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

Maxence Delorme, Manuel Iori, Silvano Martello

PII: S0377-2217(16)30249-1
DOI: 10.1016/j.ejor.2016.04.030
Reference: EOR 13651

To appear in: *European Journal of Operational Research*

Received date: 26 March 2015
Revised date: 28 March 2016
Accepted date: 16 April 2016

Please cite this article as: Maxence Delorme, Manuel Iori, Silvano Martello, Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms, *European Journal of Operational Research* (2016), doi: 10.1016/j.ejor.2016.04.030

Highlights

- Bin packing and cutting stock problems
- State of the art of mathematical models and exact approaches
- Software and computational experiments

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

Maxence Delorme[(1)], Manuel Iori[(2)], Silvano Martello[(1)]

(1) DEI "Guglielmo Marconi", University of Bologna
(2) DISMI, University of Modena and Reggio Emilia

Corresponding author `silvano.martello@unibo.it`, phone/fax +39 051 2093022/3073

Abstract

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.

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 [82]. In 1961 Gilmore and Gomory [69] introduced, for this class of problems, the concept of column generation, by deriving it from earlier ideas of Ford and Fulkerson [63] and Dantzig and Wolfe [40]. 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 given n *items*, each having an integer *weight* w_j $(j = 1, \ldots, n)$, 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 $(j = 1, \ldots, m)$, 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 paper 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 [141], who collected more than 400 books, articles, dissertations, and working papers appeared from 1961 to 1990. In 1990 Dyckhoff [50] 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 [107] on knapsack problems. Two years later Dyckhoff and Finke [51] 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. [34]. More recently, Wäscher et al. [154] re-visited the typology by Dyckhoff [50] 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 [66] and Coffman et al. [36] 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. [37, 35] and Coffman and Csirik [32]. Coffman and Csirik [31] 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. [33] presented an overview of approximation algorithms for the BPP and a number of its variants, and classified all references according to [31].

Valério de Carvalho [146] 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 [10] 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 [76] 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. [96, 97, 98].

In the next section we provide a formal definition of the BPP and the CSP. In Section 3 we briefly review the most successful upper and lower bounding techniques for the considered problems. In Sections 4, 5, and 6 we examine pseudo-polynomial formulations, enumeration algorithms, and branch-and-price approaches, respectively. Finally, in Section 7, we experimentally evaluate the computational performance of twelve computer programs available for the solution of the considered problems. Conclusions follow in Section 8.

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 [107])

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

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad (i = 1, \ldots, u), \tag{2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \qquad (j = 1, \ldots, n), \tag{3}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad (i = 1, \ldots, u; j = 1, \ldots, n). \tag{5}$$

Constraints (2) impose that the capacity of any used bin is not exceeded, while constraints (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

$$\min \quad \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \leq c y_i \quad (i = 1, \ldots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \quad (j = 1, \ldots, m), \tag{8}$$

$$y_i \in \{0, 1\} \quad (i = 1, \ldots, u), \tag{9}$$

$$\xi_{ij} \geq 0, \text{integer} \quad (i = 1, \ldots, u; j = 1, \ldots, m). \tag{10}$$

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 [65] through transformation from the 3-Partition problem.

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 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 $A(I)/OPT(I) \leq \overline{r}(A)$ 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.,

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

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 [46, 47] 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 [137]). 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 $S = \{w_1, \ldots, w_n\}$ 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 $(w_1, \ldots, w_n)$ 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. □

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 1. The most recent survey (2013), by Coffman et al. [33], examines 200 references from the literature. Among the papers that appeared subsequently, we mention those by Dósa et al. [45] on the FFD algorithm, by Rothvoß [122], who improved a classical result by Karmarkar and Karp [84], and by Balogh et al. [7], who closed a long standing open issue on on-line bin packing.

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

$$L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil , \tag{11}$$

which gives the rounded solution value of the linear programming relaxation of (1)-(5). It is easily seen that $\underline{r}(L_1) = \frac{1}{2}$ (see, e.g., Martello and Toth [107]).

A better lower bound was obtained by Martello and Toth [106]. 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

$$L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq c/2, \alpha \text{ integer}\} \tag{12}$$

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. [91] (lower bound L_{2LLM}), and by Chen and Srivastava [27]. Theoretical properties of such bounds were studied by Elhedhli [55]. Bourjolly and Rebetez [18] proved that the asymptotic WCPR of the bound L_{2LLM} proposed in [91] is $\underline{r}^{\infty}(L_{2LLM}) = \frac{3}{4}$.

Another lower bound, L_3, dominating L_2 was obtained by Martello and Toth [106] 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. [38].

A different type of lower bound computation had been considered in the eighties by Lueker [102], 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 [80]. Consider the normalized definition of the BPP (see Section 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 [60] 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. [29] and Alves et al. [2] for recent surveys on these functions and their use for the computation of BPP lower bounds.

Chao et al. [26] and Crainic et al. [39] 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 [43], Alvim et al. [3], Haouari and Gharbi [77], and Jarboui et al. [79].

Other effective lower bounds, which however require a non-polynomial time, including the famous Gilmore-Gomory column generation method, are discussed in Section 6.

3.3 Heuristics and metaheuristics

The focus of this work 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 [52] who presented a heuristic for a number of packing problems, basically consisting of algorithm BFD (see Section 3.1), plus a reshuffle routine when the solution is not equal to the continuous relaxation L_1. Roodman [119] presented a set of heuristics for variants of the CSP, mainly based on an initial greedy solution improved through local search. Vahrenkamp [144] proposed a random search for the CSP, based on a heuristic developed by Haessler [75] for generating cutting patterns for trim problems. Wäscher and Gau [153] 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 [69] model (see Section 6). The experiments were performed on random instances produced by their generator, CUTGEN (see Gau and Wäscher [67]), which creates CSP instances depending on five parameters: number of item types, minimum and maximum weight, bin capacity, and average demand.

Gupta and Ho [74] 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. [108] presented a modified FFD algorithm which was later embedded in the exact algorithm by Belov and Scheithauer [12] (see Section 6.3). Osogami and Okano [111] 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. [17], Kim and Wy [87],

and Fleszar and Charalambous [61]. The effectiveness of a hill climbing local search strategy for the BPP, also based on item exchanges, was later investigated by Lewis [93].

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 [81], while a variant of the method (called weight annealing) was proposed by Loh et al. [99]. Scholl et al. [130] used a Tabu search procedure to speed up their well-known exact algorithm (BISON) for the BPP, treated in Section 5.1. Alvim et al. [3] 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 [58]: 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 [57] improved this method through hybridization with the dominance criterion by Martello and Toth [106] (see Section 5.1), and proposed a set of benchmark instances that was later adopted by many authors for computationally testing BPP algorithms. Although Gent [68] showed that the majority of them are very easy, these instances were used, e.g., for testing the genetic approaches by Reeves [115], Bhatia and Basu [16], Singh and Gupta [138], Ülker et al. [143], and Stawowy [140]. Other genetic algorithms were proposed by Poli et al. [112] and by Rohlfshagen and Bullinaria [117, 118]. Recently, a very effective genetic algorithm was proposed by Quiroz-Castellanos et al. [113].

Levine and Ducatelle [92] used an ant colony approach combined with a local search to solve the BPP. Liang et al. [94] proposed an evolutionary programming algorithm for the CSP and some of its variants.

Hyper-heuristics

Ross et al. [120, 121] 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. [100], Sim et al. [136], and Burke et al. [21].

Bai et al. [5] tested on BPP instances their simulated annealing hyper-heuristic approach. Sim and Heart [135] used genetic programming as a generative hyper-heuristic to create deterministic heuristics.

Other meta-heuristic approaches

Fleszar and Hindi [62] obtained new heuristics for the BPP by modifying the heuristic of Gupta and Ho [74] and proposed a variable neighborhood search algorithm. Gómez-Meneses and Randall [72] proposed a hybrid extremal optimization approach with local search for the BPP.

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.

4.1 Considerations on the basic ILP model

The textbook BPP model (1)–(5), which has its roots in the seminal work by Kantorovich [82], was formally defined in 1990 by Martello and Toth [107]. It involves a polynomial number of variables and constraints but is not very efficient in practice, as shown in Section 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 $i = 1, \ldots, u-1$ 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 [64] and Kaparis and Letchford [83].

Despite these results, the computational behavior of model (1)–(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 (1)–(5).

In Section 4.2 we address the oldest such model, independently developed by Rao [114] in 1976 and by Dyckhoff [49] few years later. The most relevant approach of this kind (somehow anticipated by Wolsey [155] in 1977) was presented in 1999 by Valério de Carvalho [145] for the CSP. In 2010, Cambazard and O'Sullivan [22] presented a BPP pseudo-polynomial model based on a similar idea, but described in a form inspired by the graph construction used by Trick [142] for propagating knapsack constraints. We anticipate its description in Section 4.3, since this makes it easier to understand the Valério de Carvalho model, which is then discussed in Section 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.* □

4.2 One-cut formulation

The idea behind the Rao [114] and Dyckhoff [49] 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 $W = \{w_1, w_2, \ldots, w_m\}$ 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,

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

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

$$\min \quad \sum_{q \in W} x_{cq} \tag{13}$$

$$\text{s.t.} \quad \sum_{p \in A(q)} x_{pq} + \sum_{p \in B(q)} x_{p+q,p} \geq L_q + \sum_{r \in C(q)} x_{qr} \quad q \in (W \cup R) \backslash \{c\}, \tag{14}$$

$$x_{pq} \geq 0 \text{ and integer} \qquad p \in R, q \in W, p > q. \tag{15}$$

The objective function (13) minimizes the number of times an item is cut from a bin. Constraints (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 (13)-(15) has $O(mc)$ variables and $O(c)$ constraints.

Stadtler [139] 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)".

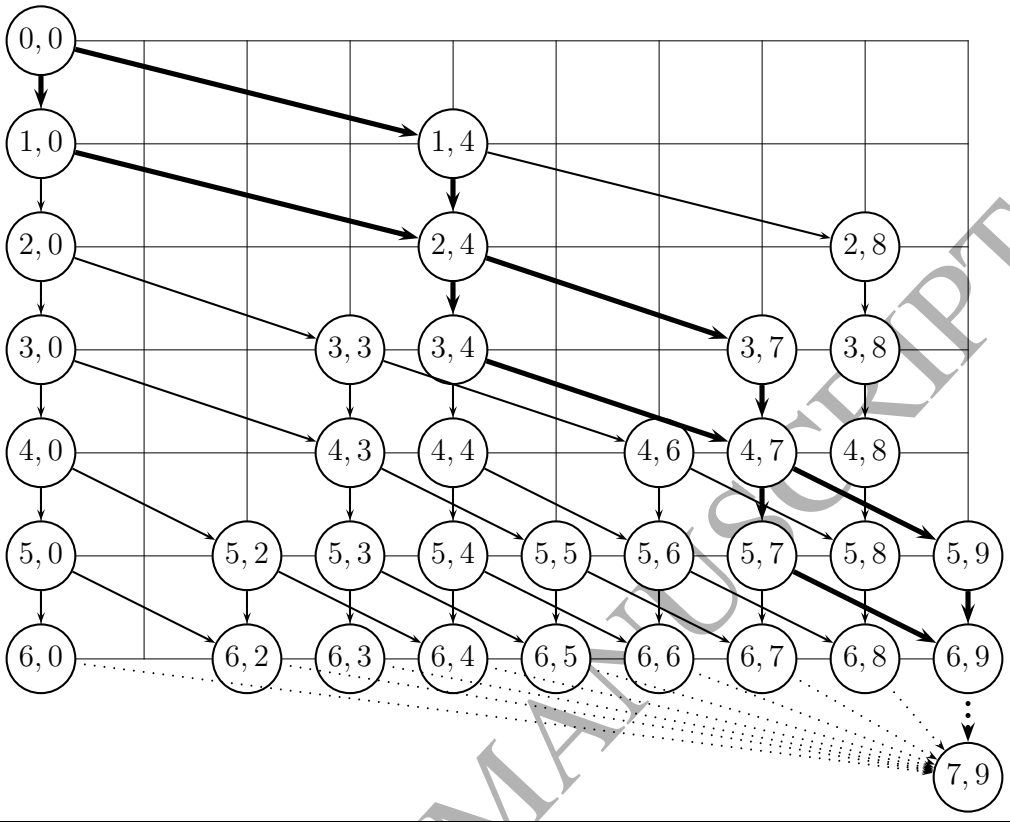
4.3 DP-flow formulation

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 [22], 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 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

Figure 1 DP-flow graph construction for Example 1



(resp. emanating from) state (j, d). The BPP can be then modeled as

$$\min \quad z \tag{16}$$

$$\text{s.t.} \quad \sum_{((j,d),(j+1,e))\in\delta^+((j,d))} x_{j,d,j+1,e} - \sum_{((j-1,e),(j,d))\in\delta^-((j,d))} x_{j-1,e,j,d} = \begin{cases} z & \text{if } (j,d) = (0,0); \\ -z & \text{if } (j,d) = (n+1,c); \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

$$\sum_{((j-1,d),(j,d+w_j))\in A} x_{j-1,d,j,d+w_j} = 1 \qquad (j = 1,\ldots,n), \tag{18}$$

$$x_{j,d,j+1,e} \geq 0 \text{ and integer} \qquad ((j,d),(j+1,e)) \in A. \tag{19}$$

The objective function (16) minimizes the number of bins. Constraints (17) impose the flow (number of bins) conservation at all nodes, while constraints (18) ensure that each item is packed exactly once. Note that a "$\geq$" sign could be used in (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 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)]. □

The DP-flow model (16)-(19) has $O(nc)$ variables and constraints. This formulation was developed in [22] 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.

4.4 Arc-flow formulations

An effective CSP pseudo-polynomial formulation, denoted *arc-flow*, was presented by Valério de Carvalho [145], who used it in a branch-and-price algorithm (see Section 6). To make its comprehension easier, consider again Example 1, and the DP representation depicted in Figure 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 shows the counterpart of Figure 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:

$$\min \quad z \tag{20}$$

$$\text{s.t.} \quad -\sum_{(d,e)\in\delta^-(e)} x_{de} + \sum_{(e,f)\in\delta^+(e)} x_{ef} = \begin{cases} z & \text{if } e = 0; \\ -z & \text{for } e = c; \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

$$\sum_{(d,d+w_i)\in A'} x_{d,d+w_i} \geq b_i \qquad (i = 1,\dots,m), \tag{22}$$

$$x_{de} \geq 0 \text{ and integer} \qquad (d,e) \in A', \tag{23}$$

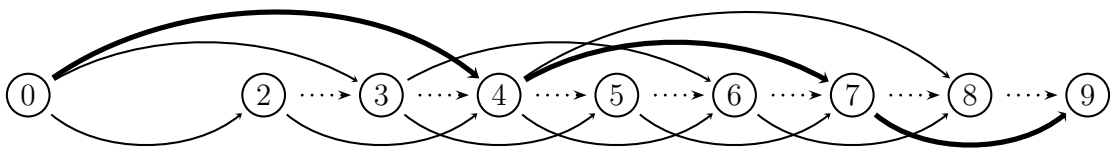
where $\delta^-(e)$ (resp. $\delta^+(e)$) denotes the set of arcs entering (resp. emanating from) e.

Constraints (21) impose the flow conservation at all nodes. Constraints (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.

Example 1 (resumed) *An optimal solution to the CSP instance consists of two identical paths* $[0,4,7,9]$*, highlighted in Figure 2.* □

The arc-flow model (20)-(23) has $O(mc)$ variables and $O(m+c)$ constraints. Valério de Carvalho [145] proposed however a number of improvements to the above basic model, aimed at reducing the number of arcs. For example (see again Figure 2), it is enough to only create

Figure 2 Arc-flow representation of the graph of Figure 1



nodes that correspond to feasible combinations of item weights. In addition, it is proved in [145] that the linear programming relaxation of (20)-(23) has the same solution value as the Gilmore and Gomory [69] model (see Section 6).

Very recently, Brandão and Pedroso [19] proposed an alternative CSP arc-flow formulation. They start with a multi-graph generalization of the arc-flow formulation by Valério de Carvalho [145] 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 (20)-(23) in which, however, those inequality constraints (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 7) proved to be very efficient on benchmark instances.

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 [53] and Gilmore and Gomory [69, 70, 71]). Starting from the early seventies, research in this field focused on branch-and-bound.

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 [52], who adapted the general enumerative scheme proposed by Balas [6] 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 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 [107]. 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 [106] proved the following

Property 3 *Given two distinct feasible sets F_1 and F_2, if there exists a partition $P = \{P_1, \ldots, P_\ell\}$ of F_2, and a subset $\{j_1, \ldots, j_\ell\}$ of F_1 such that $w_{j_h} \geq \sum_{k \in P_h} w_k$ for $h = 1, \ldots, \ell$, 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. [130] 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 [133] improved the competitiveness of MTP with respect to BISON through a lower bound provided by the column generation method developed by Gilmore and Gomory [69] (see Section 6) for the CSP.

In the early noughties Mukhacheva et al. [108] proposed a pattern oriented branch-and-bound algorithm for both the BPP and the CSP, while Korf [89, 90] proposed a "bin completion" algorithm (later improved on by Schreiber and Korf [131]) 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 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.

5.2 Constraint programming approaches

In the last decade some attempts have been proposed to solve the BPP through *Constraint Programming* (CP). Shaw [134] 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 [22] integrated pseudo-polynomial formulations discussed in Section 4 within the CP approach by Shaw. Dupuis et al. [48] used lower bound L_{2LLM} by Labbé et al. [91] and an additional reduction algorithm. Schaus et al. [124] introduced a filtering rule based on cardinality considerations.

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 [69, 70], 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 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.

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 $(a_{1p}, a_{2p}, \ldots, a_{mp})$, 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 $(j = 1, \ldots, m)$. 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

$$\min \quad \sum_{p \in P} y_p \tag{24}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \geq d_j \qquad (j = 1, \ldots, m), \tag{25}$$

$$y_p \geq 0 \text{ and integer } (p \in P). \tag{26}$$

Objective function (24) requires the minimization of the number of bins, whereas constraints (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 $(a_{1p}, a_{2p}, \ldots, a_{np})$, 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 (25) and (26) as $\sum_{p \in P} a_{jp} y_p \geq 1 \, (j = 1, \ldots, n)$ 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 (24)-(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

$$\min \quad \sum_{p \in P'} y_p \tag{27}$$

$$\text{s.t.} \quad \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \ldots, m), \tag{28}$$

$$y_p \geq 0 \qquad (p \in P'). \tag{29}$$

Once (27)-(29) has been solved, let π_j be the dual variable associated with the jth constraint (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):

$$\max \quad \sum_{j=1}^{m} \pi_j v_j, \tag{30}$$

$$\sum_{j=1}^{m} w_j v_j \leq c, \tag{31}$$

$$v_j \geq 0 \text{ and integer} \quad (j = 1, \ldots, m), \tag{32}$$

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 (30)-(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. [110], and Caprara and Monaci [24]. 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 (25). Indeed, if an optimal solution to model (24)-(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 [147] 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. [4], 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. [30] 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 6.3.

We conclude this section by mentioning some variants of column generation. Briant et al. [20] 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 [88] 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 [56] recently proposed another heuristic approach to the BPP, based on Lagrangian relaxation

and column generation.

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 [15], and Marcotte [104]), until it was disproved in the eighties. Marcotte [105] 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. [25] 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 [126, 127]) that $z_{opt} - \lceil L_{\text{LP}} \rceil \leq 1$ 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 [85] 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 [129] (see also Belov and Scheithauer [12]) created a set of hard instances, including some satisfying $z_{opt} - L_{\text{LP}} > 1$. Rietz and Dempe [116] presented methods to construct Non-IRUP instances through perturbations of the item weights that make certain cutting patterns infeasible. Caprara et al. [23] 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. [86] 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. [54] and Newman et al. [109] studied the relationship between the MIRUP conjecture for the BPP and the Becks three-permutation conjecture for discrepancy theory (see Beck and Sós [9]).

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 7.1 we discuss a method to generate Non-IRUP instances that are difficult to solve exactly.

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

When the solution obtained at the end of the column generation method of Section 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 (24)-(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 [73]). Other, non exact, methods simply use rounding heuristics (like, e.g., Roodman [119], Haessler and Sweeney [76], and Holthaus [78]), 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. [149] 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 [123] 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 [59]. The following year Scheithauer and Terno [125] 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 [148] focused on the CSP and showed that the application of the classical Dantzig–Wolfe [40] decomposition to the CSP model (6)-(10) leads to the set covering model (24)-(26), and used this result to implement two specifically tailored branching rules.

In the late nineties, Valério de Carvalho [145] proposed a column generation approach which is not based on the traditional Gilmore-Gomory model but on the arc-flow formulation of Section 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 [150] 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 [151, 152] tested on BPP and CSP instances some branching schemes he developed for general branch-and-price algorithms.

Degraeve and Schrage [42] 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 [41] 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. [128] 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 [28] 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 [11], and then embedded into a branch-and-price algorithm in Belov and Scheithauer [12]. 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. [13] 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. [44], 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 [14], who show how the set covering formulation (24)-(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 [101], who treat a number of implementation issues, including specific considerations on the BPP and the CSP.

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.

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 [57]: two classes of 80 instances each, available at Beasley's [8] 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. [130]: 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 [153]: 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 [132]: two sets ('Schwerin 1' and 'Schwerin 2') of 100 instances each with $n = 100$ and $n = 120$, respectively, and $c = 1000$;
- Schoenfield [129]: 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 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. [23]. 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 6.2), for which $L_{\mathrm{LP}} = 3$ 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.

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 5.1):
  - **MTP**, Fortran code by Martello and Toth [107];
  - **BISON**, Pascal code by Scholl et al. [130];
  - **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 [103]). 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 6.3):
  - **VANCE**, C++ implementation of the algorithm by Vance et al. [149], 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 [12], 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 [123] branching rule (also adopted in VANCE).

- Pseudo-polynomial formulations solved via ILP (see Section 4):
  - **ONECUT**, C++ implementation of the one-cut model by Rao [114], Dyckhoff [49], and Stadtler [139], 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 [145], 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 [22], solving the resulting ILP through CPLEX 12.6.0, available at the BPPLIB;
  - **VPSOLVER**, C++ implementation by Brandão and Pedroso [19], 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 (1)-(5), implemented using CPLEX 12.6.0;
  - **CSTRPROG**, C++ implementation of a simple constraint programming algorithm (Section 5.2), using the CP optimizer of CPLEX 12.6.0 and selecting constraint `IloPack` (see Shaw [134]), 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 3.1 and lower bound L_2 of Section 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.

7.3 Experiments

All the experiments but those in Table 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 1-3 give the results for the literature instances. Table 1 provides the results obtained by running the codes with a time limit of one minute. Columns 1 and 2 identify

Table 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: 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 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

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 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 has the same structure as Table 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 (1)-(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 and 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 1 and 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 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 4-6 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 4 the results are listed according to the number of items, in Table 5 according to the capacity, and in Table 6 according to the weight over capacity ratios. The entries give the same information as in Table 1. Similarly, Tables 7-9 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.)

All algorithms were also evaluated, on a subset of the instances of Tables 4-9, 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). 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 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 11 (the counterpart of Table 3) we consider again the instances studied in Tables 4-9, 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

Table 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 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 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 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 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 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)

Table 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 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**

Section 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 12 (number of solved instances and average absolute gap with respect to the lower bound) and 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 understandable. Among branch-and-price algorithms, VANCE has mainly historical interests, but it has an acceptable performance.

Table 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 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)*

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 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 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 [95] 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.

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.

Acknowledgements

We thank Armin Scholl for providing BISON, and Gleb Belov for assisting us in the experiments with his computer code. Thanks are also due to the IBM CPLEX Optimization Team, and in particular to Andrea Tramontani, for the experiments reported in Table 14. We thank the four anonymous referees for helpful comments.

References

[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-Verlag, Berlin, 2013.

[2] C. Alves, F. Clautiaux, J.M. Valério de Carvalho, and J. Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization*. Springer, Cham, 2016.

[3] A.C.F. Alvim, C.C. Ribeiro, F. Glover, and D.J. Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *J. of Heuristics*, 10:205–229, 2004.

[4] H.B. Amor, J. Desrosiers, and J.M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54:454–463, 2006.

[5] 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.

[6] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

[7] J. Balogh, J. Békési, G. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. In *SODA '15: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1425–1438, 2015.

[8] J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[9] J. Beck and V. Sós. *Discrepancy theory*, volume Handbook of Combinatorics, pages 1405–1446. Elsevier, Amsterdam, 1995.

[10] G. Belov. *Problems, models and algorithms in one-and two-dimensional cutting*. PhD thesis, Otto-von-Guericke Universität Magdeburg, 2003.

[11] 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.

[12] 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.

[13] 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. Springer, 2008.

[14] 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, 2005.

[15] C. Berge and E.L. Johnson. Coloring the edges of a hypergraph and linear programming techniques. *Annals of Discrete Mathematics*, 1:65–78, 1977.

[16] A.K. Bhatia and S.K. Basu. Packing bins using multi-chromosomal genetic representation and better fit heuristic. In *Neural Information Processing*, volume 3316 of *Lecture Notes in Computer Science*, pages 181–186. Springer-Verlag, 2004.

[17] A.K. Bhatia, M. Hazra, and S.K. Basu. Better-fit heuristic for one-dimensional bin-packing problem. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 193–196, 2009.

[18] J.-C. Bourjolly and V. Rebetez. An analysis of lower bound procedures for the bin packing problem. *Computers & Operations Research*, 32:395–405, 2005.

[19] 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.

[20] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113:299–344, 2008.

[21] 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-Verlag, 2006.

[22] 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-Verlag, 2010.

[23] 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, 2014.

[24] A. Caprara and M. Monaci. Bidimensional packing by bilinear programming. *Mathematical Programming Series A and B*, 118:75–108, 2009.

[25] 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.

[26] 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.

[27] B. Chen and B. Srivastava. An improved lower bound for the bin-packing problem. *Discrete Applied Mathematics*, 66:81–94, 1996.

[28] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[29] 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.

[30] 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.

[31] E.G. Coffman Jr. and J. Csirik. A classification scheme for bin packing theory. *Acta Cybernetica*, 18:47–60, 2007.

[32] 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.

[33] 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.

[34] 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.

[35] 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*, pages 151–208. Kluwer Academic Publishers, 1999.

[36] 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.

[37] 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*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.

[38] 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.

[39] T.G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. New bin packing fast lower bounds. *Computers & Operations Research*, 34:3439–3457, 2007.

[40] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8:101–111, 1960.

[41] 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.

[42] Z. Degraeve and L. Schrage. Optimal integer solutions to industrial cutting stock problems. *INFORMS Journal on Computing*, 11:406–419, 1999.

[43] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[44] G. Desaulniers, J. Desrosiers, and S. Spoorendonk. Cutting planes for branch-and-price algorithms. *Networks*, 58:301–310, 2011.

[45] 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.

[46] G. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 20, pages 538–549, 2013.

[47] G. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *Automata, Languages, and Programming*, pages 429–441. Springer, 2014.

[48] J. Dupuis, P. Schaus, and Y. Deville. Consistency check for the bin packing constraint revisited. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 117–122. Springer, 2010.

[49] H. Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104, 1981.

[50] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[51] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution*. Physica-Verlag, Heidelberg, 1992.

[52] S. Eilon and N. Christofides. The loading problem. *Management Science*, 17:259–268, 1971.

[53] K. Eisemann. The trim problem. *Management Science*, 3:279–284, 1957.

[54] F. Eisenbrand, D. Pálvölgyi, and T. Rothvoß. Bin packing via discrepancy of permutations. *ACM Transactions on Algorithms (TALG)*, 9:1–15, 2013.

[55] S. Elhedhli. Ranking lower bounds for the bin packing problem. *European Journal of Operational Research*, 160:34–46, 2005.

[56] S. Elhedhli and F. Gzara. Characterizing the optimality gap and the optimal packings for the bin packing problem. *Optimization Letters*, 9:209–223, 2015.

[57] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *J. of Heuristics*, 2:5–30, 1996.

[58] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE 1992 Int. Conference on Robotics and Automation*, pages 1186–1192, Nice, France, 1992.

[59] A.A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38:922–923, 1990.

[60] S.P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31, 2001.

[61] 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.

[62] K. Fleszar and K.S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29:821–839, 2002.

[63] L.R. Ford Jr. and D.R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5:97–101, 1958.

[64] 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.

[65] M.G. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. Freeman, New York, 1979.

[66] 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-Verlag, New York, 1981.

[67] 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.

[68] I.P. Gent. Heuristic solution of open bin packing problems. *J. of Heuristics*, 3:299–304, 1998.

[69] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[70] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 11:863–888, 1963.

[71] P.C. Gilmore and R.E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[72] P. Gómez-Meneses and M. Randall. A hybrid extremal optimisation approach for the bin packing problem. In K.B. Korb, M. Randall, and T. Hendtlass, editors, *Artificial Life: Borrowing from Biology*, volume 5865 of *Lecture Notes in Computer Science*, pages 242–251. Springer-Verlag, 2009.

[73] C. Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operational Research*, 44:197–208, 1990.

[74] 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.

[75] R.W. Haessler. Controlling cutting pattern changes in one dimensional trim problems. *Operations Research*, 23:483–493, 1975.

[76] R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[77] M. Haouari and A. Gharbi. Fast lifting procedures for the bin packing problem. *Discrete Optimization*, 2:201–218, 2005.

[78] 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.

[79] 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.

[80] D.S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

[81] T. Kämpke. Simulated annealing: use of a new tool in bin packing. *Annals of Operations Research*, 16:327–332, 1988.

[82] 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.

[83] K. Kaparis and A.N. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124:69–91, 2010.

[84] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. 23rd Annual IEEE Symp. Found. Comput. Sci.*, pages 312–320, 1982.

[85] 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.

[86] 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.

[87] 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.

[88] K.C. Kiwiel. An inexact bundle approach to cutting-stock problems. *INFORMS Journal on Computing*, 22:131–143, 2010.

[89] R.E. Korf. A new algorithm for optimal bin packing. In *Proceedings of the Eighteenth National Conference on Articial Intelligence*, pages 731–736, 2002.

[90] R.E. Korf. An improved algorithm for optimal bin packing. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1252–1258, 2003.

[91] M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39:616–622, 1991.

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[97] 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.

[98] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.

[99] 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.

[100] 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 companion on Genetic and evolutionary computation*, pages 257–258, 2011.

[101] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2005.

[102] G.S. Lueker. Bin packing with items uniformly distributed over intervals $[a, b]$. In *24th Annual Symposium on Foundations of Computer Science*, pages 289–297, 1983.

[103] 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.

[104] O. Marcotte. The cutting stock problem and integer rounding. *Mathematical Programming*, 33:82–92, 1985.

[105] O. Marcotte. An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4:239–243, 1986.

[106] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[107] 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`).

[108] 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.

[109] A. Newman, O. Neiman, and A. Nikolov. Beck's three permutations conjecture: A counterexample and some consequences. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 253–262, 2012.

[110] C. Nitsche, G. Scheithauer, and J. Terno. Tighter relaxations for the cutting stock problem. *European Journal of Operational Research*, 112:654–663, 1999.

[111] 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.

[112] R. Poli, J. Woodward, and E.K. Burke. A histogram-matching approach to the evolution of bin-packing strategies. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3500–3507, 2007.

[113] 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.

[114] M.R. Rao. On the cutting stock problem. *Journal of the Computer Society of India*, 7:35–39, 1976.

[115] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[116] J. Rietz and S. Dempe. Large gaps in one-dimensional cutting stock problems. *Discrete Applied Mathematics*, 156:1929–1935, 2008.

[117] 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 '07, pages 1365–1371, New York, NY, USA, 2007. ACM.

[118] P. Rohlfshagen and J.A. Bullinaria. Nature inspired genetic algorithms for hard packing problems. *Annals of Operations Research*, 179:393–419, 2010.

[119] G.M. Roodman. Near optimal solutions to one-dimensional cutting stock problem. *Computers & Operations Research*, 13:713–719, 1986.

[120] 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*, Lecture Notes in Computer Science, pages 1295–1306, 2003.

[121] 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, 2002.

[122] T. Rothvoß. Approximating Bin Packing within O(log OPT * log log OPT) bins. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 20–29, 2013.

[123] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981.

[124] 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*, Lecture Notes in Computer Science, pages 815–822. Springer, 2012.

[125] G. Scheithauer and J. Terno. A branch-and-bound algorithm for solving one-dimensional cutting stock problems exactly. *Applicationes Mathematicae*, 23:151–167, 1995.

[126] 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.

[127] 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.

[128] 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.

[129] 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.

[130] 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.

[131] E.L. Schreiber and R.E. Korf. Improved bin completion for optimal bin packing and commentnumber partitioning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 651–658, 2013.

[132] 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.

[133] 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.

[134] 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, 2004.

[135] 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 *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, GECCO '13, pages 1549–1556. ACM, New York, NY, USA, 2013.

[136] K. Sim, E. Hart, and B. Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In C.A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg, 2012.

[137] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41:579, 1994.

[138] A. Singh and A.K. Gupta. Two heuristics for the one-dimensional bin-packing problem. *OR Spectrum*, 29:765–781, 2007.

[139] 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.

[140] A. Stawowy. Evolutionary based heuristic for bin packing problem. *Computers & Industrial Engineering*, 55:465–474, 2008.

[141] 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.

[142] M.A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118:73–84, 2003.

[143] 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*, pages 1140–1149, 2008.

[144] R. Vahrenkamp. Random search in the one-dimensional cutting stock problem. *European Journal of Operational Research*, 95:191–200, 1996.

[145] 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.

[146] J.M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

[147] J.M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17:175–182, 2005.

[148] P.H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9:211–228, 1998.

[149] 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.

[150] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86:565–594, 1999.

[151] 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.

[152] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130:249–294, 2011.

[153] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: a computational study. *Operations-Research-Spektrum*, 18:131–144, 1996.

[154] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.

[155] L.A. Wolsey. Valid inequalities, covering problems and discrete dynamic programs. In *Studies in Integer Programming*, volume 1, pages 527–538. Elsevier, 1977.