et al. [47], Klein and Scholl [175], and Sprecher [265]), and *constraint programming* (CP) approaches (see, e.g., Dorndorf et al. [105], Laborie [183], Liess and Michelon [189], and Schutt et al. [253]).

The DTCTP was also extensively studied in the literature, we mention the very detailed surveys by Hartmann and Briskorn [149] and by Węglarz et al. [290], who identified three different objective functions for the problem. The one that can be obtained from the MMRCPSP is called the *budget problem* (b-DTCTP), and aims at minimizing the makespan while the budget (the unique non-renewable resource) is bounded. In the *deadline problem* (d-DTCTP), the objective is to minimize the budget while the deadline is bounded. In the *curve problem* (c-DTCTP), the objective is to provide a time-cost efficient curve, i.e., the minimum budget required for a given deadline that is allowed to vary. In terms of exact approaches, we mention the branch-and-bound procedures by Demeulemeester et al. [101] and by Değirmenci and Azizoğlu [91], and the decomposition approaches by Hazır et al. [151] and Hadjiconstantinou and Klerides [144]. In terms of heuristics, we mention the upper and lower bounding procedures based on column generation by Akkan et al. [4].

For the MMRCPSP, we refer the reader to the detailed survey by Węglarz et al. [290]. We mention in addition the work by Coelho and Vanhoucke [70] that solves heuristically the MMRCPSP by decomposing it into successions of two phases: a mode selection step (b-DTCTP) and a task scheduling step (RCPSP).

8.3 Mathematical models

In this section, we gather the most common mathematical models that have been proposed in the literature for the three problems we study. As often happens in project scheduling problems, we make use of a dummy tasks 0 with duration 0, no resource consumption, and being predecessor of all tasks, which represents the beginning of the project. We use a similar dummy task $n+1$, successor of all tasks, which represents the end of the project.

8.3.1 RCPSP formulations

The DT formulation by Pritsker et al. [232] associates for each task i ($i = 0, \dots, n+1$) and each unit of time t ($t = 0, \dots, T$), a variable decision x_{it} that takes value 1 if task i starts at time t, and 0 otherwise. It makes use of an upper bound T on the total makespan of the project and of $ES(i)$ (resp. $LS(i)$) the earliest start (resp. the latest start) of task i ($i = 0, \dots, n+1$). A trivial value of T can be computed as $T = \sum_{i=0}^{n} d_i$, while $ES(i)$ and $LS(i)$ can be computed by the critical path method, for example, and $LS(n+1)$ is bounded by T. The DT model for the RCPSP is the following

$$\min \quad z = \sum_{t=ES(n+1)}^{LS(n+1)} t\, x_{n+1,t} \tag{8.1}$$

$$\text{s.t.} \sum_{t=ES(i)}^{LS(i)} x_{it} = 1 \qquad i = 0, \dots, n+1, \tag{8.2}$$

$$\sum_{t=ES(l)}^{LS(l)} t\, x_{lt} - \sum_{t=ES(i)}^{LS(i)} t\, x_{it} \geq d_i \qquad (i,l) \in H, \tag{8.3}$$

$$\sum_{i=0}^{n+1} r_{ik} \sum_{\tau=\max(0,t-d_i+1)}^{t} x_{i\tau} \leq b_k \qquad t = 0, \dots, LS(n+1); k = 1, \dots, p, \tag{8.4}$$

$$x_{it} \in \{0,1\} \qquad i = 0, \dots, n+1; t = ES(i), \dots, LS(i). \tag{8.5}$$

Objective function (8.1) minimizes the makespan of the project, while constraints (8.2) impose that each task is scheduled exactly once. Constraints (8.3) ensure that the precedence constraints are satisfied and constraints (8.4) ensure that the resource availabilities are respected.

In the DDT formulation by Christofides et al. [63], the precedence constraints (8.3) are disaggregated and replaced by

$$\sum_{\tau=t}^{LS(i)} x_{i\tau} + \sum_{\tau=ES(l)}^{\min(LS(l),t+d_i-1)} x_{l\tau} \le 1 \qquad (i,l) \in H; t = ES(i), \ldots, LS(i). \tag{8.6}$$

8.3.2 DTCTP formulations

The textbook formulation for the DTCTP can be found in Akkan et al. [4]. It associates for each task i ($i = 0, \ldots, n+1$) and each mode j ($j = 1, \ldots, m_i$) a variable decision x_{ij} that takes value 1 if task i is chosen in mode j, and 0 otherwise. In addition, it makes use of variables S_i ($i = 0, \ldots, n+1$) that represents the starting time of task i. Values d_{ij} and c_{ij} represent the duration and the cost of tasks i in mode j. The d-DTCTP can be modelled as follow:

$$\min \quad z = \sum_{i=0}^{n+1} \sum_{j=1}^{m_i} x_{ij}\ c_{ij}, \tag{8.7}$$

$$\text{s.t.} \quad S_l \ge S_i + \sum_{j=1}^{m_i} x_{ij}\ d_{ij} \qquad (i,l) \in H, \tag{8.8}$$

$$S_{n+1} \le D, \tag{8.9}$$

$$\sum_{j=1}^{m_i} x_{ij} = 1 \qquad i = 0, \ldots, n+1, \tag{8.10}$$

$$x_{ij} \in \{0,1\} \qquad i = 0, \ldots, n+1; j = 1, \ldots, m_i, \tag{8.11}$$

$$S_i \ge 0 \qquad i = 0, \ldots, n+1. \tag{8.12}$$

Objective function (8.7) minimizes the budget of the project while constraints (8.8) ensure that the precedence constraints are satisfied. Constraint (8.9) impose the deadline restriction, while constraints (8.10) ensure that exactly one mode is chosen for each task.

The b-DTCTP can be modelled in a very similar way by minimizing S_{n+1} instead of the sum of the costs in the objective function (8.7), and replacing (8.9) by

$$\sum_{i=0}^{n+1} \sum_{j=1}^{m_i} x_{ij}\ c_{ij} \le B \tag{8.13}$$

to satisfy the budget constraint. As far as the c-DTCTP is concerned, it can be modelled by multiple d-DTCTP in which the deadline D is modified.

8.3.3 MMRCPSP formulations

A DT formulation adapted to the MMRCPSP was given by Talbot [271]. It associates for each task i ($i = 0, \dots, n+1$), each mode j ($j = 1, \dots, m_i$), and each unit of time t ($t = 0, \dots, T$) a variable decision x_{ijt} that takes value 1 if task i is chosen in mode j, and starts at time t, and 0 otherwise. The resulting MILP formulation is

$$\min \quad z = \sum_{t=ES(n+1)}^{LS(n+1)} t\, x_{n+1,1,t}, \tag{8.14}$$

$$\text{s.t.} \quad \sum_{j=1}^{m_i} \sum_{t=ES(i)}^{LS(i)} x_{ijt} = 1 \qquad i = 0, \dots, n+1, \tag{8.15}$$

$$\sum_{j=1}^{m_l} \sum_{t=ES(l)}^{LS(l)} t\, x_{ljt} - \sum_{j=1}^{m_i} \sum_{t=ES(i)}^{LS(i)} (t + d_{ij})\, x_{ijt} \geq 0 \qquad (i,l) \in H, \tag{8.16}$$

$$\sum_{i=0}^{n+1} \sum_{j=1}^{m_i} r_{ijk} \sum_{\tau=\max(0,t-d_{ij}+1)}^{t} x_{ij\tau} \leq b_k \quad t = 0, \dots, LS(n+1); k = 1, \dots, p, \tag{8.17}$$

$$\sum_{i=0}^{n+1} \sum_{j=1}^{m_i} r_{ijk'} \sum_{t=ES(i)}^{LS(i)} x_{ijt} \leq b'_k \qquad k = 1, \dots, p', \tag{8.18}$$

$$x_{ijt} \in \{0,1\} \qquad i = 0, \dots, n+1; j = 1, \dots, m_i; t = ES(i), \dots, LS(i). \tag{8.19}$$

Objective function (8.14) minimizes the makespan of the project, while constraints (8.15) impose that each task is scheduled exactly once, in only one mode. Constraints (8.16) ensure that the precedence constraints are satisfied, and constraints (8.17) and (8.18) impose that the renewable and non-renewable resource availabilities are respected.

8.4 Proposed approaches

In this section, we describe the improved algorithms that we developed for each of the three problems.

8.4.1 RCPSP improved algorithm

We used the DDT model of Christofides et al. [63], in which we added the following procedures and preprocessing techniques.

Improved ES and LS

Mingozzi et al. [217] proposed a global lower bound on the duration of the project by solving a relaxation of the RCPSP in which preemption is allowed and precedence constraints are relaxed. The method usually leads to a better $ES(n+1)$ than the one given by the critical path method. This approach makes use of the concept of maximal feasible sets F of activities, where a set of tasks f belongs to F if all the tasks that belong to f can be processed at the same time (i.e., respecting the resource and the precedence constraints). We extended this approach and used it to each task by solving the following MILP

$$\min \quad ES(l) = \sum_{f \in F} x_f, \tag{8.20}$$

$$\text{s.t.} \quad \sum_{f \in F : i \in f} x_f \geq d_i \qquad (i,l) \in H, \tag{8.21}$$

$$x_f \in \mathbb{N} \qquad f \in F. \tag{8.22}$$

Objective function (8.20) minimizes the number of feasible sets used, while constraints (8.21) ensure that for each predecessor i, at least d_i feasible sets containing task i are selected. When the upper bound T of the project is fixed, a similar approach using the successors of tasks l can be used to obtain $LS(l)$.

Destructive bounds

The concept of destructive bounds was described by Klein and Scholl [175] and transforms a minimization problem into a sequence of feasibility problems. For the RCPSP, we set T to $ES(n+1)$. If a solution is found, then it is optimal and we stop the process. If we prove that no solution exists, then we increase $ES(n+1)$ by one unit and iterate. This very common approach was also used for other optimization problem, e.g., by Delorme and Iori [97] for the BPP, or by Côté et al. [83] for the strip packing problem.

Task fixing

The concept of task fixing is derived from the preprocessing techniques usually adopted for packing problems (see, e.g., Martello and Toth [214]). At this stage, we try to fix a given task i with no unfixed predecessors to $ES(i)$. We use the concept of temporal maximal feasible sets F_t of activities, where a set of tasks f belongs to F_t if all the tasks that belong to f can be processed at the same time t (i.e., respecting the resource and the precedence constraints, and satisfying $ES(i) \leq t, LS(i) \geq t (i \in f)$). If task i belongs to all F_t for $t = ES(i), \ldots, ES(i) + d_i - 1$, then i can be fixed at $ES(i)$. Indeed, as task i does not have any unfixed predecessor, no constraint (8.6) with $(h, i) \in H$ is considered in the model. In addition, as we fix i to $ES(i)$, no specific restriction is added by constraints (8.6) with $(i, l) \in H$. Finally, as i belongs to all temporal maximal feasible sets F_t for $t = ES(i), \ldots, ES(i) + d_i - 1$, i can be processed in parallel with any set of tasks at time t, $t = ES(i), \ldots, ES(i) + d_i - 1$, thus, constraints (8.4) do not remove any valid solution by setting i to $ES(i)$. When destructive bounds are used, a similar preprocessing fixes a task with no unfixed successors at $LS(i)$. Task fixing can be iterated and is particularly useful when short tasks, or tasks using very small amount of resources, are considered at the beginning or at the end of the project.

Normal Patterns

The concept of normal patterns was introduced by Herz [153] and by Christofides and Whitlock [64] for packing problems and is nowadays widely used as a preprocessing techniques for combinatorial optimization problem (see, e.g., Côté et al. [83]). Using normal patterns, we can reduce $PS(i)$, the set of possible starts of task i, to

$$PS(i) = \left\{ x = \sum_{j \in O} d_j \; \xi_j : ES(i) \leq x \leq LS(i) \; \xi_j \in \{0, 1\} \right\}, \tag{8.23}$$

where O is the set of tasks that can be processed before the beggining of task i.

Weight lifting

Weight lifting is also a very common preprocessing in packing problems (see, e.g., Boschetti and Montaletti [43] and Martello and Toth [214]), and has the objective to increase the resource consumption without removing any maximal feasible sets of activities.

Instead of a unique resource consumption, we make use of a resource consumption per unit time r_{ikt} and determine s_{ikt}, the maximum resource consumption that can be used while task i is processed at time t, by solving a subset-sum problem. We then set r_{ikt} to $b_k - s_{ikt}$ and modify constraints (8.4) to

$$\sum_{\tau=\max(0,t-d_i+1)}^{t} \sum_{i=0}^{n+1} r_{ik\tau} x_{i\tau} \leq b_k \quad t = 0, \ldots, LS(n+1); k = 1, \ldots, p. \tag{8.24}$$

8.4.2 DTCTP improved algorithm

Pritsker and Watters [231] introduced the *step* model for the RCPSP, in which a binary variable x_{it} takes value 1 if task i starts at time t or before. We extended this approach to the d-DTCTP and propose the following model

$$\begin{aligned}
\min \quad & z = \sum_{i=0}^{n+1} x_{i1}\, c_{i1} + \sum_{i=0}^{n+1} \sum_{j=2}^{m_i} (x_{ij} - x_{i,j-1})\, c_{ij}, && && (8.25)\\
\text{s.t.} \quad & S_l \geq S_i + x_{i1}\, d_{i1} + \sum_{j=2}^{m_i} (x_{ij} - x_{i,j-1})\, d_{ij} && (i,l) \in H, && (8.26)\\
& S_{n+1} + \leq D, && && (8.27)\\
& x_{i,m_i} = 1 && i = 0, \ldots, n+1, && (8.28)\\
& x_{ij} \geq x_{i,j-1} && i = 0, \ldots, n+1; j = 2, \ldots, m_i, && (8.29)\\
& x_{ij} \in \{0,1\} && i = 0, \ldots, n+1; j = 1, \ldots, m_i, && (8.30)\\
& S_i \geq 0 && i = 0, \ldots, n+1, && (8.31)
\end{aligned}$$

in which x_{ij} takes value 1 is task i is used in mode m, $m \leq j$. Objective function (8.25) minimizes the budget of the project while constraints (8.26) ensure that the precedence constraints are satisfied. Constraint (8.27) impose the deadline restriction while constraints (8.28) ensure that a mode has been selected for each task. Constraints (8.29) impose that if a variables x_{ij} takes value one, then all variables x_{im} for $m > j$ take value one as well.

Again, the b-DTCTP can be modelled in a very similar way by minimizing S_{n+1} in the objective function (8.25), and modifying (8.27) by

$$\sum_{i=0}^{n+1} x_{i1}\ c_{i1} + \sum_{i=0}^{n+1} \sum_{j=2}^{m_i} (x_{ij} - x_{i,j-1})\ c_{ij} \leq B \tag{8.32}$$

to satisfy the budget constraint.

8.4.3 MMRCPSP improved algorithm

Based on the idea proposed by Coelho and Vanhoucke [70], we decompose the MM-RCPSP into two phases: first, we solve a b-DTCTP (with one constraint (8.32) for each non-renewable resource) that fixes the modes selected for the tasks and gives a lower bound $ES(n+1)$ for the project makespan. At a second stage, we solve a RCPSP with a fixed deadline $LS(n+1) = ES(n+1)$ and the modes selected at the first stage. If a solution exists, then the problem is solved. Otherwise, we use the complete CP approach described in Section 8.5.

8.5 Computational experiments

In this section, we experimentally compare the efficiency of the approaches proposed in Section 8.4 with mathematical formulations from the literature (see Section 8.3) and a CP approach. We used the examples given in Cplex 12.6 as a base for our CP algorithms, that run the following constraints:

- `IloMinimize(IloEndOf(`$n+1$`))`: models the objective function that minimizes the makespan of the project, i.e., the ending time of task $n+1$;
- `IloEndBeforeStart(`i,l`)` $(i,l) \in H$: model the precedence constraints, i.e., task i has to be finished before task l starts;
- `IloPulse(`i,k`)` $(i = 0,\ldots,n+1; k = 1,\ldots,p)$: model the renewable resource k used by task i;
- `IloCumulFunctionExpr(`k`)` $(k = 1,\ldots,p)$: model the total renewable resource consumption of k and is associated with the sum of the `IloPulse` for resource k;
- `IloAlternative(`i,j`)` $(i = 0,\ldots,n+1; j = 1,\ldots,m_i)$: model the possible modes j for task i.

We refer the reader to Laborie [184] for more details about CP optimizer for scheduling problems. The CP optimizer of Cplex 12.6 uses two concurrent search strategies by default: a large neighbourhood search produces feasible solutions of good quality and a failure directed search proves the infeasibility of a solution strictly better than the current solution. All our experiments were executed on an Intel Xeon 3.10 GigaHertz with 8 GigaByte RAM, equipped with four cores, and we used Cplex 12.6 as MILP and CP solver. All our experiments were performed with a single core, and the number of threads was set to one for the solver. In each table, the approach that finds the largest number of optimal solutions is highlighted in bold, and when an instance is not solved, its associated time is set to the time limit.

8.5.1 Computational experiments for the RCPSP

We used the benchmark KSD30, a set of 480 instances available at the PSPLIB of Kolisch and Sprecher [177]. Each instance has $n = 30$ tasks (plus the 2 dummy initial and final tasks), $p = 4$ resources, durations $d_i \in [1, 10]$ $(i = 1, \dots, n)$ and resource consumptions $r_{ik} \in [0, 10]$ $(i = 1, \dots, n; k = 1, \dots, p)$.

Table 8.1 provides the results obtained by running models DT, DDT, DDT with the improvements described in Section 8.4, DDT with the improvements and some modifications in the solver parameters (e.g., branching priorities and cut generation), and the CP approach, with a time limit of 300 seconds. The first column identifies the model, the two following columns count the number of instances that were solved within the time limit, and the last column represents the average CPU time expressed in seconds.

Table 8.1: Evaluation of the RCPSP approaches on the KSD30 instances

approach	# optimal solution	% optimal solution	time
DT model	420	88%	47.2
DDT model	430	90%	37.9
DDT model + improvements	461	96%	19.3
DDT model + improvements + solver tuning	467	97%	15.9
CP approach	**480**	100%	1.4

Table 8.1 shows that the MILP formulations DT and DDT are already effective, as they can solve respectively 88% and 90% of the KSD30 instances. These results are concordant with those proposed by Koné et al. [178] who obtained respectively 78% and 82% when the

models where solved by a branch-and-bound algorithm. When improvements are applied to the DDT model, we managed to increase this ratio to 96%, (97% if solver tuning is considered). However, we cannot compete on this set of instance with the CP approach, as it can solve all the tested instances with an average time of just 1.4 second.

8.5.2 Computational experiments for the DTCTP

We used three benchmarks proposed in the literature: the first one, introduced by Demeulemeester et al. [101], is composed of 1800 instances, with $n = \{10, 20, 30, 40, 50\}$, $m_i \in [1, 11]$ $(i = 1, \ldots, n)$, d_i and $c_{ij} \in [1, 100]$ $(i = 1, \ldots, n; j = 1, \ldots, m_i)$. The two remaining sets, initially introduced by Akkan et al. [4], have diverse parameters: coefficient of network complexity (CNC), complexity index (CI), m_i, time-cost function, and θ, a parameter that gives the project deadline D depending on the minimum and maximum time required to realize the project. They were used by Hadjiconstantinou and Klerides [144] who also provided us with some of the instances in those sets.

Table 8.2 provides the results obtained by running the textbook model by Akkan et al. [4], the step model described in Section 8.4, a CP approach, and the decomposition approach proposed by Hadjiconstantinou and Klerides [144], with a time limit of 200 seconds for the c-DTCTP and the d-DTCTP, and 7200 seconds for the b-DTCTP. The two first columns identify the benchmark and the corresponding number of instances. Each of the four following set of columns associate with each formulation the number of instances that were solved to proven optimality, the ratio of instances that were solved to proven optimality, and the average CPU time expressed in seconds. As the number of optimal solutions were not provided in [144], we only report the ratio of optimal solution found.

Table 8.2: Evaluation of the DTCTP approaches

Set of instances	# inst.	textbook model			step model			CP approach			Hadjiconstantinou	
		# opt	% opt	time	# opt	% opt	time	# opt	% opt	time	% opt	time
Demeulemesteer - c-DTCTP	1800	1798	100%	6.9	**1799**	100%	6.6	1018	57%	109.3	93%	35.6
Akkan 1 - d-DTCTP	960	917	96%	14.5	**941**	98%	10.6	0	0%	200.0	59%	115.8
Akkan 1 - b-DTCTP	240	234	98%	445.5	**240**	100%	50.0	38	16%	5309.2	-	-
Akkan 2 - d-DTCTP	1920	**1920**	100%	0.2	**1920**	100%	0.1	539	28%	165.7	**100%**	6.4

Table 8.2 shows that the textbook MILP formulation is already very effective, as it can solve around 99% of the tested instances. The step formulation is even better as

it can solve all but 20 instances, and seems faster for the b-DTCTP. Both formulations seem better than the decomposition approach proposed by Hadjiconstantinou and Klerides [144], but the results should be interpreted with caution, as (1) the indicator given by PassMark© Software (see `https://www.cpubenchmark.net/`) for the computer they used is at 628 while ours is at 6106, indicating that our computer is about ten times faster, and (2), the version of Cplex they use (11.1) is older with respect to ours (12.6). As far as the CP approach is concerned, we can state that it is definitely not competitive.

8.5.3 Computational experiments for the MMRCPSP

We used the benchmark J30, a set of 640 instances available at the PSPLIB of Kolisch and Sprecher [177]. Each instance has $n = 30$ tasks (plus the 2 dummy initial and final tasks), $m_i = 3$ $(i = 1, \dots, n)$ modes $p = 2$ renewable resources, $p' = 2$ non-renewable resources, durations $d_{ij} \in [1, 10]$ $(i = 1, \dots, n; j = 1, \dots, m_i)$, renewable resource consumptions $r_{ijk} \in [0, 10]$ $(i = 1, \dots, n; j = 1, \dots, m_i; k = 1, \dots, p)$, and non-renewable resource consumptions $r'_{ijk} \in [0, 10]$ $(i = 1, \dots, n; j = 1, \dots, m_i; k = 1, \dots, p')$. As out of the 640 instances, 88 are infeasible because of the non-renewable resource constraints, we consider a reduced set of 552 instances.

Table 8.3 provides the results obtained by running model DT, a CP approach, and the hybridized algorithm described in Section 8.4 with a time limit of 300 seconds. Considering the computational results we obtained on the other problems, we solved the DTCTP component of the problem with the step model, and the RCPSP component with the CP approach. In case the decomposition was unsuccessful, we used the CP approach for the overall problem. The first column identifies the approach used, the two following columns count the number of instances that were solved within the time limit, and the last column represents the average CPU time expressed in seconds. In addition, for the hybridized algorithm, we add in parenthesis the number of times the decomposition of the problem managed to close the instance.

Table 8.3: Evaluation of the MMRCPSP approaches on the J30 instances

approach	# optimal solution	% optimal solution	time
DT model	478	87%	50.5
CP approach	**523**	95%	23.5
Hybridized algorithm	**523** (251)	95% (45%)	23.1

Table 8.3 shows that the DT MILP formulation for the MMRCPSP is less effective than the CP approach. The proposed hybridized algorithm performs slightly better than the pure CP approach, especially on the two first instances of the set, for which CP uses respectively 5.0 and 177.9 seconds to close the instance, while the hybridized approach takes less than 0.1 second for both of them. This seems to indicate that a decomposition approach might be useful for some instances with specific parameters.

8.6 Conclusion

We have studied three project scheduling problems: the resource-constrained project scheduling problem, the discrete time-cost tradeoff problem, and the multi-mode resource-constrained project scheduling problem. For each problem, we proposed a set of improvements with respect to existing algorithms from the literature, we evaluated them through extensive computation experiments, and we compared them with standard mixed integer linear programming models and constraint programming approaches. The tests demonstrate that the proposed algorithms are useful tools for the solution of the three problems.

Bibliography

[1] T. Achterberg and R. Wunderling. Mixed integer programming: Analyzing 12 years of progress. In M. Jünger and G.Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer Berlin Heidelberg, 2013.

[2] S. Ahn, C. Park, and K. Yoon. An improved best-first branch and bound algorithm for the pallet-loading problem using a staircase structure. *Expert Systems with Applications*, 42:7676–7683, 2015.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice-Hall, Upper Saddle River, 1993.

[4] C. Akkan, A. Drexl, and A. Kimms. Network decomposition-based benchmark results for the discrete timecost tradeoff problem. *European Journal of Operational Research*, 165:339–358, 2005.

[5] R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. A branch-and-cut algorithm for the pallet loading problem. *Computers & Operations Research*, 32:3007–3029, 2005.

[6] R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. A tabu search algorithm for the pallet loading problem. *OR Spectrum*, 27:43–61, 2005.

[7] R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. A branch and bound algorithm for the strip packing problem. *OR Spectrum*, 31:431–459, 2009.

[8] R. Alvarez-Valdes and J.M. Tamarit. The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67:204–220, 1993.

[9] C. Alves, F. Clautiaux, J.M. Valério de Carvalho, and J. Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization.* Springer International Publishing, Cham, 2016.

[10] C Alves and J.M. Valério de Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35:1315–1328, 2008.

[11] A.C.F. Alvim, C.C. Ribeiro, F. Glover, and D.J. Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10:205–229, 2004.

[12] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[13] Y. Arahori, T. Imamichi, and H. Nagamochi. An exact strip packing algorithm based on canonical forms. *Computers & Operations Research*, 39:2991–3011, 2012.

[14] C. Artigues, S. Demassey, and E. Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications.* John Wiley & Sons, New York, 2013.

[15] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[16] C.R. Asfahl, S. Swayze, J. Lee, and R.R. Safford. An interactive computer training programming for industry. *Computers & Industrial Engineering*, 25:57–60, 1993.

[17] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, 10:43–66, 2012.

[18] B.S. Baker, E.G. Coffman, Jr., and R.L. Rivest. Orthogonal packing in two dimensions. *SIAM Journal on Computing*, 9:846–855, 1980.

[19] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

[20] R. Baldacci and M.A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183:1136 – 1149, 2007.

[21] J. Balogh, J. Békési, G. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms – SODA 2015*, pages 1425–1438, 2015.

[22] S. Barnett and G.J. Kynch. Exact solution of a simple cutting problem. *Operations Research*, 15:1051–1056, 1967.

[23] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[24] J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[25] J.E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[26] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33:49–64, 1985.

[27] J. Beck and V. Sós. *Discrepancy theory*, pages 1405–1446. Elsevier, Amsterdam, 1995.

[28] G. Belov. *Problems, models and algorithms in one-and two-dimensional cutting.* PhD thesis, Otto-von-Guericke Universität Magdeburg, 2003.

[29] G. Belov and G. Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141:274–294, 2002.

[30] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171:85–106, 2006.

[31] G. Belov, G. Scheithauer, C. Alves, and J.M. Valério de Carvalho. Gomory cuts from a position-indexed formulation of 1D stock cutting. In A. Bortfeldt, J. Homberger, H. Kopfer, G.r Pankratz, and R. Strangmeier, editors, *Intelligent Decision Support*, pages 3–14. Gabler, 2008.

[32] H. Ben Amor, J. Desrosiers, and J.M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54:454–463, 2006.

[33] H. Ben Amor and J. Valério de Carvalho. Cutting stock problems. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 131–161. Springer US, 2005.

[34] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[35] B.E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The Computer Journal*, 25:353–357, 1982.

[36] J.A. Bennell, J.F. Oliveira, and G. Wäscher. Cutting and packing. *International Journal of Production Economics*, 145(2):449 – 450, 2013.

[37] C. Berge and E.L. Johnson. Coloring the edges of a hypergraph and linear programming techniques. *Annals of Discrete Mathematics*, 1:65–78, 1977.

[38] J.O. Berkey and P.Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[39] A.K. Bhatia and S.K. Basu. Packing bins using multi-chromosomal genetic representation and better fit heuristic. In *Neural Information Processing – ICONIP 2004*, volume 3316 of *Lecture Notes in Computer Science*, pages 181–186. Springer Berlin Heidelberg, 2004.

[40] A.K. Bhatia, M. Hazra, and S.K. Basu. Better-fit heuristic for one-dimensional bin-packing problem. In *Proceedings of the IEEE international advance computing conference – IACC 2009*, pages 193–196. IEEE, 2009.

[41] E.G. Birgin, R.D. Lobato, and R. Morabito. An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *Journal of the Operational Research Society*, 61:306–320, 2010.

[42] L. Bodin and S.I. Gass. On teaching the analytic hierarchy process. *Computers & Operations Research*, 30:1487–1497, 2003.

[43] M.A. Boschetti and L. Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58:1774–1791, 2010.

[44] J.-C. Bourjolly and V. Rebetez. An analysis of lower bound procedures for the bin packing problem. *Computers & Operations Research*, 32:395–405, 2005.

[45] F. Brandão and J.P. Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.

[46] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113:299–344, 2008.

[47] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.

[48] R.E. Burkard, S.E Karisch, and F. Rendl. QAPLIB– a quadratic assignment problem library. *European Journal of Operational Research*, 55:115 – 119, 1991.

[49] R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB– a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997.

[50] E.K. Burke, M.R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869. Springer Berlin Heidelberg, 2006.

[51] E.K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52:655–671, 2004.

[52] H. Cambazard and B. O'Sullivan. Propagating the bin packing constraint using linear programming. In *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 129–136. Springer Berlin Heidelberg, 2010.

[53] K. Cantor. *Blown Film Extrusion: An Introduction.* Hanser Publishers, Munich, second edition, 2011.

[54] A. Caprara, M. Dell'Amico, J.C. Díaz Díaz, M. Iori, and R. Rizzi. Friendly bin packing instances without integer round-up property. *Mathematical Programming*, 150:5–17, 2015.

[55] A. Caprara, A. Lodi, S. Martello, and M. Monaci. Packing into the smallest square: Worst-case analysis of lower bounds. *Discrete Optimization*, 3:317–326, 2006.

[56] A. Caprara and M. Monaci. Bidimensional packing by bilinear programming. *Mathematical Programming*, 118:75–108, 2009.

[57] M. Casazza and A. Ceselli. Exactly solving packing problems with fragmentation. *Computers & Operations Research*, 75:202–213, 2016.

[58] P.M. Castro and J.F. Oliveira. Scheduling inspired models for two-dimensional packing problems. *European Journal of Operational Research*, 215:45–56, 2011.

[59] L.M.A. Chan, D. Simchi-Levi, and J. Branel. Worst-case analyses, linear programming and the bin-packing problem. *Mathematical Programming*, 83:213–227, 1998.

[60] H-Y Chao, M.P. Harper, and R.W. Quong. A tight lower bound for optimal bin packing. *Operations Research Letters*, 18:133–138, 1995.

[61] B. Chazelle. The bottom-left bin packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32:697–707, 1983.

[62] B. Chen and B. Srivastava. An improved lower bound for the bin-packing problem. *Discrete Applied Mathematics*, 66:81–94, 1996.

[63] N. Christofides, R. Alvarez-Valdes, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

[64] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[65] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[66] F. Clautiaux, C. Alves, and J.M. Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179:317–342, 2010.

[67] F. Clautiaux, C. Alves, J.M. Valério de Carvalho, and J. Rietz. New stabilization procedures for the cutting stock problem. *INFORMS Journal on Computing*, 23:530–545, 2011.

[68] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35:944–959, 2008.

[69] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54:756–766, 2006.

[70] J. Coelho and M. Vanhoucke. The multi-mode resource-constrained project scheduling problem. In C. Schwindt and J. Zimmermann, editors, *Handbook on Project Management and Scheduling Vol.1*, pages 491–511. Springer International Publishing, Cham, 2015.

[71] E. G. Coffman, Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization.* Kluwer Academic Publishers, Boston, MA, 1999.

[72] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, 1997.

[73] E.G. Coffman Jr. and J. Csirik. A classification scheme for bin packing theory. *Acta Cybernetica*, 18:47–60, 2007.

[74] E.G. Coffman Jr. and J. Csirik. Performance guarantees for one-dimensional bin packing. In T.F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, chapter 32, pages 1–18. Chapman & Hall, 2007.

[75] E.G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Survey and classification. In P.M. Pardalos, D.-Z. Du, and R.L. Graham, editors, *Handbook of Combinatorial Optimization.* Springer New York, 2013.

[76] E.G. Coffman Jr., J. Csirik, D.S. Johnson, and G.J. Woeginger. An introduction to bin packing. Unpublished manuscript, available at `https://www.inf.u-szeged.hu/~csirik/ed5ut.pdf`, 2004.

[77] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing - an updated survey. In G. Ausiello, M. Lucentini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49–106. Springer Vienna, 1984.

[78] J.R. Correa. Resource augmentation in two-dimensional packing with orthogonal rotations. *Operations Research Letters*, 34:85–93, 2006.

[79] I. Correia, L. Gouveia, and F. Saldanha da Gama. Solving the variable size bin packing problem with discretized formulations. *Computers & Operations Research*, 35:2103–2113, 2008.

[80] G. Costa, C. D'Ambrosio, and S. Martello. A free educational java framework for graph algorithms. *Journal of Computer Science*, 6:87–91, 2010.

[81] G. Costa, C. D'Ambrosio, and S. Martello. GraphsJ 3: A modern didactic application for graph algorithms. *Journal of Computer Science*, 10:1115–1119, 2014.

[82] G. Costa, M. Delorme, M. Iori, E. Malaguti, and S. Martello. A training software for orthogonal packing problems. Technical Report OR-17-4, DEI "Guglielmo Marconi", University of Bologna, Italy, 2017.

[83] J.F. Côté, M. Dell'Amico, and M. Iori. Combinatorial Benders' cuts for the strip packing problem. *Operations Research*, 62:643–661, 2014.

[84] J.F. Côté and M. Iori. The meet-in-the-middle principle for cutting and packing problems. Technical Report CIRRELT-2016-28, CIRRELT, Montreal, Canada, 2016.

[85] T.G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. Computing the asymptotic worst-case of bin packing lower bounds. *European Journal of Operational Research*, 183:1295–1303, 2007.

[86] T.G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. New bin packing fast lower bounds. *Computers & Operations Research*, 34:3439–3457, 2007.

[87] T.G. Crainic, G. Perboli, W. Rei, and R. Tadei. Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers & Operations Research*, 38:1474–1482, 2011.

[88] L.L. Crumpton and E.L. Harden. Using virtual reality as a tool to enhance classroom instruction. *Computers & Industrial Engineering*, 33:217–220, 1997.

[89] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8:101–111, 1960.

[90] P. De, E. J. Dunne, J.B. Ghosh, and C.E. Wells. Complexity of the discrete time-cost tradeoff problem for project networks. *Operations Research*, 45:302–306, 1997.

[91] G. Değirmenci and M. Azizoğlu. Branch and bound based solution algorithms for the budget constrained discrete time/cost trade-off problem. *Journal of the Operational Research Society*, 64:1474–1484, 2013.

[92] Z. Degraeve and M. Peeters. Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS Journal on Computing*, 15:58–81, 2003.

[93] Z. Degraeve and L. Schrage. Optimal integer solutions to industrial cutting stock problems. *INFORMS Journal on Computing*, 11:406–419, 1999.

[94] M. Dell'Amico, M. Iori, S. Martello, and M. Monaci. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, 20(3):333 – 344, 2008.

[95] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[96] M. Delorme and M. Iori. Pseudo-polynomial formulations for bin packing and cutting stock problems. Column Generation 2016, Búzios (Brazil), 2016. Available on line at `https://www.gerad.ca/colloques/ColumnGeneration2016/PDF/Iori.pdf` (last accessed: April 2017).

[97] M. Delorme and M Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. Technical report, DEI "Guglielmo Marconi", University of Bologna, Italy, 2017.

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

[99] M. Delorme, M. Iori, and S. Martello. Bpplib: A library for bin packing and cutting stock problems. Technical report, DEI "Guglielmo Marconi", University of Bologna, Italy, 2017.

[100] M. Delorme, M. Iori, and S. Martello. Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, 78:290–298, 2017.

[101] E. Demeulemeester, B. De Reyck, B. Foubert, W. Herroelen, and M Vanhoucke. New computational results on the discrete time/cost trade-off problem in project networks. *Journal of the Operational Research Society*, 49:1153–1163, 1998.

[102] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

[103] E. Demeulemeester and W. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485–1492, 1997.

[104] G. Desaulniers, J. Desrosiers, and S. Spoorendonk. Cutting planes for branch-and-price algorithms. *Networks*, 58:301–310, 2011.

[105] U. Dorndorf, E. Pesch, and T. Phan-Huy. A branch-and-bound algorithm for the resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52:413–439, 2000.

[106] G. Dósa, R. Li, X. Han, and Z. Tuza. Tight absolute bound for First Fit Decreasing bin-packing: FFD (l) $\leq$ 11/9opt (l) + 6/9. *Theoretical Computer Science*, 510:13–61, 2013.

[107] G. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science – STACS 2013*, volume 20, pages 538–549, 2013.

[108] G. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *Automata, Languages, and Programming*, volume 8572 of *Lecture Notes in Computer Science*, pages 429–441. Springer Berlin Heidelberg, 2014.

[109] K.A. Dowsland. An exact algorithm for the pallet loading problem. *European Journal of Operational Research*, 31:78–84, 1987.

[110] J. Dupuis, P. Schaus, and Y. Deville. Consistency check for the bin packing constraint revisited. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 117–122. Springer Berlin Heidelberg, 2010.

[111] H. Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104, 1981.

[112] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[113] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution.* Physica-Verlag, Heidelberg, 1992.

[114] S. Eilon and N. Christofides. The loading problem. *Management Science*, 17:259–268, 1971.

[115] K. Eisemann. The trim problem. *Management Science*, 3:279–284, 1957.

[116] F. Eisenbrand, D. Pálvölgyi, and T. Rothvoß. Bin packing via discrepancy of permutations. *ACM Transactions on Algorithms (TALG)*, 9:1–15, 2013.

[117] S. Elhedhli. Ranking lower bounds for the bin packing problem. *European Journal of Operational Research*, 160:34–46, 2005.

[118] S. Elhedhli and F. Gzara. Characterizing the optimality gap and the optimal packings for the bin packing problem. *Optimization Letters*, 9:209–223, 2015.

[119] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *J. of Heuristics*, 2:5–30, 1996.

[120] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE international Conference on Robotics and Automation – ICRA 1992*, pages 1186–1192. IEEE, 1992.

[121] A.A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38:922–923, 1990.

[122] S.P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31, 2001.

[123] K. Fleszar and C. Charalambous. Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. *European Journal of Operational Research*, 210:176–184, 2011.

[124] K. Fleszar and K.S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29:821–839, 2002.

[125] L.R. Ford Jr. and D.R. Fulkerson. A suggested computation for maximal multicommodity network flows. *Management Science*, 5:97–101, 1958.

[126] H.A. Friberg. CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation*, 8:191–214, 2016.

[127] F. Furini, E. Malaguti, and D. Thomopulos. Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, 28:736–751, 2016.

[128] V. Gabrel and M. Minoux. A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. *Operations Research Letters*, 30:252–264, 2002.

[129] M. Gardner. Mathematical games: the problem of Mrs. Perkins' quilt, and answers to last month's puzzles. *Scientific American*, 215:264–272, 1966.

[130] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* Freeman, New York, 1979.

[131] M.R. Garey and D.S. Johnson. Approximation algorithms for bin-packing problems: A survey. In G. Ausiello and Lucertini, editors, *Analysis and Design of Algorithms in Combinatorial Optimization*, pages 147–172. Springer Vienna, 1981.

[132] T. Gau and G. Wäscher. CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:572–579, 1995.

[133] I.P. Gent. Heuristic solution of open bin packing problems. *J. of Heuristics*, 3:299–304, 1998.

[134] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[135] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 11:863–888, 1963.

[136] P.C. Gilmore and R.E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[137] P. Gómez-Meneses and M. Randall. A hybrid extremal optimisation approach for the bin packing problem. In *Artificial Life: Borrowing from Biology – ACAL 2009*, volume 5865 of *Lecture Notes in Computer Science*, pages 242–251. Springer-Verlag Berlin Heidelberg, 2009.

[138] C. Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operational Research*, 44:197–208, 1990.

[139] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In E.L. Johnson P.L. Hammer and B.H. Korte, editors, *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.

[140] M.C.N. Gramani, P.M. França, and M.N. Arenales. A lagrangian relaxation approach to a coupled lot-sizing and cutting stock problem. *International Journal of Production Economics*, 119(2):219 – 227, 2009.

[141] C. Groër, B. Golden, and E. Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2:79–101, 2010.

[142] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, 28:175–194, 2016.

[143] J.N.D. Gupta and J.C. Ho. A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning and Control*, 10:598–603, 1999.

[144] E. Hadjiconstantinou and E. Klerides. A new path-based cutting plane approach for the discrete time-cost tradeoff problem. *Computational Management Science*, 7:313–336, 2010.

[145] R.W. Haessler. Controlling cutting pattern changes in one dimensional trim problems. *Operations Research*, 23:483–493, 1975.

[146] R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[147] M. Haouari and A. Gharbi. Fast lifting procedures for the bin packing problem. *Discrete Optimization*, 2:201–218, 2005.

[148] M. Haouari and M. Serairi. Heuristics for the variable sized bin-packing problem. *Computers & Operations Research*, 36:2877–2884, 2009.

[149] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207:1–14, 2010.

[150] H. Hashimoto, Y. Hu, S. Imahori, and M. Yagiura. Private communication, 2015.

[151] Ö. Hazır, M. Haouari, and E. Erel. Discrete time/cost trade-off problem: A decomposition-based solution algorithm for the budget version. *Computers & Operations Research*, 37:649 – 655, 2010.

[152] V. Hemmelmayr, V. Schmid, and C. Blum. Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research*, 39:1097–1108, 2012.

[153] J.C Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[154] O. Holthaus. Decomposition approaches for solving the integer one-dimensional cutting stock problem with different types of standard lengths. *European Journal of Operational Research*, 141:295–312, 2002.

[155] J.N Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction.* John Wiley & Sons, 2000.

[156] J.N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55:588–602, 2007.

[157] J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[158] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128:34–57, 2001.

[159] S. Hu, S. Wang, Y. Kao, T. Ito, and X. Sun. A branch and bound algorithm for project scheduling problem with spatial resource constraints. *Mathematical Problems in Engineering*, 2015:9, 2015.

[160] M. Iori, S. Martello, and M. Monaci. Metaheuristic algorithms for the strip packing problem. In P. Pardalos and V. Korotkich, editors, *Optimization and Industry: New Frontiers*, pages 159–179. Kluwer, Boston, 2003.

[161] M. Iori, J. J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253 – 264, 2007.

[162] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[163] B. Jarboui, S. Ibrahim, and A. Rebai. A new destructive bounding scheme for the bin packing problem. *Annals of Operations Research*, 179:187–202, 2010.

[164] D.S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

[165] J. Kallrath, S. Rebennack, J. Kallrath, and R. Kusche. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. *European Journal of Operational Research*, 238:374 – 389, 2014.

[166] T. Kämpke. Simulated annealing: use of a new tool in bin packing. *Annals of Operations Research*, 16:327–332, 1988.

[167] L.V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science, English translation of a 1939 paper written in Russian*, 6:366–422, 1960.

[168] K. Kaparis and A.N. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124:69–91, 2010.

[169] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science – SFCS 1982*, pages 312–320. IEEE, 1982.

[170] V.M. Kartak. Sufficient conditions for the integer round-up property to be violated for the linear cutting stock problem. *Automation and Remote Control*, 65:407–412, 2004.

[171] V.M. Kartak, A.V. Ripatti, G. Scheithauer, and S. Kurz. Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 187:120–129, 2015.

[172] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198:73–83, 2009.

[173] B.I. Kim and J. Wy. Last two fit augmentation to the well-known construction heuristics for one-dimensional bin-packing problem: an empirical study. *The International Journal of Advanced Manufacturing Technology*, 50:1145–1152, 2010.

[174] K.C. Kiwiel. An inexact bundle approach to cutting-stock problems. *INFORMS Journal on Computing*, 22:131–143, 2010.

[175] R. Klein and A. Scholl. Scattered branch and bound: an adaptive search strategy applied to resource-constrained project scheduling. *Central European Journal of Operations Research*, 7:177–201, 1999.

[176] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3:103, 2011.

[177] R. Kolisch and A. Sprecher. Psplib - a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1997.

[178] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38:3–13, 2011.

[179] R.E. Korf. A new algorithm for optimal bin packing. In *Proceedings of the Eighteenth National Conference on Articial Intelligence – AAAI 2002*, pages 731–736. AAAI Press, 2002.

[180] R.E. Korf. An improved algorithm for optimal bin packing. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence – IJCAI 2003*, pages 1252–1258. Morgan Kaufmann Publishers Inc., 2003.

[181] R.E. Korf, M.D. Moffitt, and M.E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179:261–295, 2010.

[182] M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39:616–622, 1991.

[183] P. Laborie. Complete MCS-based search: Application to resource constrained project scheduling. In *International joint conferences on artificial intelligence*, pages 181–186, 2005.

[184] P. Laborie. IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In W.-J. van Hoeve and J.N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings*, pages 148–162. Springer, Berlin, Heidelberg, 2009.

[185] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, 90:7–14, 2004.

[186] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55:705–716, 2004.

[187] R. Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research*, 36:2295–2310, 2009.

[188] K.H. Liang, X. Yao, C. Newton, and D. Hoffman. A new evolutionary approach to cutting stock problems with and without contiguity. *Computers & Operations Research*, 29:1641–1659, 2002.

[189] O. Liess and P. Michelon. A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operations Research*, 157:25–36, 2007.

[190] L. Lins, S. Lins, and R. Morabito. An l-approach for packing (l,w)-rectangles into rectangular and l-shaped pieces. *Journal of the Operational Research Society*, 54:777–789, 2003.

[191] F. Llaugel and S. Confesor. Computer-aided statistical quality control learning. *Computers & Industrial Engineering*, 33:125–128, 1997.

[192] A. Lodi. Mixed integer programming computation. In M. Jünger, T. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G.Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer-Verlag, 2009.

[193] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[194] A. Lodi, S. Martello, M. Monaci, and D. Vigo. Two-dimensional bin packing problems. *Paradigms of Combinatorial Optimization: Problems and New Approaches, Volume 2*, pages 107–129, 2010.

[195] A. Lodi, S. Martello, M. Monaci, and D. Vigo. Two-dimensional bin packing problems. In V. Th. Paschos, editor, *Paradigms of Combinatorial Optimization: Problems and New Approaches*, chapter 5, pages 107–129. ISTE and John Wiley & Sons, 2014.

[196] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[197] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.

[198] A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional Knapsack problems. *Mathematical Programming, Series B*, 94(2):257 – 278, 2003.

[199] K.H Loh, B. Golden, and E. Wasil. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35:2283–2291, 2008.

[200] E. López-Camacho, H. Terashima-Marín, and P. Ross. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation – GECCO 2011*, pages 257–258. ACM, 2011.

[201] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363 – 397. Springer US, 2010.

[202] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2005.

[203] G.S. Lueker. Bin packing with items uniformly distributed over intervals $[a, b]$. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science SFCS 1983*, pages 289–297. IEEE, 1983.

[204] J. Lysgaard. CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Technical report, Aarhus School of Business, Denmark, 2003.

[205] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004.

[206] R. Macedo, C. Alves, and J.M. Valério de Carvalho. Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37:991 – 1001, 2010.

[207] J.N. MacGregor and Y. Chu. Human performance on the traveling salesman and related problems: A review. *The Journal of Problem Solving*, 3:1–19, 2011.

[208] J.N. MacGregor and T. Ormerod. Human performance on the traveling salesman problem. *Perception & Psychophysics*, 58:527–539, 1996.

[209] O. Marcotte. The cutting stock problem and integer rounding. *Mathematical Programming*, 33:82–92, 1985.

[210] O. Marcotte. An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4:239–243, 1986.

[211] S. Martello and M. Monaci. Models and algorithms for packing rectangles into the smallest square. *Computers & Operations Research*, 63:161–171, 2015.

[212] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15:310–319, 2003.

[213] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45:414–424, 1999.

[214] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990. available on line at `www.or.deis.unibo.it`.

[215] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[216] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[217] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.

[218] H. Miyata, S. Watanabe, and Y. Minagawa. Performance of young children on traveling salesperson navigation tasks presented on a touch screen. *PLOS ONE*, 9:1–19, 2014.

[219] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097 – 1100, 1997.

[220] M. Monacci. *Algorithms for packing and scheduling problems.* PhD thesis, Universit di Bologna, 2002.

[221] M. Mrad. An arc flow-based optimization approach for the two-stage guillotine strip cutting problem. *Journal of the Operational Research Society*, 66:1850–1859, 2015.

[222] E.A. Mukhacheva, G.N. Belov, V.M. Kartack, and A.S. Mukhacheva. Linear one-dimensional cutting-packing problems: numerical experiments with the sequential value correction method (SVC) and a modified branch-and-bound method (MBB). *Pesquisa Operacional*, 20:153–168, 2000.

[223] V. Nesello, M. Delorme, M. Iori, and A. Subramanian. Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society*, 2017. To appear.

[224] A. Newman, O. Neiman, and A. Nikolov. Beck's three permutations conjecture: A counterexample and some consequences. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science FOCS 2012*, pages 253–262. IEEE, 2012.

[225] C. Nitsche, G. Scheithauer, and J. Terno. Tighter relaxations for the cutting stock problem. *European Journal of Operational Research*, 112:654–663, 1999.

[226] T. Osogami and H. Okano. Local search algorithms for the bin packing problem and their relationships to various construction heuristics. *J. of Heuristics*, 9:29–49, 2003.

[227] E. Özcan, Z. Kai, and J.H. Drake. Bidirectional best-fit heuristic considering compound placement for two dimensional orthogonal rectangular strip packing. *Expert Systems with Applications*, 40:4035–4043, 2013.

[228] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[229] C. Picouleau. Worst-case analysis of fast heuristics for packing squares into a square. *Theoretical Computer Science*, 164:59–72, 1996.

[230] R. Poli, J. Woodward, and E.K. Burke. A histogram-matching approach to the evolution of bin-packing strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC 2007*, pages 3500–3507. IEEE, 2007.

[231] A.A.B. Pritsker and L.J. Watters. A zero-one programming approach to scheduling with limited resources. Technical report, RAND Corporation, RM-5561-PR, 1968.

[232] A.A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–108, 1969.

[233] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez S., H. Fraire Huacuja, and A. Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015.

[234] M.R. Rao. On the cutting stock problem. *Journal of the Computer Society of India*, 7:35–39, 1976.

[235] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[236] J. Rietz and S. Dempe. Large gaps in one-dimensional cutting stock problems. *Discrete Applied Mathematics*, 156:1929–1935, 2008.

[237] P. Rohlfshagen and J.A. Bullinaria. A genetic algorithm with exon shuffling crossover for hard bin packing problems. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation – GECCO 2007*, pages 1365–1371. ACM, 2007.

[238] P. Rohlfshagen and J.A. Bullinaria. Nature inspired genetic algorithms for hard packing problems. *Annals of Operations Research*, 179:393–419, 2010.

[239] G.M. Roodman. Near optimal solutions to one-dimensional cutting stock problem. *Computers & Operations Research*, 13:713–719, 1986.

[240] P. Ross, J.G. Marín-Blázquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 1295–1306. Springer Berlin Heidelberg, 2003.

[241] P. Ross, S. Schulenburg, J.G. Marín-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2002*, pages 942–948. Morgan Kaufmann Publishers Inc, 2002.

[242] T. Rothvoß. Approximating Bin Packing within O(log OPT * log log OPT) bins. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2013*, pages 20–29. IEEE, 2013.

[243] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland, 1981.

[244] R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal Heuristics for Branch-and-Price: the assets of diving methods. working paper or preprint, 2016.

[245] P. Schaus, J.-C. Régin, R. Van Schären, W. Dullärt, and B. Raa. Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem. In *Principles and Practice of Constraint Programming*, volume 7514 of *Lecture Notes in Computer Science*, pages 815–822. Springer Berlin Heidelberg, 2012.

[246] G. Scheithauer and J. Terno. A branch-and-bound algorithm for solving one-dimensional cutting stock problems exactly. *Applicationes Mathematicae*, 23:151–167, 1995.

[247] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:562–571, 1995.

[248] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20:93–100, 1997.

[249] G. Scheithauer, J. Terno, A. Müller, and G. Belov. Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm. *Journal of the Operational Research Society*, 52:1390–1401, 2001.

[250] J.E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, Huntsville, Alabama, USA, 2002.

[251] A. Scholl, R. Klein, and C. Jürgens. Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24:627–645, 1997.

[252] E.L. Schreiber and R.E. Korf. Improved bin completion for optimal bin packing and number partitioning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence – IJCAI 2013*, pages 651–658. AAAI Press, 2013.

[253] A. Schutt, T. Feydy, P.J. Stuckey, and M.G. Wallace. Explaining the cumulative propagator. *Constraints*, 16:250–282, 2010.

[254] P. Schwerin and G. Wäscher. The bin-packing problem: a problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4:377–389, 1997.

[255] P. Schwerin and G. Wäscher. A new lower bound for the bin-packing problem and its integration into mtp. *Pesquisa Operacional*, 19:111–129, 1999.

[256] J.F. Shapiro. Dynamic programming algorithms for the integer programming problem-I: The integer programming problem viewed as a knapsack type problem. *Operations Research*, 16:103–121, 1968.

[257] P. Shaw. A constraint for bin packing. In *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer Berlin Heidelberg, 2004.

[258] E. Silva, F. Alvelos, and J. M. Valério de Carvalho. Integrating two-dimensional cutting stock and lot-sizing problems. *Journal of the Operational Research Society*, 65(1):108–123, 2014.

[259] E. Silva, F. Filipe Alvelos, and J.M. Valério de Carvalho. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699 – 708, 2010.

[260] E. Silva, J.F. Oliveira, and G. Wäscher. The pallet loading problem: a review of solution methods and computational experiments. *International Transactions in Operational Research*, 23:147–172, 2016.

[261] K. Sim and E. Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation GECCO 2013*, pages 1549–1556. ACM, 2013.

[262] K. Sim, E. Hart, and B. Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg, 2012.

[263] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41:579, 1994.

[264] A. Singh and A.K. Gupta. Two heuristics for the one-dimensional bin-packing problem. *OR Spectrum*, 29:765–781, 2007.

[265] A. Sprecher. Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46:710–723, 2000.

[266] H. Stadtler. A comparison of two optimization procedures for 1-and 1 1/2-dimensional cutting stock problems. *Operations-Research-Spektrum*, 10:97–111, 1988.

[267] A. Stawowy. Evolutionary based heuristic for bin packing problem. *Computers & Industrial Engineering*, 55:465–474, 2008.

[268] A. Subramanian. *Heuristic Exact and Hybrid Approaches for Vehicle Routing Problems.* PhD thesis, Universidade Federal Fluminense, 2012.

[269] A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi, and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery.

Computers & Operations Research, 37:1899 – 1911, 2010. Metaheuristics for Logistics and Vehicle Routing.

[270] P.E. Sweeney and E.R. Paternoster. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

[271] F. B. Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28:1197–1210, 1982.

[272] J. Thomas and N.S. Chaudhari. A new metaheuristic genetic-based placement algorithm for 2D strip packing. *Journal of Industrial Engineering International*, 10:1–16, 2014.

[273] E.S. Thorsteinsson. Branch and check: A hybrid framework integrating mixed integer programming and constraint programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP2001)*, pages 16–30. Springer-Verlag, Berlin, 2001.

[274] M.A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118:73–84, 2003.

[275] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257:845–858, 2017.

[276] Ö. Ülker, E.E. Korkmaz, and E. Özcan. A grouping genetic algorithm using linear linkage encoding for bin packing. In *Parallel Problem Solving from Nature – PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 1140–1149. Springer Berlin Heidelberg, 2008.

[277] R. Vahrenkamp. Random search in the one-dimensional cutting stock problem. *European Journal of Operational Research*, 95:191–200, 1996.

[278] J.M. Valério de Carvalho. Exact solution of bin packing problems using column generation and branch and bound. *Annals of Operations Research*, 86:629–659, 1999.

[279] J.M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

[280] J.M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17:175–182, 2005.

[281] P.H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9:211–228, 1998.

[282] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

[283] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86:565–594, 1999.

[284] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48:111–128, 2000.

[285] F. Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. *Management Science*, 47:864 – 879, 2001.

[286] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130:249–294, 2011.

[287] F. Vanderbeck and L.A. Wolsey. Reformulation and decomposition of integer programs. In M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[288] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: a computational study. *Operations-Research-Spektrum*, 18:131–144, 1996.

[289] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.

[290] J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra. Project scheduling with finite or infinite number of activity processing modes a survey. *European Journal of Operational Research*, 208:177–205, 2011.

[291] L. Wei, W.C. Oon, W. Zhu, and A. Lim. A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research*, 215:337–346, 2011.

[292] J. Westerlund, L.G. Papageorgiou, and T. Westerlund. A MILP model for N-dimensional allocation. *Computers & Chemical Engineering*, 31:1702–1714, 2007.

[293] L.A. Wolsey. Valid inequalities, covering problems and discrete dynamic programs. *Annals of Discrete Mathematics*, 1:527–538, 1977.