Solving the parallel processor scheduling and bin packing problems with contiguity constraints: mathematical models and computational studies

Fatih Burak Akçay, Maxence Delorme

Department of Econometrics and Operations Research, Tilburg University, The Netherlands
Corresponding author m.delorme@tilburguniversity.edu

Abstract

The parallel processor scheduling and bin packing problems with contiguity constraints are important in the field of combinatorial optimization because both problems can be used as components of effective exact decomposition approaches for several two-dimensional packing problems. In this study, we provide an extensive review of existing mathematical formulations for the two problems, together with some model enhancements and lower bounding techniques, and we empirically evaluate the computational behavior of each of these elements using a state-of-the-art solver on a large set of literature instances. We also assess whether recent developments such as meet-in-the middle patterns and the reflect formulation can be used to solve the two problems more effectively. Our experiments demonstrate that some features, such as the mathematical model used, have a major impact on whether an approach is able to solve an instance, whereas other features, such as the use of symmetry-breaking constraints, do not bring any significant empirical advantage despite being useful in theory. Overall, our goal is to help the research community design more effective yet simpler algorithms to solve the parallel processor scheduling and bin packing problems with contiguity constraints and closely related extensions so that, eventually, those can be integrated into more exact methods for two-dimensional packing problems.

Keywords: Packing, Contiguity constraints, Exact algorithms, Computational evaluation.

1 Introduction

Cutting and Packing (C&P) problems have been widely studied in the literature for the last eighty years, starting from the seminal work of Kantorovich in the late thirties [55]. There are numerous practical applications for C&P problems, whether those are one-dimensional (automobile component manufacturing [4] and high performance computing [74]), two-dimensional (glass manufacturing industry [69] and newspaper layouting [76]), three-dimensional (container loading [68] and 3D printing [24]), or even higher [51].

Two of the most studied two-dimensional C&P problems are the *Strip Packing Problem* (SPP) and the *Orthogonal Packing Problem* (OPP). In the SPP, we are given a rectangular strip with fixed width and infinite height together with a set of rectangular items. The goal is to pack all the items into the strip while minimizing the strip height. An illustrative example outlining an SPP instance and its corresponding optimal solution are shown in Figures 1a and 1b. In the OPP, we are given a rectangular bin with fixed width and fixed height and a set of rectangular items. The goal is to determine whether all the items can be packed into the bin. As observed by Iori et al. [52], although the OPP is said to be in *decision version* (i.e., the expected solution is either YES or NO, also referred to as *recognition version* in the complexity literature), one is generally requested to also produce the specific packing (if it exists) in practical applications. In most SPP and OPP related literature, it is assumed that the items (i) do not

overlap, (ii) must entirely lie within the container, and (iii) are packed with their edges parallel to the borders of the container. While some of these assumptions were relaxed in a few C&P studies [6, 46, 84], we point out that, to the best of our knowledge, this has never been the case for the SPP or the OPP.

According to the typology proposed by Wäsher et al.[78], the SPP falls into the category of *(two-dimensional) open dimension problems*. Following the three-field typology proposed by Lodi et al.[60], the SPP is denoted as 2SP|O|F, where "O" stands for "oriented items" (i.e., item rotation by 90° is not allowed) and where "F" stands for "free" (i.e., guillotine cutting, also known as edge-to-edge cutting, is not required). Note that the SPP was not formally defined according to the (older) typology proposed by Dyckhoff [39] and that none of the typologies offers a classification for the OPP.

The SPP is known to be strongly $\mathcal{NP}$-hard by polynomial reduction from the one-dimensional bin packing problem [61]. Therefore, its recognition version, the OPP, is strongly $\mathcal{NP}$-complete. Nevertheless, both problems have attracted the attention of the C&P research community who has developed over the years a large number of heuristics [19, 67], metaheuristics [53, 60], and exact approaches [52]. Those exact approaches include, among others, integer linear programming (ILP) models solved through an ILP solver [21], branch-and-bound (B&B) algorithms [61], branch-and-price (B&P) algorithms [14], constraint programming (CP) [45], and decomposition approaches [26]. Based on recent empirical studies focusing on exact approaches for the SPP [27, 37], decomposition methods appear to be the most competitive.

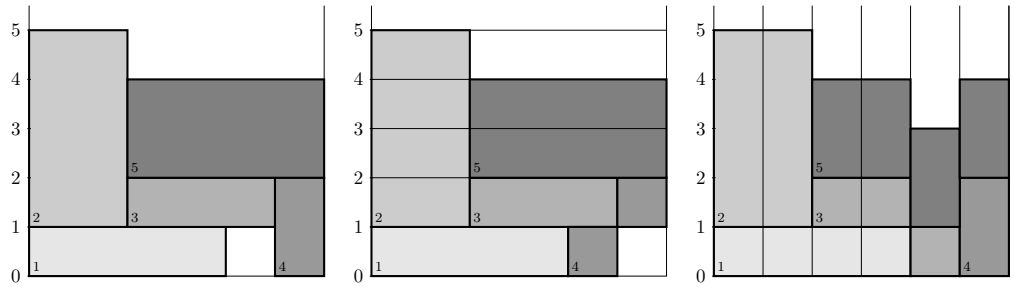
In all existing decomposition algorithms proposed in the literature to solve either the SPP or the OPP [25, 26, 27, 37], the *original problem* is divided into two (smaller) sub-problems: a *main problem* (MP) and a *secondary problem* (SP). In the MP, one solves a relaxed version of the original problem that fixes one coordinate (the abscissa or the ordinate) of every item. In the SP, one determines whether or not it is possible to complete the MP solution (i.e., find the second coordinate for every item) such that the solution becomes feasible for the original problem. If the answer to the SP is Yes, then the MP solution is also feasible for the original problem whereas if the answer is No, then one needs to add a constraint to prevent the MP solution from being generated again.

As far as the MP is concerned, two main relaxations can be considered for the SPP: the one-dimensional bin packing problem with contiguity constraints (1CBP) and the parallel processor scheduling problem with contiguity constraints (P|*cont*|C_{max}) [27]. To the best of our knowledge, only the latter was used in the literature as the MP of a decomposition approach whereas the former was mostly used to derive valid bounds [2, 61]. In the 1CBP, the items of the original problem are cut into unit-height slices and the objective is to pack all the resulting slices into the minimum number of identical one-dimensional bins while enforcing that the slices belonging to the same item are packed into contiguous bins. In this problem, the bin capacity of each one-dimensional bin is equal to the width of the container. An illustrative example outlining the optimal solution for the 1CBP relaxation of the SPP instance described in Figure 1a is provided in Figure 1c. In the P|*cont*|C_{max}, the items of the original problem are cut into unit-width slices and the objective is to pack all the resulting slices into a fixed number of one-dimensional bins whose capacity is to be minimized while enforcing that the slices belonging to the same item are packed into contiguous bins. In this problem, the number of one-dimensional bins is equal to the width of the container. An illustrative example outlining the optimal solution for the P|*cont*|C_{max} relaxation of the SPP instance described in Figure 1a is provided in Figure 1d. One can observe that the recognition version of the 1CBP is the same as the recognition version of the P|*cont*|C_{max} since, in both cases, the number of bins is equal to one of the two container dimensions whereas the bin capacity is equal to the other container dimension. Therefore, any solution method tailored to solve one problem can also be used to solve the other. We point out, however, that applying the 1CBP relaxation to an OPP instance does not produce the same numerical instance as the one obtained after applying the P|*cont*|C_{max} relaxation.

Figure 1: An SPP instance, its optimal solution, and the solution of its two relaxations [27]

	w	h
strip	6	∞
item 1	4	1
item 2	2	4
item 3	3	1
item 4	1	2
item 5	4	2

(a) SPP instance

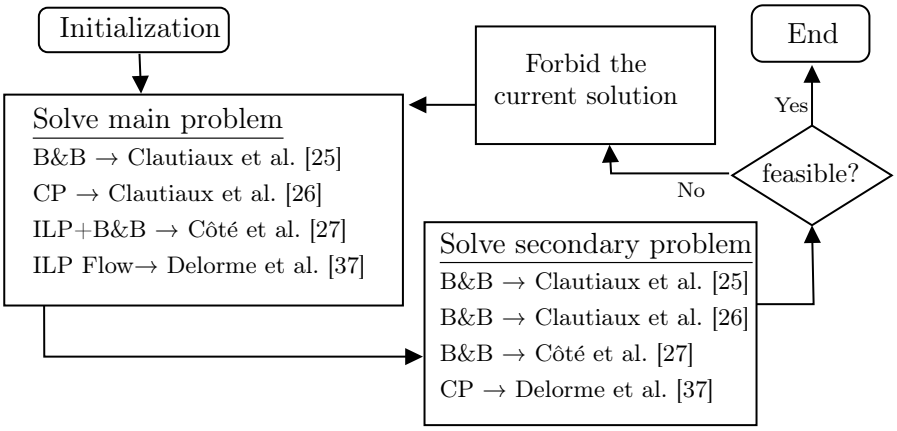


(b) SPP solution (c) 1CBP solution (d) P|*cont*|C_{max} solution

As far as the SP is concerned, even though the terminology varied in the literature, it can be defined as a scheduling problem with non-overlapping constraints among subsets of tasks.

A decomposition approach for the SPP and for the OPP can roughly be summarized as (i) an algorithm to solve the MP, (ii) an algorithm to solve the SP, and (iii) a set of enhancing techniques (e.g., preprocessing, symmetry-breaking constraints, and cut lifting/strengthening). We report in Figure 2 a schematic representation of the decomposition approaches proposed so far for the SPP and the OPP. As one can observe, several combinations of MP/SP solution methods were tested in the literature and most papers included an extensive set of computational experiments comparing the performance of the newly introduced decomposition approach with the performance of its predecessors. However, these experiments measured the performance of each algorithm as a whole, never component by component. In fact, it could be possible that a (new) decomposition algorithm composed of the MP of Côté et al. [27] and the SP of Delorme et al. [37] obtains state-of-the-art results for the SPP. This is why identifying the most effective solution technique to solve each component of the decomposition is crucial in order to develop better algorithms for the SPP and the OPP. This is especially important considering that such algorithms can be extended to solve related two-dimensional packing problems such as the two-dimensional knapsack problem [48] and the two-dimensional bin packing problem [29].

Figure 2: Existing decomposition algorithms for the SPP and the OPP



The main objective of this paper is to review the solution methods that were proposed for the P|*cont*|C_{max} and the 1CBP and to test these solution methods on literature benchmarks in order to determine the ones that are the most effective. In addition, we also assess whether or not recent advances in the C&P area such as reduced-cost variable fixing [32], meet-in-the-middle patterns [30], and the reflect formulation [35] could be used to solve the two problems more efficiently. We also review and evaluate a number of lower bounding techniques and introduce new ones. In order to make sure that the evaluated

methods are easy to re-implement and modify by the C&P community, which is essential if we want our findings to be used in the future, our work focuses on techniques that both work well in practice and are easy to code and modify. In addition, an implementation of all the reviewed techniques is made available online.

The paper is organized as follows. In Section 2, we review the most important approaches proposed in the literature for the SPP, the OPP, and closely related problems. In Section 3, we describe the mathematical models for the 1CBP and the P|*cont*|C_{max} that have been proposed in the literature, and in Section 4, we detail existing model enhancement techniques. In Section 5, we discuss the existing lower bounds for the 1CBP and the P|*cont*|C_{max} and propose new ones. In Section 6, we evaluate each of the reviewed techniques through an extensive set of computational experiments and derive some conclusions and future research directions in Section 7.

2 Literature review

Exact approaches for two-dimensional packing problems have gathered a lot of attention in the C&P literature, especially in recent years. We refer the reader to Iori et al. [52] for a comprehensive survey on the topic.

One of the first exact algorithms designed to solve the SPP was proposed by Martello et al. [61]. The authors introduced a B&B algorithm that uses the concept of a staircase placement together with some advanced reduction procedures and bounding techniques. Later on, Alvarez-Valdés et al. [2] and Boschetti and Montaletti [16] improved the performance of that B&B algorithm by integrating advanced preprocessing techniques, enhanced dominance criteria, and additional bounding procedures. Castro and Oliveira [22] and Castro and Grossmann [21] developed ILP models for the SPP inspired by time representation in scheduling problems. Soh et al. [75] translated the OPP into a SAT problem and solved it trough a SAT encoding. By iteratively solving a number of OPP in which the bin height is increased until a feasible solution is found (also known as destructive bound), they showed that their method could be used to solve the SPP. Another SAT encoding was proposed later on by Grandcolas and Pinto [45]. Recently, Silva et al. [72] proposed the so-called "floating-cuts" formulation, a model for general rectangular cutting problems with fixed container dimensions (i.e., that could be use for the OPP but not for the SPP). Whereas the aforementioned literature solves the SPP (or the OPP) with a direct approach, the most competitive (and recent) approaches use decomposition strategies instead (see Figure 2).

The SPP and the OPP are closely related to the *two-dimensional knapsack problem* (2D-KP) and the *two-dimensional bin packing problem* (2D-BPP). In the 2D-KP, the objective is to pack a subset of rectangular items with maximum value into a rectangular bin. Caprara and Monaci [20] introduced a decomposition approach to solve the problem where the MP selects the items contained in the solution while the SP, an OPP, determines whether it is possible to pack the selected items into the bin. In the 2D-BPP, the goal is to pack a set of rectangular items into the minimum number of identical rectangular bins. The most effective algorithm to solve the problem was proposed by Côté et al. [29] and also uses a decomposition strategy. The MP assigns a set of items to each bin whereas the SP, a sequence of OPP, determines whether it is possible to pack each item set into a bin.

Our two problems of interest, the 1CBP and the P|*cont*|C_{max}, belong to the category of one-dimensional packing problems. To the best of our knowledge, the P|*cont*|C_{max} was never studied as a standalone problem and the only study focusing on the 1CBP was proposed by Mesyagutov et al. [66] who introduced an ILP model based on the well-known set covering formulation [44] and solved it within a B&P framework. Among the papers who tackled the 1CBP as a component of a more general algorithm, we mention the work of Martello et al. [61], who solved the problem with a B&B algorithm, the work of Alvarez-Valdés et al. [2], who introduced an ILP formulation for the problem and solved it with an ILP solver, and the

work of Friedow and Scheithauer [41], who solved the model proposed by Mesyagutov et al. [66] with a cutting plane algorithm. Among the papers who tackled the P|*cont*|C_{max} as a component of a more general algorithm, we mention the work of Boschetti and Montaletti [16] who introduced an ILP formulation for the problem and solved it with an ILP solver, the work of Côté et al. [27] who solved the problem with a hybridized procedure that first calls a tailored B&B algorithm for a fixed duration and then solves the model of Boschetti and Montaletti [16] through an ILP solver, and the work of Delorme et al. [37] who introduced an ILP model based on the well-known arcflow formulation of Valério de Carvalho [77] and solved it with an ILP solver. Other works tackled the recognition version of the 1CBP and the P|*cont*|C_{max}. Clautiaux et al. [25] solved the problem through B&B while Belov et al. [10] introduced another ILP formulation to model the problem where the novelty with respect to the model introduced by Alvarez-Valdés et al. [2] lies in the way to represent the contiguity constraints. We categorize the aforementioned 1CBP and P|*cont*|C_{max} literature in Table 1 and indicate, for each paper, the problem tackled (1CBP, P|*cont*|C_{max}, or the recognition version of these problems), its purpose (to obtain a valid bound or to serve as a component of a decomposition algorithm), and the method used to solve it (ILP, CP, or a tailored enumerative algorithm). Each of the methods marked with the symbol "⊗" is reviewed in the next section and empirically evaluated in Section 5.

Table 1: Categorization of the 1CBP and P|*cont*|C_{max} literature

Reference	Problem			Purpose		Solution approach		
	1CBP	P|*cont*|C_{max}	Recognition version	Bounds	Decomposition component	ILP	CP	Enumerative
Alvarez-Valdés et al. [2]	×			×		⊗		
Belov et al. [10]			×	×		⊗		
Boschetti and Montaletti[16]		×		×		⊗		
Clautiaux et al. [25]			×		×			×
Clautiaux et al. [26]			×		×		⊗	
Côté et al. [27]		×	×		×	⊗		×
Delorme et al. [37]		×	×		×	⊗		
Friedow and Scheithauer[41]	×			×				×
Martello et al. [61]	×			×				×
Mesyagutov et al. [66]	×			×		×		×

The 1CBP is a special case of the bin packing problem with time lags recently introduced by Letelier et al. [58] in which a time lag of exactly one bin is imposed between sequences of item slices. The 1CBP is also a special case of the resource-constrained project scheduling problem [56] where only one resource is considered and where precedence constraints are not allowed. That observation was used by Clautiaux et al. [26] to derive an effective CP approach to solve the recognition version of the problem. We also point out that, if every item in a 1CBP instance has at most two slices (i.e., if every item in the original SPP instance has height 1 or 2), then the problem can be formulated as a two bar charts packing problem [7, 40].

We finish this literature review by pointing out that several 1CBP and P|*cont*|C_{max} relaxations were also considered in the literature, leading to even weaker SPP and OPP relaxations. On the 1CBP side, Belov et al. [10] and Alvarez-Valdés et al. [2] considered a relaxation in which the contiguity constraints were replaced by conflict constraints where slices belonging to the same item were forbidden to be packed in the same bin, resulting in a bin packing problem with conflicts. Alvarez-Valdés et al. [2] studied an even weaker relaxation in which the contiguity constraints were completely removed, resulting in the very well-known one-dimensional bin packing problem [36]. Note that, even though they were never studied in the SPP and OPP context, similar relaxations can also be derived on the P|*cont*|C_{max} side, resulting in the P||C_{max} with conflicts [57] and the P||C_{max} [33]. We point out that, to the best of our knowledge, these weaker relaxations were never used as the MP of an effective decomposition approach for the SPP or the OPP.

3 Mathematical models

In this section, we review the mathematical models (ILP and CP) that were proposed in the literature for the 1CBP and the P|*cont*|C_{max}. We start by introducing the mathematical notation used in the models, we then present three ILP models for the P|*cont*|C_{max} followed by one ILP model for the 1CBP and demonstrate that a formulation solving one problem can easily be modified to solve the other, and we finish the section by describing a CP formulation that can be used to model both problems. Throughout the section, we use the terminology associated with the optimization version of the 1CBP and the P|*cont*|C_{max} (i.e., in which an objective function is considered), however, all models can easily be adapted to address the recognition version of the problems (i.e., when both the number of bins and the capacity are limited).

3.1 Mathematical notation

Because the P|*cont*|C_{max} and the 1CBP originate from the SPP literature, we first introduce the mathematical notation for the SPP and reuse it for the two studied relaxations. In the SPP, we are given a set of n items to be packed in a strip with fixed width W and minimum height. In the recognition version of the problem, the OPP, the strip height H is fixed. Each item i ($i = 1, \dots, n$) has a width w_i and a height h_i. Without loss of generality, we assume hereafter that w_i and h_i are positive integers ($i = 1, \dots, n$). As it can be more convenient for modelling purposes, an SPP instance can also be defined as a set of m item types where each item type j ($j = 1, \dots, m$) has a width w_j, a height h_j, and a demand d_j (where $\sum_{j=1}^{m} d_j = n$). Using a Cartesian coordinate system, an SPP solution identifies, for every item, the coordinate (p, q) in which the bottom-left corner of the item is packed. We call a *row* the unit-height slice obtained by cutting the strip horizontally, and we call a *column* the unit-width slice obtained by cutting the strip vertically. Columns are indexed from 0 to $W - 1$ whereas rows are indexed from 0 to UB $- 1$ (where UB is an instance-specific valid upper bound). We say that an item *engages* a column p' (or a row q') when a portion of the item intersects with p' (or q'). When packed in coordinate (p, q), an item i engages columns $p, p + 1, \dots, p + w_i - 1$ and rows $q, q + 1, \dots, q + h_i - 1$.

Using a similar notation, in the P|*cont*|C_{max}, we are given a set of n items to be packed into a fixed number W of identical bins whose capacity is to be minimized. Following the scheduling terminology, a "bin" is a "machine" and the "bin capacity" is the "makespan". Each item i ($i = 1, \dots, n$) has length l_i and occupies a given number k_i of consecutive bins (corresponding to columns in the SPP). In the P|*cont*|C_{max} relaxation of an SPP instance, $l_i = h_i$ and $k_i = w_i$ for every item i ($i = 1, \dots, n$). A P|*cont*|C_{max} solution identifies, for every item, the first column p occupied by the item. When packed in column p, an item i engages columns $p, p+1, \dots, p+w_i-1$. In Figure 1d, a feasible P|*cont*|C_{max} solution with makespan 5 is depicted. In the figure, item 5 is packed in column 2 and engages columns $2, 3, 4$ and 5. Note that item types can also be used to describe a P|*cont*|C_{max} instance and that the notion of rows is not relevant for the problem.

Similarly, in the 1CBP, we are given a set of n items to be packed in the minimum number of bins with capacity W. Each item i ($i = 1, \dots, n$) has length l_i and occupies a given number k_i of consecutive bins (corresponding to rows in the SPP). In the 1CBP relaxation of an SPP instance, $l_i = w_i$ and $k_i = h_i$ for every item i ($i = 1, \dots, n$). A 1CBP solution identifies, for every item, the first row q occupied by the item. When packed in row q, an item i engages rows $q, q+1, \dots, q+h_i-1$. In Figure 1c, a feasible 1CBP solution using 5 bins is depicted. In the figure, item 5 is packed in row 2 and engages rows 2 and 3. Note that item types can also be used to describe a 1CBP instance and that the notion of columns is not relevant for the problem.

We now introduce several sets that are used in the mathematical models hereafter:

- $\mathcal{N} = \{1, 2, \dots, n\}$ is the item set ($\mathcal{M}$ is used for the set of item types);
- $\mathcal{W} = \{0, 1, \dots, W - 1\}$ is the set of columns;

- $\mathcal{W}_i = \{0, 1, \ldots, W - w_i\}$ is the set of columns in which item i can be packed in a feasible P|*cont*|C_{max} solution ($\mathcal{W}_j$ are used for the item types);
- $\mathcal{W}_i(p) = \{p' \in \mathcal{W}_i : 0 \leq p - w_i + 1 \leq p' \leq p\}$ gathers every column p' such that item i would engage column p if i was packed in p' ($\mathcal{W}_j(p)$ are used for the item types);
- $\mathcal{H} = \{0, 1, \ldots, UB - 1\}$ is the set of rows;
- $\mathcal{H}_i = \{0, 1, \ldots, UB - h_i\}$ is the set of rows in which item i can be packed in a feasible 1CBP solution ($\mathcal{H}_j$ are used for the item types);
- $\mathcal{H}_i(q) = \{q' \in \mathcal{H}_i : 0 \leq q - h_i + 1 \leq q' \leq q\}$ is the subset of rows for which item i would engage row q if i was packed in a row of the subset ($\mathcal{H}_j(q)$ are used for the item types).

3.2 ILP models for the P|*cont*|C_{max}

3.2.1 BKRS formulation

The first ILP formulation we review was proposed by Belov et al. [10] and will be referred to as **BKRS** hereafter. Even though this formulation was initially proposed to solve the recognition version of the P|*cont*|C_{max}, **BKRS** can trivially be adapted to solve the optimization version of the problem.

Key idea: Model **BKRS** builds on the textbook P||C_{max} ILP formulation [71] in which binary decision variables indicate whether a given item (or in our case, a given item slice) is packed into a given column (or bin). In order to take contiguity into account, **BKRS** uses a set of binary variables that keeps track of the column index in which the first slice of each item is packed. Contiguity is then imposed through an additional set of constraints that only allows an item slice to be packed in a column if (i) this is the first slice of that item to be packed in a column or (ii) the (immediate) previous column also contains a slice of that item.

Variables and mathematical formulation: Model **BKRS** requires: (i) binary decision variables x_{ip} taking value 1 if column p is the column in which the first slice of item i is packed, and value 0 otherwise ($i \in \mathcal{N}, p \in \mathcal{W}_i$), (ii) binary decision variables a_{ip} taking value 1 if a slice of item i is packed in column p, and value 0 otherwise ($i \in \mathcal{N}, p \in \mathcal{W}_i$), and (iii) decision variable z indicating the maximum column height (or the bin capacity). The model can be defined as follows:

$$\min \quad z \tag{1}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{W}_i} x_{ip} = 1 \qquad i \in \mathcal{N}, \tag{2}$$

$$\sum_{i \in \mathcal{N}} h_i a_{ip} \leq z \qquad p \in \mathcal{W}, \tag{3}$$

$$a_{i0} = x_{i0} \qquad i \in \mathcal{N}, \tag{4}$$

$$a_{ip} \leq a_{i,p-1} + x_{ip} \qquad p \in \mathcal{W}_i \setminus \{0\}, i \in \mathcal{N}, \tag{5}$$

$$a_{ip} \leq a_{i,p-1} \qquad p \in \mathcal{W} \setminus \mathcal{W}_i, i \in \mathcal{N}, \tag{6}$$

$$\sum_{p \in \mathcal{W}} a_{ip} = w_i \qquad i \in \mathcal{N}, \tag{7}$$

$$x_{ip} \in \{0, 1\} \qquad i \in \mathcal{N}, p \in \mathcal{W}_i, \tag{8}$$

$$a_{ip} \in \{0, 1\} \qquad i \in \mathcal{N}, p \in \mathcal{W}. \tag{9}$$

The objective function (1) minimizes the maximum column height. Constraints (2) make sure that, for each item i, exactly one bin is identified as the column in which the first slice of item i is packed. Constraints (3) prevent the bin capacity from being violated. Constraints (4)-(6) ensure that a slice of item i is only allowed to be packed in column p (i.e., $a_{ip} = 1$) if this is the first slice of item i to be packed in a column (i.e., if $x_{ip} = 1$) or if column $p-1$ also contains a slice of item i (i.e., if $a_{i,p-1} = 1$). Constraints (7) make sure that, for each item, the required number of slices is packed. Note that model **BKRS** requires $O(nW)$ variables and $O(nW)$ constraints and cannot easily be extended to take item types into account instead of item indices.

3.2.2 BM formulation

The second formulation we review was proposed by Boschetti and Montaletti [16] and will be referred to as **BM** hereafter. According to the authors, **BM** was inspired by mathematical models that were previously introduced for related two-dimensional C&P problems [5, 15].

Key idea: Unlike **BKRS** in which a decision variable was created for each column and for each item slice, model **BM** only considers one decision variable for each column and for each item. Intuitively, **BM** can be seen as a version of **BKRS** where one only decides the column in which the first slice of each item is packed. Contiguity is implicitly taken into account in the model by assuming that, if the first slice of item i is packed in column p, then columns $p+1, p+2, \ldots, p+w_i-1$ also contain a slice of item i.

Variables and mathematical formulation: Model **BM** requires the same variables x_{ip} and z as previously introduced for **BKRS** and can be defined as follows:

$$\min \quad z \tag{10}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{W}_i} x_{ip} = 1 \qquad i \in \mathcal{N}, \tag{11}$$

$$\sum_{i \in \mathcal{N}} \sum_{p' \in \mathcal{W}_i(p)} h_i x_{ip'} \leq z \qquad p \in \mathcal{W}, \tag{12}$$

$$x_{ip} \in \{0,1\} \qquad i \in \mathcal{N}, p \in \mathcal{W}_i, \tag{13}$$

where the objective function (10) and constraints (11) are the same as in **BKRS**, and where capacity constraints (12) are updated to sum over $x_{ip'}$ variables (indicating whether the first slice of item i is packed in a column p' such that, due to the contiguity constraints, a slice of i must also be packed in column p) instead of a_{ip} variables (indicating whether a slice of item i is packed in column p). Note that model **BM** requires $O(nW)$ variables and $O(n+W)$ constraints and, unlike **BKRS**, can easily be extended to take item types into account instead of item indices.

3.2.3 FLOW-PCC formulation

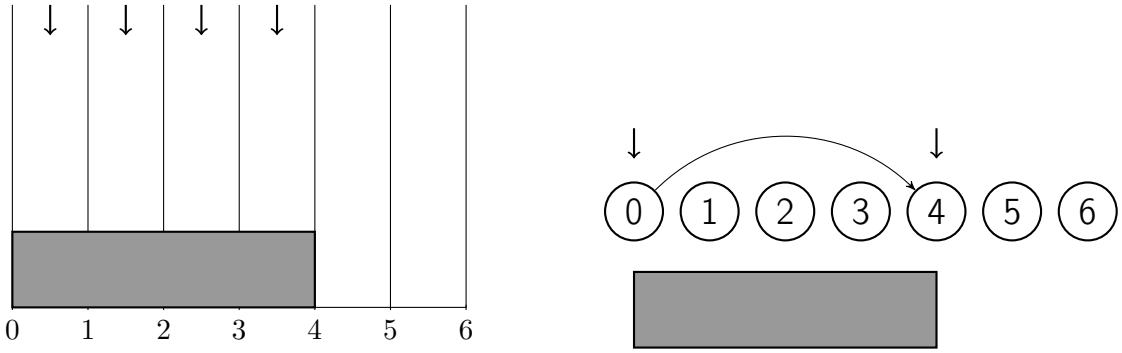
The third formulation we review was proposed by Delorme et al. [37] and will be referred to as **FLOW-PCC** hereafter. **FLOW-PCC** is based on the well-known arcflow formulation introduced by Wolsey [82] and popularized by Valério de Carvalho [77] for solving the one-dimensional cutting stock problem. Because state-of-the-art mathematical solvers can often solve large-sized arcflow models to optimality within a reasonable computational time, these formulations have become very popular in the C&P research community. We refer the reader to the work of de Lima et al. [31] for a recent survey on arcflow formulations.

Key idea: As **BM**, model **FLOW-PCC** requires one decision variable for each column and for each item. The main difference between the two models lie in the way capacity constraints are enforced.

- In **BM**, one keeps track of the length of the item slices contained in every column and makes sure that the total length in a column never exceeds the capacity z;
- In **FLOW-PCC**, the capacity is seen as a resource. One keeps track of the resource available in every column and makes sure that the first slice of an item can only be packed in a column with enough resources left. If the first slice of item i is packed in column p, then h_i resources are consumed in column p and released in column $p+w_i$. In the first column (with index 0), z resources are available.

A graphical representation summarizing the difference between the two models is available in Figure 3. In **BM** (on the left part of the figure), packing the first slice of the grey item in column 0 means that a slice of the item also needs to be counted in columns 1, 2, and 3 (i.e., the decision variable appears in four capacity constraints). In **FLOW-PCC** (on the right part of the figure), packing the first slice of the grey item in column 0 means that the resource is locked in column 0 and released in column 4 (i.e., the decision variable only appears in two flow conservation constraints).

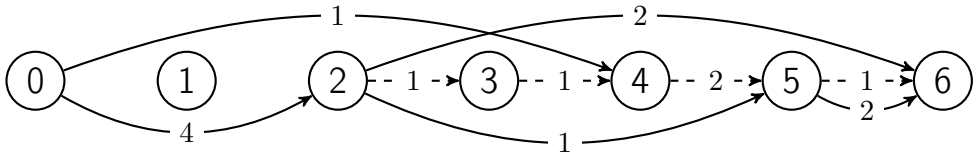
Figure 3: Difference between **BM** and **FLOW-PCC** models



(a) **BM** model (b) **FLOW-PCC** model

Variables and mathematical formulation: Model **FLOW-PCC** uses graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where vertex set $\mathcal{V} = \{0, 1, \ldots, W\}$ and where arc set $\mathcal{A}$ is composed of (i) item arcs $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where $\mathcal{A}_i$ contains every arc $(d, d + w_i)$ such that $d \in \mathcal{W}_i$ and (ii) loss arcs $\mathcal{A}_0$ containing every arc $(d, d + 1)$ such that $d = 0, \ldots, W - 1$. If selected in a solution, item arc $(d, d + w_i) \in \mathcal{A}_i$ carries h_i units of flow from node d to node $d + w_i$. If selected in a solution, loss arc $(d, d+1) \in \mathcal{A}_0$ carries 1 unit of flow from node d to node $d + 1$ and represents one unit of unused space in column d (note that a loss arc may be selected multiple times). For notation purposes, we assume $h_0 = 1$. A feasible **FLOW-PCC** solution can then be defined as a $0 - W$ flow where z flows are sent from node 0 to node W. A **FLOW-PCC** solution corresponding to the example depicted in Figure 1d is shown in Figure 4. In the figure, the number located in the middle of an arc indicates the amount of flow it carries. Loss arcs are represented with dotted lines.

Figure 4: A **FLOW-PCC** solution corresponding to the example depicted in Figure 1d



As far as variables are concerned, model **FLOW-PCC** requires: (i) binary decision variables x_{ide} taking value 1 if arc $(d, e) \in \mathcal{A}_i$ is selected in the solution (or in other words, if column d is the column in which the first slice of item i is packed), and value 0 otherwise ($i \in \mathcal{N}$), (ii) integer decision variables x_{0de} indicating the number of times arc $(d, e) \in \mathcal{A}_0$ is selected in the solution (or in other words, the amount

of unused space in column d), and (iii) integer decision variable z indicating the amount of flow sent from node 0 to node W. The model can be defined as follows:

$$\min \quad z \tag{14}$$

$$\text{s.t.} \quad \sum_{(d,e)\in\mathcal{A}_i} x_{ide} = 1 \qquad i \in \mathcal{N}, \tag{15}$$

$$\sum_{i\in\{0\}\cup\mathcal{N}} \sum_{\substack{(d,e)\in\mathcal{A}_i \\ d=0}} h_i x_{ide} = z, \tag{16}$$

$$\sum_{i\in\{0\}\cup\mathcal{N}} \sum_{\substack{(d,e)\in\mathcal{A}_i \\ d=p}} h_i x_{ide} = \sum_{i\in\{0\}\cup\mathcal{N}} \sum_{\substack{(d,e)\in\mathcal{A}_i \\ e=p}} h_i x_{ide} \qquad p \in \mathcal{V}\setminus\{0,W\}, \tag{17}$$

$$\sum_{i\in\{0\}\cup\mathcal{N}} \sum_{\substack{(d,e)\in\mathcal{A}_i \\ e=W}} h_i x_{ide} = z, \tag{18}$$

$$x_{0de} \in \mathbb{N}_0 \qquad (d,e) \in \mathcal{A}_0, \tag{19}$$

$$x_{ide} \in \{0,1\} \qquad i \in \mathcal{N}, (d,e) \in \mathcal{A}_i. \tag{20}$$

where the objective function (14) and constraints (15) are the same as in **BM** and where flow conservation constraints (16)-(18) replace capacity constraints (12). Note that, like **BM**, model **FLOW-PCC** requires $O(nW)$ variables and $O(n+W)$ constraints and can easily be extended to take item types into account instead of item indices. We point out that the constraint matrix in **FLOW-PCC** is sparser than the constraint matrix of **BM** as every x_{ide} **FLOW-PCC** variable ($i \in \mathcal{N}, (d,e) \in \mathcal{A}_i$) appears in exactly 2 flow conservation constraints (16)-(18) whereas every x_{ip} **BM** variable ($i \in \mathcal{N}, p \in \mathcal{W}_i$) appears in w_i capacity constraints (12). We also point out that, because of the loss arcs, **FLOW-PCC** requires up to W additional variables compared to **BM**.

3.3 ILP models for the 1CBP

3.3.1 APT formulation

To the best of our knowledge, the only ILP model for the 1CBP available in the literature was proposed by Alvarez-Valdés et al. [2] and will be referred to as **APT** hereafter. We first provide an updated version of the model introduced by Alvarez-Valdés et al., we show afterwards how a minor modelling change can strengthen the continuous relaxation value of the formulation, and we finish by outlining the similarities between the resulting version of **APT** and model **BM**.

Key idea: Similarly to **BM** and **FLOW-PCC**, model **APT** requires one decision variable for every item $i \in \mathcal{N}$ and for every row $q \in \mathcal{H}_i$ that indicates whether the first slice of item i is packed in row q. Like in **BM**, contiguity is implicitly taken into account in the model by assuming that, if the first slice of item i is packed in row q, then rows $q+1, q+2, \ldots, q+h_i-1$ also contain a slice of item i. The main difference between **APT** and **BM** lies in the way the models handle the objective function: whereas the latter can efficiently use a unique decision variable z in both the objective function (10) and the capacity constraints (12), the former requires one decision variable for every bin (or row) so that it can keep track of the total number of bins used in the solution.

Variables and mathematical formulation: Model **APT** requires: (i) binary decision variables x_{iq} taking value 1 if row q is the row in which the first slice of item i is packed, and value 0 otherwise

($i \in \mathcal{N}, q \in \mathcal{H}_i$) and (ii) binary decision variables z_q taking value 1 if row $q-1$ is the last row that is engaged by any item ($q \in \mathcal{H} \cup \{\text{UB}\}$). The model can be defined as follows:

$$\min \quad \sum_{q \in \mathcal{H} \cup \{\text{UB}\}} q z_q \tag{21}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{H} \cup \{\text{UB}\}} z_q = 1, \tag{22}$$

$$\sum_{q \in \mathcal{H}_i} x_{iq} = 1 \qquad i \in \mathcal{N}, \tag{23}$$

$$\sum_{i \in \mathcal{N}} \sum_{q' \in \mathcal{H}_i(q)} w_i x_{iq'} \leq W \qquad q \in \mathcal{H}, \tag{24}$$

$$\sum_{\substack{q' \in \mathcal{H}_i \\ q' \geq q}} x_{iq'} + \sum_{\substack{q' \in \mathcal{H} \cup \{\text{UB}\} \\ q' \leq q + h_i - 1}} z_{q'} \leq 1 \qquad i \in \mathcal{N}, q \in \mathcal{H}_i, \tag{25}$$

$$x_{iq} \in \{0,1\} \qquad i \in \mathcal{N}, q \in \mathcal{H}_i, \tag{26}$$

$$z_q \in \{0,1\} \qquad q \in \mathcal{H} \cup \{\text{UB}\}. \tag{27}$$

The objective function (21) minimizes the index of the first unused row. Constraint (22) makes sure that exactly one row is defined as the first unused row. Demand constraints (23) and capacity constraints (24) are similar to constraints (11) and (12) in **BM**. Constraints (25) make sure that, for a given item i and a given row q, if the first slice of item i is packed in row q or any subsequent row, then every row $q' = 0, \ldots, q + h_i - 1$ should be considered engaged. Note that model **APT** requires $O(n\text{UB})$ variables and $O(n\text{UB})$ constraints and can easily be extended to take item types into account instead of item indices. We also point out that, as suggested by Alvarez-Valdés et al. [2], one can improve the model performance by using a valid lower bound LB on the optimal solution value and setting all variables $z_q = 0, \ldots, \text{LB} - 1$ to 0 (as far as implementation is concerned, one can reduce the model size by not generating such variables). As demonstrated in our computational experiments, doing so greatly reduces the number of constraints (25) and improves the quality of the LP-relaxation value of the model.

Another improvement (which was, to the best of our knowledge, never proposed in the literature) simply consists in using the same strategy as the one adopted in the well-known textbook model for the one-dimensional bin packing problem [36] where variable z_q indicates whether bin q is used or not. The updated version of model **APT**, called **APTP** hereafter is as follows:

$$\min \quad \sum_{q \in \mathcal{H}} z_q \tag{28}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{H}_i} x_{iq} = 1 \qquad i \in \mathcal{N}, \tag{29}$$

$$\sum_{i \in \mathcal{N}} \sum_{q' \in \mathcal{H}_i(q)} w_i x_{iq'} \leq W z_q \qquad q \in \mathcal{H}, \tag{30}$$

$$z_q \leq z_{q-1} \qquad q \in \mathcal{H} \setminus \{0\}, \tag{31}$$

$$x_{iq} \in \{0,1\} \qquad i \in \mathcal{N}, q \in \mathcal{H}_i, \tag{32}$$

$$z_q \in \{0,1\} \qquad q \in \mathcal{H}. \tag{33}$$

Observe that **APTP** has $O(n\text{UB})$ variables and $O(n + \text{UB})$ constraints and can be seen as a direct **BM** adaptation for the 1CBP. We point out that, like in **APT**, one can improve the model performance by using a valid lower bound LB on the optimal solution value and setting all variables $z_q = 0, \ldots, \text{LB} - 1$ to 1.

3.3.2 FLOW-CBP formulation

In the same way model **APTP** can be seen as an adaptation of model **BM** for the 1CBP, model **FLOW-PCC** can also be extended to the 1CBP. This adaptation will be referred to as **FLOW-CBP** hereafter.

Variables and mathematical formulation: Model **FLOW-CBP** uses graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where vertex set $\mathcal{V} = \{0, 1, \ldots, \text{UB}\}$ and where arc set $\mathcal{A}$ is composed of (i) item arcs $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where $\mathcal{A}_i$ contains every arc $(d, d + h_i)$ such that $d \in \mathcal{H}_i$ and (ii) loss arcs $\mathcal{A}_0$ containing every arc $(d, d + 1)$ such that $d \in \mathcal{H}$ If selected in a solution, item arc $(d, d + h_i) \in \mathcal{A}_i$ carries w_i units of flow from node d to node $d + h_i$. If selected in a solution, loss arc $(d, d+1) \in \mathcal{A}_0$ carries 1 unit of flow from node d to node $d+1$ and represents one unit of unused space in row d. For notation purposes, we assume $w_0 = 1$. A feasible **FLOW-CBP** solution can then be defined as a $0 - q$ flow where W flows are sent from node 0 to node q. As far as variables are concerned, model **FLOW-CBP** uses the same binary decision variables x_{ide} and integer decision variables x_{0de} as **FLOW-PCC**, but replaces integer variable z with binary decision variables z_q taking value 1 if node q is the target node of the network (i.e., the node with W incoming and 0 outgoing flows). **FLOW-CBP** can be defined as follows:

$$\min \quad \sum_{q \in \mathcal{H} \cup \{\text{UB}\}} q z_q \tag{34}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{H} \cup \{\text{UB}\}} z_q = 1, \tag{35}$$

$$\sum_{(d,e) \in \mathcal{A}_i} x_{ide} = 1 \qquad i \in \mathcal{N}, \tag{36}$$

$$\sum_{i \in \{0\} \cup \mathcal{N}} \sum_{\substack{(d,e) \in \mathcal{A}_i \\ d=0}} w_i x_{ide} = W, \tag{37}$$

$$\sum_{i \in \{0\} \cup \mathcal{N}} \sum_{\substack{(d,e) \in \mathcal{A}_i \\ d=q}} w_i x_{ide} + W z_q = \sum_{i \in \{0\} \cup \mathcal{N}} \sum_{\substack{(d,e) \in \mathcal{A}_i \\ e=q}} w_i x_{ide} \qquad q \in \mathcal{V} \setminus \{0\}, \tag{38}$$

$$x_{0de} \in \mathbb{N}_0 \qquad (d, e) \in \mathcal{A}_0, \tag{39}$$

$$x_{ide} \in \{0, 1\} \qquad i \in \mathcal{N}, (d, e) \in \mathcal{A}_i, \tag{40}$$

$$z_q \in \{0, 1\} \qquad q \in \mathcal{H} \cup \{\text{UB}\}. \tag{41}$$

We outline the minor change in flow conservation constraints (38) that was required in order to identify the target node of the network. We also point out that, like in **APT** and **APTP**, one can improve the model performance by using a valid lower bound LB on the optimal solution value and setting all variables $z_q = 0, \ldots, \text{LB} - 1$ to 0.

Overall, it appears that most ILP models for the P|*cont*|$\text{C}_{\max}$ can easily be adapted to solve the 1CBP and vice versa. Models solving the P|*cont*|$\text{C}_{\max}$ have the advantages of (i) not requiring a valid upper bound UB, (ii) having a number of variables and constraints that both depend on W instead of UB (most SPP instances proposed in the literature have the strip width W significantly lower than the optimal solution value of the instance z_{opt}), and (iii) only requiring one variable z to model the objective function of the problem. Models solving the 1CBP have the main advantage that several variables can be fixed when a valid lower bound is available.

3.4 CP formulations for the P|*cont*|C_{max} and the 1CBP

CP has received a lot of attention from the research community in the last two decades, even though the peak of interest seems to have passed already. Nevertheless, the recent literature has shown that CP is particularly effective to solve a targeted set of highly constrained combinatorial optimization problems [43, 54], especially in the scheduling area.

Key idea: Well-known scheduling problems such as the job-shop [83], the flow-shop [81], and the open-shop [3] scheduling problems are commonly encountered in production settings and fall under the umbrella of the *Resource-Constrained Project Scheduling Problem* (RCPSP). In the RCPSP, one wants to determine a schedule that minimizes a given objective function (e.g., the total makespan) such that a number of practical constraints (e.g., precedence and resource constraints) are satisfied. The P|*cont*|C_{max} can be seen as a special version of the RCPSP with a unique renewable resource with availability z in which a task with duration w_i consuming h_i resources per time unit is created for every item $i \in \mathcal{N}$. In that version, the makespan of the schedule is limited to W and the objective is to minimize the resource availability z. As far as the 1CBP is concerned, the problem corresponds to the more standard version of the RCPSP where one resource with availability W is considered and where a task with duration h_i consuming w_i resources per time unit is created for every item $i \in \mathcal{N}$. In that version, it is the makespan of the schedule that is to be minimized. Note that precedence constraints are not present in either version.

Variables and mathematical formulations: Both the CP model solving the P|*cont*|C_{max} (called **CP-PCC** hereafter) and the CP model solving the 1CBP (called **CP-CBP** hereafter) use the following variables and constraint types:

- ivl_i, interval variable that represents the execution of task $i \in \mathcal{N}$;
- `setStartMin`(ivl_i, ℓ): forces task i to start at time ℓ or later (similarly, `setStartMax` forces task i to start at time ℓ or earlier);
- `setSizeMin`(ivl_i, ℓ): forces the duration of task i to be at least ℓ (similarly, `setSizeMax` forces the duration of task i to be at most ℓ);
- `endof`(ivl_i): indicates the completion time of task i;
- `pulse`(ivl_i, ℓ): counts ℓ units of occupation from the starting time of task i to its completion time.

Models **CP-PCC** (on the left) and **CP-CBP** (on the right) can be defined as follows:

$$\begin{aligned}
\min\ & z && && (42)\\
\text{s.t.}\ & \texttt{setStartMin}(ivl_i, 0) && i \in \mathcal{N}, && (43)\\
& \texttt{setStartMax}(ivl_i, W - w_i) && i \in \mathcal{N}, && (44)\\
& \texttt{setSizeMin}(ivl_i, w_i) && i \in \mathcal{N}, && (45)\\
& \texttt{setSizeMax}(ivl_i, w_i) && i \in \mathcal{N}, && (46)\\
& \sum_{i \in \mathcal{N}} \texttt{pulse}(ivl_i, h_i) \leq z. && && (47)
\end{aligned}$$

$$\begin{aligned}
\min\ & z && && (48)\\
\text{s.t.}\ & \texttt{setStartMin}(ivl_i, 0) && i \in \mathcal{N}, && (49)\\
& \texttt{endof}(ivl_i) \leq z && i \in \mathcal{N}, && (50)\\
& \texttt{setSizeMin}(ivl_i, h_i) && i \in \mathcal{N}, && (51)\\
& \texttt{setSizeMax}(ivl_i, h_i) && i \in \mathcal{N}, && (52)\\
& \sum_{i \in \mathcal{N}} \texttt{pulse}(ivl_i, w_i) \leq W. && && (53)
\end{aligned}$$

Whereas the two CP models appear to have less variables and constraints than their ILP counterparts, we point out that such a size comparison does not really make sense given the fact that the methodology used by ILP solvers differ significantly from the methodology used by CP solvers.

4 Model enhancements

A large part of the recent literature focusing on mathematical models for combinatorial optimization problems contains a dedicated section that introduces ad hoc techniques aimed at improving the "performance" of the proposed models (by performance, we mean the average computation time required by a state-of-the-art solver to solve a given instance of the model to optimality). Such techniques can be used, for example, to reduce the size of the model (i.e., the number of variables, the number of constraints, or the number of non-zero elements in the coefficient matrix), to strengthen the model continuous relaxation bound, or to remove some symmetry in the model solution space.

Several of these techniques were proposed in the literature for the SPP, and a subset of those were also extended to the P|*cont*|C_{max} and the 1CBP. We emphasize that, since the P|*cont*|C_{max} and the 1CBP are two relaxations of the SPP, a valid preprocessing for the SPP is not necessarily valid for the P|*cont*|C_{max} or the 1CBP. In this section, we present four sets of model enhancement techniques that can be applied to (some of) the previously introduced models. The first set shows how one can exploit the item multiplicity in order to reduce the model size. The second set fixes some decision variables to positive integer values, which reduces the model size, while also adjusting some instance dimensions, which strengthens the model continuous relaxation value. The third set fixes some decision variables to zero, reducing the model size. The last set is composed of various symmetry-breaking constraints. We conclude the section with a table that identifies the enhancement techniques that are compatible with each of the previously introduced mathematical models. Every technique in this section is presented for model **BM**, but can easily be extended to the other models unless stated otherwise.

4.1 Exploiting item multiplicity

Item multiplicity arises when an instance contains several items with the exact same dimensions. Depending on the modelling strategy, it is sometimes possible to gather all the items with the same dimensions into a unique item type together with a corresponding demand. By doing so, one may reduce the overall size of the model as one variable set per item type is needed instead of one variable set per item (see, e.g., the well-studied cases of the bin packing and cutting stock problems [36]). Note, however, that when items have two dimensions or more, the number of item types is very likely to be close to (if not equal to) the number of items, unless item multiplicity is an explicit feature of the instance.

As observed in Section 3, it is trivial to take the item multiplicity into account in **BM** by replacing binary decision variables x_{ip} ($i \in \mathcal{N}, p \in \mathcal{W}_i$) with integer decision variables x_{jp} ($j \in \mathcal{M}, p \in \mathcal{W}_j$) and replacing the right-hand-side of constraints (11) by d_j, the demand of item type j. Note that exploiting the item multiplicity does not always improve the performance of a model as the solver-specific inner machinery seems to be more effective when dealing with binary variables than it is when dealing with integer variables. In addition, an integer variable cannot be included in a no-good cut, which can be a significant issue if one wishes to solve the P|*cont*|C_{max} or the 1CBP as the MP of a decomposition algorithm. A well-known solution consists in using a binary expansion where every integer variable x_{jp} ($j \in \mathcal{M}, p \in \mathcal{W}_j$) is replaced by a set of binary variables x^B_{jpg} ($j \in \mathcal{M}, p \in \mathcal{W}_j, g = 0, \ldots, \lfloor \log_2(d_j) \rfloor$) such that:

$$x_{jp} = \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} 2^g x^B_{jpg} \quad j \in \mathcal{M}, p \in \mathcal{W}_j \tag{54}$$

Binary expansion and integer variables can be used in all models presented in Section 3 except **BKRS** and the CP formulations as CP does not have a practical way to handle duplicated interval variables.

4.2 Prepacking large items and adjusting the instance dimensions

Prepacking large items, reducing the strip dimensions, and increasing the item dimensions are common reduction procedures for the SPP [16, 27, 37]. In the following, we briefly describe how each technique can also be used when solving the P|*cont*|C_{max} and the 1CBP.

Prepacking large items: Following an idea introduced by Boschetti and Montaletti [16], in the SPP, one may prepack a set of "large" items $\mathcal{S}$ (an item i is said to be large if $w_i > \frac{W}{2}$) at the bottom of the strip if there exists a feasible packing with height $\sum_{i\in\mathcal{S}} h_i$ that contains every "$\mathcal{S}$-compatible" item (an item i' is said to be $\mathcal{S}$-compatible if at least one item in $\mathcal{S}$ can be packed side by side with i', or in other words, if $\min_{i\in\mathcal{S}}\{w_i\} + w_{i'} \leq W$). Indeed, if such a packing exists, no other item can be packed side by side with any of the large items in $\mathcal{S}$ (meaning that the empty space cannot be utilized), and one can therefore find an optimal solution for the SPP in which the items in $\mathcal{S}$ are packed together with the $\mathcal{S}$-compatible items in an isolated block.

The same idea may be used for the P|*cont*|C_{max}: one may also prepack a set of large items $\mathcal{S}$ at the bottom of the columns if there exists a feasible solution with makespan $\sum_{i\in\mathcal{S}} h_i$ that contains every $\mathcal{S}$-compatible item. Indeed, even though one might think that the unused space could be utilized by packing (up to) $W - \min_{i\in\mathcal{S}}\{w_i\}$ slices of an item i' that is not $\mathcal{S}$-compatible, doing so would simply shift the unused space on these (up to) $W - \min_{i\in\mathcal{S}}\{w_i\}$ columns by $h_{i'}$ units as one slice of item i' needs to be packed in the subsequent column because of the contiguity constraints, increasing the makespan of that column to $\sum_{i\in\mathcal{S}} h_i + h_{i'}$. Note that in the recognition version of the problem, an additional preprocessing step can be applied in which W is replaced by UB and w_i is replaced by h_i. That idea may be also be used for the 1CBP: one may prepack a set of large items $\mathcal{S}$ in the first rows if there exists a feasible solution with $\sum_{i\in\mathcal{S}} h_i$ bins that contains every $\mathcal{S}$-compatible item (the definitions of large and $\mathcal{S}$-compatible are unchanged).

Reducing W: Following an idea introduced by Alvarez-Valdés et al. [2], one can decrease W by one unit as long as there does not exist any subset of items whose total width is equal to W in both the P|*cont*|C_{max} and the 1CBP. In practice, this can be done by solving the following knapsack problem:

$$\max \sum_{i\in\mathcal{N}} w_i\xi_i : \sum_{i\in\mathcal{N}} w_i\xi_i \leq W, \xi_i \in \{0,1\}\ (i \in \mathcal{N}) \tag{55}$$

and by setting W to the optimal solution value of the problem. In the recognition version of the two problems, this preprocessing step can also be applied to UB (in that case, w_i should be replaced by h_i).

Increasing the item dimensions: Following an idea introduced by Boschetti et al. [15], one can also increase the width of an item i' by one unit as long as there are no combinations of (other) item heights whose sum is equal to $W - w_{i'}$ in both the P|*cont*|C_{max} and the 1CBP. In practice, this can also be done by solving a knapsack problem for every item $i' \in \mathcal{N}$:

$$\max \sum_{i\in\mathcal{N}} w_i\xi_i : \sum_{i\in\mathcal{N}\setminus\{i'\}} w_i\xi_i \leq W - w_{i'}, \xi_i \in \{0,1\}\ (i \in \mathcal{N} \setminus \{i'\}) \tag{56}$$

and by setting $w_{i'}$ to W minus the optimal solution value of the problem. In the recognition version of the two problems, this preprocessing step can also be applied to $h_{i'}$ (in that case, w_i should be replaced by h_i and W should be replaced by UB). Note that, if one aims at exploiting item multiplicity, one should be careful when applying this preprocessing step as it may create new item types. Consider for example an instance with 3 items where $W = 30$ and $w_1 = w_2 = w_3 = 9$: after applying this preprocessing, $w_1 = 12$

and $w_2 = w_3 = 9$, which results in a new item type. One may therefore opt for alternative solutions, such as applying this preprocessing on the item types with unit demand first, or distributing the width increase among all the items with the same item type, if possible, resulting in $w_1 = w_2 = w_3 = 10$.

4.3 Reducing the number of variables

One may observe that the model size is directly impacted by the size of sets $\mathcal{W}, \mathcal{W}_i, \mathcal{H}$, and $\mathcal{H}_i$. Whereas a trivial definition of these sets was proposed in Section 3, more elaborated versions (with a reduced size) can be built by using the concepts of *canonical dissection*, *normal patterns*, and *meet-in-the-middle patterns* introduced by Herz [49], Christofides and Whitlock [23] and Côté and Iori [30], respectively. Given one of these enhanced sets, it was shown in the corresponding introductory paper that there always exists at least one optimal solution in which every item is packed in a coordinate belonging to the set.

Normal patterns: Similar to canonical dissection, normal patterns use the idea that, given a feasible solution for the SPP, every item can be pushed as much downward and as much to the left as possible without deteriorating the solution value, and therefore, that there always exists an optimal SPP solution in which no item can be pushed further downward or to the left. A necessary condition for a solution to display such a property is that every item should be packed in a coordinate that is a combination of the other item dimensions. Normal patterns can be used to reduce the size of the P|*cont*|$\mathrm{C_{max}}$ models (resp., the 1CBP models) introduced in Section 3 by defining sets $\mathcal{W}_{i'}$ (resp., $\mathcal{H}_{i'}$) for every item $i' \in \mathcal{N}$ such that the set contains every abscissa w (resp., every ordinate h) satisfying:

$$w = \sum_{i \in \mathcal{N} \setminus \{i'\}} w_i \xi_i : w \leq W - w_{i'}, \xi_i \in \{0,1\} \ (i \in \mathcal{N} \setminus \{i'\}) \tag{57}$$

$$\Big(\text{resp., } h = \sum_{i \in \mathcal{N} \setminus \{i'\}} h_i \xi_i : h \leq UB - h_{i'}, \xi_i \in \{0,1\} \ (i \in \mathcal{N} \setminus \{i'\})\Big) \tag{58}$$

Meet-in-the-middle patterns: Sometimes referred to as "MIM patterns" in the literature [30], the meet-in-the-middle patterns improve upon the normal patterns by alleviating one of the main drawbacks of the method: the further away from the origin a packing coordinate is, the more likely it is to be included in the normal patterns. To do so, given a certain threshold, every item packed on the left side of the threshold is pushed as much to the left as possible whereas every item packed on the right side of the threshold is pushed as much to the right as possible. The name "meet-in-the-middle" comes from the fact that a given MIM pattern for a certain instance of a given problem (say, for example, the P|*cont*|$\mathrm{C_{max}}$) is composed of two normal patterns: one from 0 to the threshold t corresponding to the starting coordinate of the items packed on the left side of the threshold, and one from W to $W-t$ corresponding to the ending coordinate of the items packed on the right side of the threshold, the two sets meeting in the middle (or, more precisely, meeting in t). In practice, one tries every possible threshold t and uses the one that minimizes a certain evaluation criterion (such as $\sum_{i \in \mathcal{N}} |\mathcal{W}_i|$ or $|\cup_{i \in \mathcal{N}} \mathcal{W}_i|$ for the P|*cont*|$\mathrm{C_{max}}$). Since the MIM patterns are equivalent to the normal patterns when the threshold t is set to 0 or to W, using the former is, in theory, at least as good as using the latter based on the chosen evaluation criterion. We point out that MIM patterns by Côté and Iori [30] cannot directly be used for the 1CBP. Indeed, pushing as much to the right as possible the items packed on the right side of a given threshold may increase the 1CBP objective function value, which was not the case for the P|*cont*|$\mathrm{C_{max}}$. We illustrate this behaviour in an example available in Appendix A.

We also outline that sets $\mathcal{W}$ (resp. $\mathcal{H}$) may sometimes also be reduced thanks to normal or MIM patterns by setting $\mathcal{W} = \cup_{i \in \mathcal{N}} \{\mathcal{W}_i\}$ (resp. $\mathcal{H} = \cup_{i \in \mathcal{N}} \{\mathcal{H}_i\}$). Note, however, that such a reduction is not always valid: for example, it cannot be applied in constraints (31) of **APTP**.

4.4 Symmetry-breaking constraints

Symmetry is a common phenomenon in combinatorial optimization that is often seen as problematic because it significantly increases the size of the search space. For a given ILP formulation, we say that two distinct feasible solutions are symmetric if one can be obtained from the other after applying a certain transformation. For example, in the well-known graph coloring problem, permuting the colors of a given solution produces a symmetric solution. Similarly, in the classical vehicle routing problem, permuting the routes of two identical vehicles produces a symmetric solution. In the P|*cont*|C_{max}, the solution displayed in Figure 1d and the solution obtained after rotating the packing layout by 180° are also symmetric.

Symmetry-breaking constraints are supplementary cuts added to a model with the objective of decreasing (or sometimes completely removing) the presence of symmetry in the search space of the model. Note that using such constraints is not always beneficial in practice since (i) it can make it harder for the solver heuristics to find good quality solutions and (ii) the extra time spent solving each node of the solution tree (caused by the supplementary cuts) can be longer than the time earned by not exploring the nodes corresponding to symmetric solutions.

Gapless packing: In some aspects, using normal patterns to define sets $\mathcal{W}, \mathcal{W}_i, \mathcal{H}$, and $\mathcal{H}_i$ can be seen as a form of symmetry-breaking constraints: every optimal solution in which at least one item is not packed in a normal pattern has an equivalent solution that does. Nevertheless, using normal patterns does not remove all forms of symmetry: it is true that there is always a gap (i.e., an item that can be pushed further to the left) in an optimal solution in which at least one item is not packed in a normal pattern, but there are also optimal solutions in which every item is packed in a normal pattern that do have a gap too. Boschetti & Montaletti [16] proposed two symmetry-breaking constraints to reduce the number of gaps in an SPP solution. Similar constraints can be used in **BM** for the P|*cont*|C_{max} as follows:

$$\sum_{i \in \mathcal{N}} x_{i0} \geq 1, \tag{59}$$

$$\sum_{\substack{i' \in \mathcal{N} \\ p - w_{i'} \in \mathcal{W}_{i'}}} x_{i',p-w_{i'}} \geq x_{ip} \qquad i \in \mathcal{N}, p \in \mathcal{W}_i \setminus \{0\}. \tag{60}$$

Constraint (59) ensures that the first slice of at least one item is packed in the first column whereas constraints (60) make sure that if the first slice of an item is packed in column p ($p > 0$), then column $p-1$ contains the last slide of at least one item. Considering the (potentially very) high number of constraints (60), Boschetti and Montaletti [16] suggested to only add such constraints if their number was below a certain threshold. Belov et al. [10] proposed instead to merge those constraints so as to obtain:

$$M \sum_{\substack{i' \in \mathcal{N} \\ p - w_{i'} \in \mathcal{W}_{i'}}} x_{i',p-w_{i'}} \geq \sum_{\substack{i \in \mathcal{N} \\ p \in \mathcal{W}_i}} x_{ip}, \quad p \in \mathcal{W} \setminus \{0\}, \tag{61}$$

where the big-M coefficient may be set to n. For the CP models, such constraints may be obtained with `StartAtEnd`$(ivl_i, ivl_{i'})$, which enforces task i' to start when task i ends, together with the `OR` operator.

Mirror symmetry: Every optimal solution for the SPP has an obvious symmetric solution in which the packing layout is rotated by 180°. Belov et al. [10] used that observation to derive a symmetry-breaking constraint that forces an arbitrary item to be packed in a coordinate located in the first half of the strip. A similar idea can be used in the P|*cont*|C_{max} by picking an arbitrary item i and adding one of the two following constraints to the model:

$$\sum_{\substack{p\in\mathcal{W}_i \\ p\leq\lfloor(W-w_i)/2\rfloor}} x_{ip} = 1, \qquad (62) \qquad\qquad \sum_{\substack{p\in\mathcal{W}_i \\ p>\lfloor(W-w_i)/2\rfloor}} x_{ip} = 0. \qquad (63)$$

From an implementation point of view, using (62) adds one extra constraint whereas using (63) reduces the number of variables. If identical items are merged, mirror symmetry can be avoided by using:

$$\sum_{\substack{p\in\mathcal{W}_j \\ p\leq\lfloor(W-w_j)/2\rfloor}} x_{jp} \geq \left\lceil \frac{d_j}{2} \right\rceil, \qquad (64)$$

for an arbitrary item type j. For the CP models, mirror symmetry can be avoided with `setStartMax`(ivl_i, $\lfloor(W - w_i)/2\rfloor$) for an arbitrary item i, which forbids task i to start after time $\lfloor(W - w_i)/2\rfloor$.

Identical items: In case one cannot (or does not want to) exploit item multiplicity by merging the identical items, then a valid symmetry-breaking constraint forbids every solution in which there exists a pair of identical items (i, i') in which the first slice of i is packed in column p and the first slice of i' is packed in column p' such that $i < i'$ and $p > p'$. In **BM**, such a constraint can be modelled as follows:

$$\sum_{p\in\mathcal{W}_i} p x_{ip} \leq \sum_{p'\in\mathcal{W}_{i'}} p' x_{i'p'} \qquad i \in \mathcal{N}, i' \in \mathcal{N} : i < i', w_i = w_{i'}, h_i = h_{i'}. \qquad (65)$$

For the CP models, such constraints may be obtained using constraint type `startBeforeStart` ($ivl_i, ivl_{i'}$), which forbids task i to start later than task i'.

4.5 Compatibility table

We conclude this section with a summary table that indicates, for each of the models introduced in Section 3, whether the aforementioned model enhancement techniques can be applied.

Table 2: Compatibility between mathematical models and model enhancements

Type	Model enhancement	P|*cont*|C_{max}				1CBP			
		BKRS	BM	FLOW	CP-PCC	APT	APTP	FLOW-CBP	CP-CBP
Item multiplicity	Binary expansion		✓	✓		✓	✓	✓	
	Integer demand		✓	✓		✓	✓	✓	
Preprocessing	Prepacking large items	✓	✓	✓	✓	✓	✓	✓	✓
	Reducing W	✓	✓	✓	✓	✓	✓	✓	✓
	Increasing the item dimensions	✓	✓	✓	✓	✓	✓	✓	✓
Variable reduction	Normal patterns	✓	✓	✓	✓	✓	✓	✓	✓
	Meet-in-the-middle patterns	✓	✓	✓	✓				
Symmetry breaking	Gapless packing	✓	✓	✓	✓	✓	✓	✓	✓
	Mirror symmetry	✓	✓	✓	✓	✓	✓	✓	✓
	Identical items	✓	✓	✓	✓	✓	✓	✓	✓

We point out that some of the listed techniques are not always pairwise compatible, meaning that they cannot be used together in the same model unless some non-trivial adjustments are made, or they cannot be used together in the same model at all. For example, the symmetry-breaking constraints enforcing "gapless packing" requires some adjustments in order to be used together with the MIM patterns. Similarly, if one exploits item multiplicity (either with integer variables or with binary expansion), then the symmetry-breaking constraint aimed at ordering the identical items cannot be applied.

5 Lower bounds

Various lower bounds have been proposed in the literature for the SPP, two of the strongest ones resulting from the P|*cont*|C_{max} and the 1CBP relaxations. In the following, we review the SPP lower bounds that can be adapted for the P|*cont*|C_{max} and the 1CBP.

5.1 Lower bounds for the P|*cont*|C_{max}

L_1: A trivial lower bound for the P|*cont*|C_{max} can be obtained by selecting the maximum value between (i) the quotient $\frac{\sum_{i \in \mathcal{N}} h_i w_i}{W}$ rounded up to the nearest integer and (ii) the largest item height $\max_{i \in \mathcal{N}}\{h_i\}$.

L_2^{PCC}: As observed by Alvarez-Valdés et al. [2], a better lower bound for the P|*cont*|C_{max} can be obtained by relaxing the contiguity constraints and solving the P||C_{max}. This bound is called L_2^{PCC} hereafter. Even though the P||C_{max} is an $\mathcal{NP}$-hard combinatorial optimization problem, large-sized instances can be solved with state-of-the-art optimization algorithms. For instance, after observing that the recognition version of the P||C_{max} is the same as the recognition version of the one-dimensional cutting stock problem, one can use the concept of a *destructive bound* and iteratively solve one-dimensional cutting stock problems (e.g., by using one of the techniques described in [36]) until an optimal solution for the P||C_{max} is found. In practice, one sets the bin capacity to a valid lower bound (e.g., L_1) and increases it by one unit until the optimal solution value of the cutting stock problem is at most W.

$L_2^{PCC'}$: Considering that there exists ILP formulations for the cutting stock problem with a very tight LP-relaxation value (e.g., it is conjectured that the LP-relaxation of the arcflow formulation rounded up to the nearest integer is at most one bin away from the optimal solution value, see [36]), we also considered an alternative bound $L_2^{PCC'}$ in which the bin capacity is increased until the LP-relaxation of the arcflow model is at most W. A similar strategy was proposed by Côté et al. [27] who derived a lower bound for the SPP (which is also valid for the P|*cont*|C_{max}) by solving the LP-relaxation of a set-covering-like formulation with a column generation procedure.

L_3^{PCC} and $L_3^{PCC'}$: As also observed by Alvarez-Valdés et al. [2], an even better lower bound for the P|*cont*|C_{max} can be obtained by replacing the contiguity constraints with conflict constraints and solving the P||C_{max} with conflicts. This bound is called L_3^{PCC} hereafter. As in L_2^{PCC}, one can notice that the recognition version of the P||C_{max} with conflicts is the same as the recognition version of the one-dimensional bin packing problem with conflicts. Therefore, an optimal solution for the P||C_{max} with conflicts can be obtained by solving multiple bin packing problems with conflicts. Brandao and Pedroso [17] showed how the bin packing problem with conflicts could be reduced to the vector packing problem and proposed an extension of the arcflow formulation to solve the latter problem. That extension is particularly effective to compute L_3^{PCC} because of the peculiar structure of its conflict graph (it is composed of a set of n disjoint cliques), resulting in a limited number of vector dimensions (one for the item weights and n for the conflicts). Since that extension of the arcflow model also displays a very good LP-relaxation quality, we also considered an alternative bound $L_3^{PCC'}$ in which the bin capacity is increased until the LP-relaxation of the arcflow extension is at most W.

L_4^{PCC} and $L_4^{PCC'}$: Delorme and Iori [35] introduced *reflect*, an extension of the arcflow formulation for the cutting stock problem that uses a pair of 0-t and 0-t' paths where $t + t' = W$ to model a bin instead of a single 0-W path like in the arcflow model. The authors demonstrated that using such a strategy produces smaller ILP models that can be solved faster with a state-of-the-art ILP solver. Since model

FLOW-PCC is largely inspired by the arcflow model, we also tested an extension of *reflect* tailored to the P|*cont*|C_{max} referred to as **REFLECT** hereafter. The model is described in Appendix B. Because of the h_i coefficient in the flow conservation constraints, a feasible **REFLECT** solution with objective value z cannot always be converted into a feasible **FLOW-PCC** solution with the same objective value (see the example provided in Appendix B). Nevertheless, the solution value produced by **REFLECT** can be used as a valid lower bound for the P|*cont*|C_{max}. This bound is called L_4^{PCC} hereafter. Inspired by the previous bounds, and since computing L_4^{PCC} can take a long time in practice, we also considered lower bound $L_4^{PCC'}$ obtained by rounding up the LP-relaxation of **REFLECT**.

5.2 Lower bounds for the 1CBP

As far the 1CBP is concerned, the following lower bounds (which are adaptations or copies of the previously described bounds) can be used:

- $L_1 = \max\{\max_{i\in\mathcal{N}}\{h_i\}, \left\lceil \frac{\sum_{i\in\mathcal{N}} h_i w_i}{W} \right\rceil\}$;
- L_2^{CBP} is the bound obtained after relaxing the contiguity constraints and solving the one-dimensional cutting stock problem with the arcflow model;
- $L_2^{CBP'}$ is the bound obtained after solving the LP-relaxation of the arcflow model used to solve the one-dimensional cutting stock problem;
- L_3^{CBP} is the bound obtained after replacing the contiguity constraints by conflict constraints and solving the bin packing problem with conflicts with an extension of the arcflow model;
- $L_3^{CBP'}$ is the bound obtained after solving the LP-relaxation of the arcflow extension used to solve the bin packing problem with conflicts.

We point out that bounds L_4^{CBP} and $L_4^{CBP'}$ do not seem interesting to compute in practice because one needs to fix the number of bins in order to run the reflect extension, whereas the number of bins is not an information known in the 1CBP. We also emphasize that one iteration of the reflect extension is time-consuming, limiting the interest of applying an iterative procedure in which the number of bins is incremented until a stopping criterion is reached (as was done for L_2^{PCC} and L_3^{PCC}).

6 Computational experiments

In this section, we thoroughly evaluate the performance of each of the proposed methods, with and without the reviewed enhancement techniques, and we report the outcome of these experiments in seven subsections. The first part evaluates the performance of a baseline version of each P|*cont*|C_{max} and 1CBP models introduced in Section 3. The second part measures how effective are the variable reduction techniques introduced in Section 4.3. The third part demonstrates the empirical improvements that can be obtained after exploiting the item multiplicity as described in Section 4.1. The fourth part assesses whether the symmetry-breaking constraints introduced in Section 4.4 are beneficial in practice. The fifth part tests the usefulness of two popular (but non-tailored) optimization techniques: destructive bounds and reduced-cost variable fixing. The sixth part identifies the combination of approaches that solves the maximum number of instances in the tested dataset. It also measures how the results vary when changing the random seed of the solver. The last part evaluates the quality of the lower bounds introduced in Section 5. Every approach was implemented in `C++` and can be downloaded from `https://github.com/mdelorme2/Parallel_processor_scheduling_and_bin_packing_problems_with_contiguity`. All computational tests were

executed on a single thread of a virtual machine `AMD EPYC-Rome Processor` with 2.00 GHz and 64 GB of RAM memory, running under `Ubuntu 20`. The ILP models were solved using `Gurobi 10.0.1` whereas the CP models were solved using `IBM ILOG CPLEX CP Optimizer 22.1.1`. In each run, a time limit of 3600 seconds was imposed. The experiments were performed on the following two-dimensional packing datasets:

- **NGCUT:** a set of twelve instances proposed by Beasley [9];
- **CGCUT:** a set of three instances proposed by Christofides and Whitlock [23];
- **GCUT:** a set of thirteen instances proposed by Beasley [8];
- **BENG:** a set of ten instances proposed by Bengtsson [12];
- **HT:** a set of nine instances proposed by Hopper and Turton [50];
- **BKW:** a set of thirteen SCP instances proposed by Burke et al. [19];
- **CLASS1,. . . ,CLASS10:** ten sets, each consisting of fifty instances, proposed by Berkey and Wang [13] (first six classes) and Martello and Vigo [62] (last four classes). Because CLASS4, CLASS6, and CLASS8 are notoriously difficult datasets [27], these are only tested in a subset of the experiments.

The main features of each class are summarized in Table 3. For each dataset, we report the number of instances contained in the dataset and the average $n, m, W, w_{min}, w_{max}, h_{min}$, and h_{max} values.

Table 3: Dataset features

Dataset	#inst.	n	m	W	$w_{\min}$	$w_{\max}$	$h_{\min}$	$h_{\max}$
CGCUT	3	33.7	12	50	6.3	28.3	6.7	24
HT	9	23.2	21	40	1.8	17.9	1.4	10.7
BENG	10	90	52.9	32.5	1	8	1	12
NGCUT	12	14.4	7.3	17.5	1.3	13.7	2.4	17.3
BKW	13	354.8	58.1	110.8	3.1	49.8	3.1	56.1
GCUT	13	27.8	27.8	769.2	161.6	538.4	171.3	456.5
CLASS1	50	60	44.1	10	1	10	1	10
CLASS2	50	60	44.1	30	1	10	1	10
CLASS3	50	60	58.5	40	1.4	34.6	1.5	34.4
CLASS4	50	60	58.5	100	1.4	34.6	1.5	34.4
CLASS5	50	60	59.9	100	2.6	98.4	2.9	97.7
CLASS6	50	60	59.9	300	2.6	98.4	2.9	97.7
CLASS7	50	60	59.2	100	4.7	99.4	1.8	95.6
CLASS8	50	60	59.5	100	1.9	94.9	6.7	99.4
CLASS9	50	60	59.6	100	7	99.3	6.7	99
CLASS10	50	60	59.5	100	1.9	96.2	2.2	96.4

6.1 Baseline version of the models

6.1.1 $\mathbf{P}|cont|\mathbf{C}_{\max}$

We report in Table 4 the results obtained by a baseline version of **BKRS**, **BM**, **FLOW-PCC**, and **CP-PCC** on a subset of the above-mentioned datasets. Only the three preprocessing techniques described in Section 4.3 aimed at prepacking large items, reducing W, and increasing the item dimensions were included in the baseline version. The first two columns of the table identify the dataset and the number of instances included in the dataset. The following columns provide, for each of the tested models, the number of optimal solutions found (column "#opt"), the average CPU time in seconds over each run of the dataset including (column "T(s)") or excluding (column "T_{opt}(s)") the ones terminated by the time limit or the memory limit. We also report in columns **BEST** the results obtained by the best approach among the

Table 4: Results of **BKRS**, **BM**, **FLOW-PCC**, **FLOW-PCC** and **BEST** on P|*cont*|C_{max} instances

Dataset	#inst	**BKRS**			**BM**			**FLOW-PCC**			**CP-PCC**			**BEST**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	1	2400	0	2	1201	1	2	1250	75	1	2400	0	2	1201	1
HT	9	6	1268	103	8	407	8	8	409	11	8	431	35	8	403	3
BENG	10	8	1399	849	10	8	8	8	778	73	10	1	1	10	1	1
NGCUT	12	12	16	16	12	0	0	12	55	55	12	0	0	12	0	0
BKW	13	2	3109	1	7	2040	700	5	2635	1084	4	2741	809	8	1847	751
GCUT	13	4	2575	89	11	581	22	9	1121	18	9	1108	0	11	570	19
CLASS1	50	50	1	1	50	0	0	50	0	0	44	436	4	50	0	0
CLASS2	50	44	562	148	50	14	14	50	115	115	50	2	2	50	2	2
CLASS3	50	34	1248	141	47	348	141	43	505	1	31	1386	28	47	325	116
CLASS5	50	34	1441	425	48	147	3	47	302	92	39	797	6	48	146	2
CLASS7	50	50	6	6	50	0	0	50	0	0	50	0	0	50	0	0
CLASS9	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS10	50	12	2932	810	24	1907	73	21	2174	204	19	2249	46	25	1810	21
Total	410	307	1015	143	369	396	39	355	535	60	327	743	18	371	373	34

four tested algorithms for every instance. In other words, **BEST** simulates the performance of a hyper-algorithm able to predict with 100% accuracy the methods that is the fastest to solve a given instance.

Further model-specific measures are available in Table 5 where the continuous relaxation (column "cont."), the number of variables (column "#var"), constraints (column "#const"), and non-zero elements in the constraint matrix (column "#nz") are reported for each ILP formulation.

Table 5: Model-specific metrics for **BKRS**, **BM**, and **FLOW-PCC** on P|*cont*|C_{max} instances

Dataset	#inst	**BKRS**				**BM**				**FLOW-PCC**			
		cont.	#var	#const	#nz	cont.	#var	#const	#nz	cont.	#var	#const	#nz
CGCUT	3	240.1	4074	2154	12 236	245.2	1357	78	28 773	245.2	1406	84	4169
HT	9	21.7	2019	1095	6070	21.7	842	62	7829	21.7	881	63	2603
BENG	10	89.4	6301	3363	18 843	89.4	2823	123	15 481	89.4	2855	123	8532
NGCUT	12	36.3	400	241	1203	36.5	153	28	884	36.5	169	29	490
BKW	13	177.7	328 189	164 909	984 317	177.7	160 852	459	1 944 411	177.7	160 959	461	482 770
GCUT	13	4240.5	32 893	17 166	99 339	4283.9	10 358	541	4 638 076	4283.9	11 029	692	32 415
CLASS1	50	186.6	565	348	1673	187.2	178	36	846	187.2	187	38	551
CLASS2	50	60.1	3601	1950	10 770	60.1	1530	90	9510	60.1	1559	90	4646
CLASS3	50	503.4	2321	1257	6970	506.7	669	58	9706	506.7	707	68	2083
CLASS5	50	1619.4	4244	2250	12 795	1629.2	1155	78	40 720	1629.2	1234	101	3620
CLASS7	50	1580.9	1006	594	3090	1588.0	255	49	8438	1588.0	327	78	907
CLASS9	50	3340.2	257	187	822	3341.6	60	24	1687	3341.6	104	46	265
CLASS10	50	909.7	10 246	5325	30 784	915.6	3332	142	95 128	915.6	3431	151	10 193
Total	410	1145.6	14 401	7355	43 236	1150.4	6406	96	229 747	1150.4	6478	112	19 361

We observe that **BM** obtains the best results on average as the algorithm is able to solve 369 out of the 410 tested instances. Despite having a theoretical advantage over **BM** when it comes to the number of non-zero elements in the constraint matrix (one order of magnitude less on average at the expense of a few supplementary variables and constraints), **FLOW-PCC** solves 14 instances less than **BM** while also being 21s slower on average. As far as **CP-PCC** is concerned, the approach is able to solve 327 out of the 410 tested instances. Interestingly, **CP-PCC** solves an instance in 18s on average, outlining a "solving fast or never" behavior for CP that was already observed in the literature for other problems. **BKRS** does not appear to be competitive as it only solves 307 instances in total. Note that **BEST** solves 2 instances more than **BM** (through CP-PCC), which indicates that **BM** does not clearly dominate all the other approaches.

When it comes to model-specific measures, we see that **BKRS** requires on average more than twice the number of variables and more than fifty times the number of constraints compared to **BM** and **FLOW-PCC**. This model size increase does not produce any improvement in the continuous relaxation value of the model. We also observe that **BM** and **FLOW-PCC** have comparable model-specific measures, both in terms of model size (the former has slightly less variables and constraints, but more non-zero elements than the latter) and in terms of continuous relaxation value (which was the same for the two models in

all 410 tested instances). We finally point out that there is not a perfect correlation between the size of a model and the ability for the approach using that model to solve an instance to optimality: for example **BM** could not solve instance CGCUT02 (for which 1286 variables and 87 constraints were needed) whereas it could solve instance GCUT08 - (for which 13 414 variables and 423 constraints were needed).

We conclude this first set of experiments for the P|*cont*|C_{max} by assessing the effectiveness of the preprocessing techniques included in the baseline version of our approaches. Indeed, even though these techniques have always been included by default in recent algorithms [16, 27, 37], one may wonder whether those are worthwhile to implement: considering how sophisticated tailored state-of-the-art optimization algorithms have become, one may decide to leave aside a few components if they only provide an epsilon improvement. We therefore tested a version of **BKRS**, **BM**, **FLOW-PCC**, and **CP-PCC** without any of the three preprocessing techniques described in Section 4.3 and provide the outcome of these experiments in Appendix C. In brief, using these preprocessing techniques allows each approach to solve up to seven instances more. As **BKRS** does not seem to offer any competitive advantage over **BM**, we do not test the approach in further experiments.

6.1.2 1CBP

We report in Table 6 the results obtained by a baseline version of **APT**, **APTP**, **FLOW-CBP**, and **CP-CBP** on a subset of the above-mentioned datasets. The three preprocessing techniques described in Section 4.3 and a lower bound $L_2^{CBP'}$ were included in the baseline version. For the ILP models, the necessary upper bound was obtained by using **CP-CBP** until a feasible solution was found.

Table 6: Results of **APT**, **APTP**, **FLOW-CBP**, **CP-CBP**, and **BEST** on 1CBP instances

Dataset	#inst	**APT**			**APTP**			**FLOW-CBP**			**CP-CBP**			**BEST**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1206	9	2	1203	5	2	1210	14	1	2400	0	2	1203	5
HT	9	9	18	18	9	9	9	9	31	31	8	415	17	9	6	6
BENG	10	5	1897	194	8	768	60	3	2577	190	10	36	36	10	36	36
NGCUT	12	12	4	4	12	4	4	12	10	10	12	1	1	12	1	1
BKW	13	4	2649	510	7	2207	1013	4	2500	27	5	2229	36	8	1886	816
GCUT	13	4	2586	305	4	2509	53	3	2770	1	7	1705	81	7	1705	81
CLASS1	50	41	723	91	41	725	94	35	1185	151	50	105	105	50	103	103
CLASS2	50	44	527	108	42	714	164	26	1813	164	49	122	51	50	42	42
CLASS3	50	21	2104	39	20	2164	9	18	2308	12	36	1022	19	36	1022	19
CLASS5	50	20	2161	3	21	2114	62	20	2161	3	34	1158	7	34	1157	7
CLASS7	50	47	219	3	47	237	23	47	275	63	50	1	1	50	1	1
CLASS9	50	50	1	1	50	6	6	50	3	3	50	0	0	50	0	0
CLASS10	50	6	3183	122	6	3171	23	6	3232	536	14	2594	8	14	2594	8
Total	410	265	1309	56	269	1291	80	235	1579	73	326	762	31	332	724	48

We observe that **CP-CBP** obtains the best results on average as it is able to solve 326 out of the 410 tested instances. **APTP** comes second with 269 instances solved, followed by **APT** with 4 instances less. We notice similar trends between these two models (they both solve a comparable number of instances in each dataset), even though **APT** runs out of memory more often than **APTP**. **FLOW-PCC** comes last with 235 instances solved. **BEST** solves 6 instances more than **CP-CBP** (mostly through APTP), which indicates that **CP-CBP** does not clearly dominate all the other approaches. We also point out that: (i) for 328 instances, both the (numerical instance resulting from the) P|*cont*|C_{max} relaxation and the (numerical instance resulting from the) 1CBP relaxation could be solved by **BEST**, (ii) for 43 instances, the P|*cont*|C_{max} relaxation could be solved by **BEST**, but not the 1CBP relaxation, (iii) for 4 instances, the 1CBP relaxation could be solved by **BEST**, but not the P|*cont*|C_{max} relaxation, and (iv) for 35 instances, neither relaxations could be solved by **BEST**. This indicates that the 1CBP relaxation of the tested instances seems to be harder to solve than the P|*cont*|C_{max} relaxation overall, even though this is not

the case for all instances. From a practical point of view, this demonstrates that an effective decomposition approach for the SPP could use both the 1CBP and the P|*cont*|C_{max} as MP.

We also tested the effectiveness of providing $L_2^{CBP'}$ to the approaches by running a version of each of the four methods in which the lower bound was set to 0. The outcome of these experiments, together with some model-specific measures, are reported in Appendix C. In summary, using $L_2^{CBP'}$ allows each approach to solve between 12 and 38 instances more. As far as model-specific measures are concerned, **APTP** is better on average than **APT** according to both the size of the model and its continuous relaxation value. Like for their P|*cont*|C_{max} counterparts, **APTP** and **FLOW-CBP** have the same continuous relaxation values for all 410 tested instances and comparable model sizes. This is interesting considering that the performance of **APTP** and **APT** are almost identical whereas the two approaches clearly outperformed **FLOW-CBP**.

As **APT** does not offer any competitive advantage over **APTP** (either in terms of model size, continuous relaxation value, or computational behavior), we do not test the approach in further experiments.

6.2 Variable reduction techniques

6.2.1 P|*cont*|C_{max}

We report in Table 7 the results obtained by a baseline version of **BM** without any variable reduction techniques, and compare it with a version that uses normal patterns and with a version that uses MIM patterns. For normal patterns and MIM patterns, we also report between round brackets the average variable reduction ratio and the average constraint reduction ratio compared to the baseline version of **BM**. Such ratios are computed as follows: for CGCUT1, 125 variables were needed for the baseline version of **BM** whereas 124 variables were needed for the version using normal patterns, resulting in a variable reduction ratio equals to $\frac{1}{125} = 0.8\%$; considering that these ratios were 25% for CGCUT2 and 35% for CGCUT3, the average reduction ratio for dataset CGCUT was $\frac{0.8\%+25\%+35\%}{3} = 20.3\%$.

Table 7: Results of **BM** varying the chosen variable reduction technique

		BM														
Dataset	#inst	**Baseline**					**Normal patterns**					**MIM patterns**				
		#opt	T(s)	T_{opt}(s)	#var	#const	#opt	T(s)	T_{opt}(s)	#var (% red.)	#const (% red)	#opt	T(s)	T_{opt}(s)	#var (% red)	#const (% red)
CGCUT	3	2	1201	1	1357	78	2	1201	2	939(-20.3%)	69(-9.5%)	3	11	11	580(-38%)	64(-13.9%)
HT	9	8	407	8	842	62	8	406	7	823(-2.8%)	62(-1.4%)	8	404	4	804(-5.7%)	62(-1.4%)
BENG	10	10	8	8	2823	123	10	8	8	2823(-0%)	123(-0%)	10	8	8	2823(-0%)	123(-0%)
NGCUT	12	12	0	0	153	28	12	1	1	135(-14.9%)	26(-9.8%)	12	1	1	128(-17.9%)	25(-10.4%)
BKW	13	7	2040	700	160 852	459	7	1989	605	160 655(-4%)	457(-1.7%)	6	2233	635	160 457(-8%)	457(-1.8%)
GCUT	13	11	581	22	10 358	541	12	280	3	4322(-72.9%)	224(-64.7%)	12	279	2	2446(-79.5%)	182(-66.1%)
CLASS1	50	50	0	0	178	36	50	0	0	143(-32.2%)	34(-12.4%)	50	0	0	124(-40.9%)	34(-12.5%)
CLASS2	50	50	14	14	1530	90	50	13	13	1529(-0.1%)	90(-0%)	50	13	13	1528(-0.2%)	90(-0%)
CLASS3	50	47	348	141	669	58	47	336	128	487(-51.1%)	47(-32%)	47	339	130	408(-62.4%)	47(-32.3%)
CLASS5	50	48	147	3	1155	78	48	146	2	624(-53.9%)	48(-40.1%)	48	147	3	432(-63.7%)	46(-41.1%)
CLASS7	50	50	0	0	255	49	50	0	0	20(-70.6%)	9(-63.1%)	50	0	0	13(-72.3%)	9(-63.1%)
CLASS9	50	50	0	0	60	24	50	0	0	4(-41.8%)	3(-39.9%)	50	0	0	4(-41.9%)	3(-39.9%)
CLASS10	50	24	1907	73	3332	142	24	1909	78	2937(-23.5%)	128(-14.9%)	24	1916	92	2660(-34.4%)	125(-17.6%)
Total	410	369	396	39	6406	96	370	383	35	6029(-36.4%)	71(-27.2%)	370	383	35	5890(-42.2%)	69(-27.8%)

Overall, we observe that the variable reduction techniques are empirically effective for **BM** as using normal patterns reduces the number of variables by 36% on average whereas using MIM patterns reduces the number of variables by 42% on average. We notice that the variable reduction is more pronounced for some datasets (e.g., CGCUT or CLASS3) than others (e.g., HT or BENG). As far as the number of instances solved to optimality is concerned, it appears that variable reduction techniques only have a minor impact as using normal or MIM patterns only allows **BM** to solve one instance more to optimality within the time limit. We report in Table 8 the outcome of similar experiments for **FLOW-PCC** and **CP-PCC**.

We notice that the variable reduction techniques have a more significant impact on **FLOW-PCC** as the approach could solve 5 more instances to optimality within the time limit when using the MIM patterns than it did in the baseline version. As far as **CP-PCC** is concerned, it appears that using one of

Table 8: Results of **FLOW-PCC** and **CP-PCC** varying the chosen variable reduction technique

Dataset	#inst	**FLOW-PCC**									**CP-PCC**								
		Baseline			**Normal patterns**			**MIM Patterns**			**Baseline**			**Normal patterns**			**MIM patterns**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1250	75	2	1303	155	2	1211	17	1	2400	0	1	2400	0	1	2400	0
HT	9	8	409	11	8	403	4	8	421	24	8	431	35	8	409	10	8	413	15
BENG	10	8	778	73	8	778	72	8	778	72	10	1	1	10	1	1	10	1	1
NGCUT	12	12	55	55	12	8	8	12	7	7	12	0	0	12	0	0	12	0	0
BKW	13	5	2635	1084	3	2773	10	4	2587	304	4	2741	809	4	2735	789	4	2793	978
GCUT	13	9	1121	18	10	831	1	12	492	233	9	1108	0	9	1108	0	9	1108	0
CLASS1	50	50	0	0	50	0	0	50	0	0	44	436	4	43	507	3	43	508	5
CLASS2	50	50	115	115	49	178	108	50	100	100	50	2	2	50	2	2	50	2	2
CLASS3	50	43	505	1	43	505	1	43	505	1	31	1386	28	31	1369	1	30	1440	0
CLASS5	50	47	302	92	47	240	26	48	201	59	39	797	6	38	897	44	37	936	0
CLASS7	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS9	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS10	50	21	2174	204	23	2033	194	23	2038	203	19	2249	46	18	2306	6	19	2257	67
Total	410	355	535	60	355	512	34	360	482	49	327	743	18	324	768	16	323	778	17

the two proposed variable reduction techniques worsens the performance of the approach. This could be due to the fact that preventing the first slice of an item to be packed in a given column adds a constraint in **CP-PCC** (`IloForbidStart`) whereas doing so in **BM** or in **FLOW-PCC** removes a variable.

6.2.2 1CBP

We compare in Table 9 the results obtained by two versions of **APTP**, **FLOW-CBP**, and **CP-CBP**: one without any variable reduction technique and one using normal patterns.

Table 9: Results of **APTP**, **FLOW-CBP**, **CP-CBP** varying the chosen variable reduction technique

Dataset	#inst	**APTP**						**FLOW-CBP**						**CP-CBP**					
		Baseline			**Normal patterns**			**Baseline**			**Normal patterns**			**Baseline**			**Normal patterns**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1203	5	2	1206	9	2	1210	14	2	1205	6	1	2400	0	1	2400	0
HT	9	9	9	9	9	26	26	9	31	31	9	21	21	8	415	17	8	419	21
BENG	10	8	768	60	8	768	59	3	2577	190	3	2577	191	10	36	36	10	39	39
NGCUT	12	12	4	4	12	11	11	12	10	10	12	5	5	12	1	1	12	1	1
BKW	13	7	2207	1013	6	2241	645	4	2500	27	5	2355	343	5	2229	36	6	2166	493
GCUT	13	4	2509	53	5	2220	2	3	2770	1	5	2260	3	7	1705	81	7	1703	77
CLASS1	50	41	725	94	41	724	92	35	1185	151	36	1175	232	50	105	105	50	115	115
CLASS2	50	42	714	164	42	714	164	26	1813	164	26	1813	164	49	122	51	49	122	51
CLASS3	50	20	2164	9	20	2166	12	18	2308	12	18	2313	17	36	1022	19	36	1026	25
CLASS5	50	21	2114	62	21	2128	89	20	2161	3	21	2111	41	34	1158	7	34	1156	5
CLASS7	50	47	237	23	48	174	31	47	275	63	47	217	1	50	1	1	50	1	1
CLASS9	50	50	6	6	50	0	0	50	3	3	50	0	0	50	0	0	50	0	0
CLASS10	50	6	3171	23	7	3101	36	6	3232	536	6	3190	124	14	2594	8	14	2594	9
Total	410	269	1291	80	271	1268	71	235	1579	73	240	1538	71	326	762	31	327	762	42

Just like for the P|*cont*|C_{max}, using normal patterns seems slightly beneficial for **APTP** and **FLOW-CBP** whereas the impact is less visible for **CP-CBP**. We provide in Appendix D the size of models **APTP** and **FLOW-CBP** and outline that the variable reduction obtained from using normal patterns is less significant (-13.1% for **APTP** and -14.9% for **FLOW-CBP**) than it was for the P|*cont*|C_{max}.

These experiments showed that, even though variable reductions techniques could often bring a significant decrease in terms of model size, such a decrease did not necessarily translate into an increase in terms of performance. Based on these results, we decided to keep using normal patterns in the ILP models and to not use any variable reduction techniques in the CP models.

6.3 Exploiting item multiplicity

6.3.1 P|*cont*|C_{max}

We compare in Table 10 the outcome of extensive experiments aimed at evaluating the impact of exploiting item multiplicity. To do so, we ran three versions of **BM** and **FLOW-PCC**: one version that does not exploit item multiplicity at all, one that gathers identical items and uses a binary expansion, and one that gathers identical items and uses integer variables.

Table 10: Results of **BM** and **FLOW-PCC** with binary variables, binary expansion, and integer variables

		BM+NOR									**FLOW-PCC+NOR**								
Dataset	#inst	**Binary var**			**Binary expansion**			**Integer var**			**Binary var**			**Binary expansion**			**Integer var**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1201	2	3	73	73	3	153	153	2	1303	155	2	1204	7	2	1229	43
HT	9	8	406	7	9	27	27	9	8	8	8	403	4	9	390	390	9	324	324
BENG	10	10	8	8	9	365	6	10	125	125	8	778	72	7	1185	150	10	591	591
NGCUT	12	12	1	1	12	1	1	12	1	1	12	8	8	12	12	12	12	12	12
BKW	13	7	1989	605	9	1268	231	10	1001	221	3	2773	10	8	1531	237	8	1471	141
GCUT	13	12	280	3	12	280	3	12	280	3	10	831	1	10	831	1	10	831	1
CLASS1	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS2	50	50	13	13	50	6	6	50	23	23	49	178	108	48	208	67	50	124	124
CLASS3	50	47	336	128	47	297	86	47	284	72	43	505	1	44	487	62	43	505	1
CLASS5	50	48	146	2	48	146	2	48	145	1	47	240	26	46	290	2	46	291	3
CLASS7	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS9	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS10	50	24	1909	78	24	1940	141	24	1915	90	23	2033	194	22	2082	150	22	2134	269
Total	410	370	383	35	373	350	28	375	334	29	355	512	34	358	496	45	362	476	62

We notice that exploiting item multiplicity seems effective for both approaches. Using binary expansion allows each method to solve 3 instances more to optimality whereas using integer variables allows **BM** and **FLOW-PCC** to solve respectively 5 and 7 instances more to optimality within the time limit. As one could expect, exploiting item multiplicity is more effective when solving a dataset with a high number of identical items (such as BKW) than it is when solving a dataset with a low number of identical items (such as CLASS10). Among the numerous methods we have evaluated so far, **BM** with normal patterns and integer variables is the one that solves the highest number of instances to optimality (375 out of 410).

6.3.2 1CBP

The outcome of similar experiments performed on **APTP** and **FLOW-CBP** are reported in Table 11.

Table 11: Results of **APTP**, **FLOW-CBP** with binary variables, binary expansion, and integer variables

		APTP+NOR									**FLOW-CBP+NOR**								
Dataset	#inst	**Binary var**			**Binary expansion**			**Integer var**			**Binary var**			**Binary expansion**			**Integer var**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1206	9	2	1201	2	2	1202	3	2	1205	6	2	1202	3	2	1202	3
HT	9	9	26	26	9	30	30	9	11	11	9	21	21	9	18	18	9	10	10
BENG	10	8	768	59	6	1552	187	6	1601	269	3	2577	191	2	2880	1	2	2880	3
NGCUT	12	12	11	11	12	1	1	12	2	2	12	5	5	12	3	3	12	3	3
BKW	13	6	2241	645	7	1799	249	8	1606	355	5	2355	343	6	2087	315	7	2005	633
GCUT	13	5	2220	2	5	2222	2	5	2220	2	5	2260	3	5	2217	3	5	2217	3
CLASS1	50	41	724	92	41	698	61	40	755	44	36	1175	232	38	989	165	36	1109	140
CLASS2	50	42	714	164	37	1142	278	40	910	238	26	1813	164	27	1737	150	29	1655	246
CLASS3	50	20	2166	12	21	2116	63	20	2173	29	18	2313	17	18	2309	10	18	2308	8
CLASS5	50	21	2128	89	21	2128	89	21	2128	89	21	2111	41	21	2107	41	21	2107	41
CLASS7	50	48	174	31	48	174	32	48	174	31	47	217	1	47	217	1	47	217	1
CLASS9	50	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0
CLASS10	50	7	3101	36	7	3101	36	7	3101	36	6	3190	124	6	3183	124	6	3183	124
Total	410	271	1268	71	266	1316	79	268	1296	75	240	1538	71	243	1502	59	244	1503	76

Interestingly, it appears that **APTP** does not seem to benefit from exploiting item multiplicity when it comes to the total number of instances solved to optimality as the approach solves 5 instances less when using binary expansion and 3 instances less when using integer variables than it does when using binary vari-

ables. Note, however, that the version of **APTP** that uses integer variables performs well on dataset BKW, which has a high number of identical items. As far as **FLOW-CBP** is concerned, it seems like exploiting item multiplicity has a positive effect since the method is able to solve 3 instances more when using binary expansion and 4 instances more when using integer variables than it does when using binary variables.

These experiments showed that exploiting item multiplicity usually improves the performance of the ILP-based approaches, even if this is not always the case. Based on these results, we decided to keep the version of the ILP models that uses integer variables even if it could also make sense to design a set of empirical rules in which one decides whether or not item multiplicity should be exploited based on the average number of identical items contained in the dataset one wishes to solve.

6.4 Symmetry-breaking constraints

6.4.1 P|*cont*|C_{max}

We report in Table 12 the outcome of the experiments aimed at evaluating the impact of symmetry-breaking constraints. To do so, we start by running three versions of **BM**, **FLOW-PCC**, and **CP-PCC**: one version that does not use any symmetry-breaking constraints at all (referred to as "S_B"), one that uses (59) together with the desegregated gapless packing constraints (60) (referred to as "S^1_G"), and one that uses (59) together with the aggregated gapless packing constraints (61) (referred to as "S^2_G").

Table 12: Results of **BM**, **FLOW-PCC**, **CP-PCC** with/without gapless packing constraints

		BM+NOR+ INT						**FLOW-PCC+NOR+ INT**						**CP-PCC**					
Dataset	#inst	S_B		S^1_G		S^2_G		S_B		S^1_G		S^2_G		S_B		S^1_G		S^2_G	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	3	153	2	1201	2	1201	2	1229	2	1203	2	1206	1	2400	1	2400	1	2400
HT	9	9	8	8	444	9	406	9	324	8	412	9	99	8	431	8	456	8	411
BENG	10	10	125	10	21	10	16	10	591	8	1333	8	1056	10	1	10	2	10	1
NGCUT	12	12	1	12	1	12	0	12	12	12	3	12	3	12	0	12	1	12	0
BKW	13	10	1001	9	1202	9	1156	8	1471	8	1766	7	1907	4	2741	2	2931	1	3046
GCUT	13	12	280	12	562	12	360	10	831	10	831	10	831	9	1108	8	1108	8	1108
CLASS1	50	50	0	50	0	50	0	50	0	50	0	50	0	44	436	43	507	42	577
CLASS2	50	50	23	50	6	49	82	50	124	49	208	50	156	50	2	50	1	50	5
CLASS3	50	47	284	46	376	46	332	43	505	43	505	44	493	31	1386	30	1379	29	1440
CLASS5	50	48	145	48	157	48	153	46	291	48	228	48	199	39	797	35	793	33	937
CLASS7	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	47	0	47	0
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	34	0	34	0
CLASS10	50	24	1915	23	2105	23	2000	22	2134	22	2062	22	2047	19	2249	18	2304	17	2385
Total	410	375	334	370	397	370	379	362	476	360	499	362	477	327	743	298	764	292	810

It appears that adding gapless packing constraints slightly deteriorates the performance of **BM** and **FLOW-PCC** and significantly deteriorates the performance of **CP-PCC**. As a result, such constraints are not considered anymore in subsequent P|*cont*|C_{max} experiments.

We continue by running two more versions of **BM**, **FLOW-PCC** and **CP-PCC**. For **BM** and **FLOW-PCC**, the first version uses mirror symmetry-breaking constraints (64) (which was applied to the item type with the smallest width) and is referred to as "S^1_M" in the table whereas the second version uses mirror symmetry-breaking constraints (63) (which was applied to the item type with the smallest width among the item types with demand 1) and is referred to as "S^2_M" in the table. For **CP-PCC**, the first version also uses mirror symmetry-breaking constraints (63) whereas the second version uses symmetry-breaking constraints (65) (related to the permutation of identical items) and is referred to as "S_I" in the table. Note that (i) one cannot apply constraints (64) to **CP-PCC** since the model does not exploit item multiplicity and (ii) one cannot apply constraints (65) to the tested versions of **BM** and **FLOW-PCC** as those already exploit item multiplicity through the use of integer variables. Just like gapless packing constraints, it appears that mirror symmetry-breaking constraints slightly deteriorate the

Table 13: Results of **BM**, **FLOW-PCC**, **CP-PCC** with/without other symmetry-breaking constraints

Dataset	#inst	BM+NOR+ INT						FLOW-PCC + NOR + INT						CP-PCC					
		$S_{\mathbf{B}}$		S^1_{M}		S^2_{M}		$S_{\mathbf{B}}$		S^1_{M}		S^2_{M}		$S_{\mathbf{B}}$		S^2_{M}		$S_{\mathbf{I}}$	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	3	153	2	1200	3	712	2	1229	2	1202	2	1205	1	2400	1	2400	1	2400
HT	9	9	8	8	412	9	39	9	324	8	408	8	408	8	431	8	406	8	447
BENG	10	10	125	10	10	10	83	10	591	8	823	7	1366	10	1	10	1	10	1
NGCUT	12	12	1	12	0	12	0	12	12	12	2	12	3	12	0	12	0	12	0
BKW	13	10	1001	8	1469	8	1396	8	1471	8	1565	10	1523	4	2741	4	2890	3	2965
GCUT	13	12	280	12	280	12	280	10	831	11	742	11	733	9	1108	10	841	9	1108
CLASS1	50	50	0	50	0	50	0	50	0	50	0	50	0	44	436	46	309	47	224
CLASS2	50	50	23	50	31	50	5	50	124	50	44	48	170	50	2	50	1	50	1
CLASS3	50	47	284	45	374	47	298	43	505	43	505	43	508	31	1386	31	1369	31	1393
CLASS5	50	48	145	48	147	48	146	46	291	47	225	47	257	39	797	38	865	38	868
CLASS7	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS10	50	24	1915	24	1904	24	1930	22	2134	22	2097	22	2146	19	2249	18	2304	19	2249
Total	410	375	334	369	373	373	351	362	476	361	461	360	499	327	743	328	736	328	734

performance of **BM** and **FLOW-PCC**. However, those seem useful for **CP-PCC** as the approach was able to solve one additional instance to optimality within the time limit. Forbidding the permutation of identical items also allows **CP-PCC** to solve one supplementary instance. We also tested a version of **CP-PCC** that forbids the permutation of identical items in addition of using mirror symmetry-breaking constraints and we can report that such a version solved 330 instances to optimality out of 410. Note, however, that the mentioned improvements do not appear in all datasets: in CLASS10 for example, the version of **CP-PCC** that works best is the one without any symmetry-breaking constraints.

6.4.2 1CBP

The outcome of similar experiments conducted with the 1CBP models are reported in Tables 14 and 15.

Table 14: Results of **APTP**, **FLOW-CBP**, **CP-CBP** with/without symmetry-breaking constraints

Dataset	#inst	APTP + NOR+ INT						FLOW-CBP+NOR+ INT						CP-CBP					
		$S_{\mathbf{B}}$		$S^1_{\mathbf{G}}$		$S^2_{\mathbf{G}}$		$S_{\mathbf{B}}$		$S^1_{\mathbf{G}}$		$S^2_{\mathbf{G}}$		$S_{\mathbf{B}}$		$S^1_{\mathbf{G}}$		$S^2_{\mathbf{G}}$	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	2	1202	2	1203	2	1201	2	1202	2	1202	2	1201	1	2400	1	2400	1	2400
HT	9	9	11	8	601	9	31	9	10	9	59	9	117	8	415	9	184	9	148
BENG	10	6	1601	2	2882	4	2218	2	2880	2	2881	2	2881	10	36	10	148	10	111
NGCUT	12	12	2	12	4	12	2	12	3	12	5	12	3	12	1	12	1	12	2
BKW	13	8	1606	7	1832	5	2254	7	2005	7	2158	8	1870	5	2229	5	2329	5	2441
GCUT	13	5	2220	5	2220	5	2222	5	2217	5	2217	5	2216	7	1705	6	1939	6	1942
CLASS1	50	40	755	39	908	43	667	36	1109	34	1232	38	1041	50	105	50	155	50	123
CLASS2	50	40	910	26	1856	35	1280	29	1655	28	1877	24	1916	49	122	47	270	48	206
CLASS3	50	20	2173	20	2166	22	2030	18	2308	19	2258	19	2262	36	1022	36	1039	35	1168
CLASS5	50	21	2128	23	2028	22	2030	21	2107	22	2029	22	2024	34	1158	34	1162	33	1262
CLASS7	50	48	174	48	151	49	137	47	217	48	204	48	211	50	1	50	1	50	39
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS10	50	7	3101	7	3119	7	3125	6	3183	6	3191	7	3177	14	2594	15	2617	13	2686
Total	410	268	1296	249	1468	265	1336	244	1503	244	1535	246	1508	326	762	325	800	322	831

Overall, we observe that using symmetry-breaking constraints does not have a clear positive effect on **APTP**, even though we notice that one additional instance could be solved to optimality when using mirror symmetry-breaking constraints (64). As far as **FLOW-CBP** is concerned, it appears that using (59) together with the aggregated gapless packing constraints (61) has a positive effect, and so does using mirror symmetry-breaking constraints (64). We also tested a version of **FLOW-CBP** that uses (59), (61), and (64) at the same time, and we can report that such a version solved 249 instances to optimality out of 410. When it comes to **CP-CBP**, it seems like forbidding the permutation of identical items is the only symmetry-breaking constraint that brings an empirical improvement.

Table 15: Results of **APTP**, **FLOW-CBP**, **CP-CBP** with/without other symmetry-breaking constraints

Dataset	#inst	APTP + NOR+ INT						FLOW-CBP+NOR+ INT						CP-CBP					
		$S_{\mathbf{B}}$		$S^1_{\mathbf{M}}$		$S^2_{\mathbf{M}}$		$S_{\mathbf{B}}$		$S^1_{\mathbf{M}}$		$S^2_{\mathbf{M}}$		$S_{\mathbf{B}}$		$S^2_{\mathbf{M}}$		$S_{\mathbf{I}}$	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	2	1206	2	1202	2	1202	2	1202	2	1202	2	1202	1	2400	1	2400	2	1286
HT	9	9	26	9	79	9	8	9	10	9	11	9	99	8	415	8	410	8	418
BENG	10	8	768	6	1765	6	1758	2	2880	3	2567	3	2633	10	36	10	36	10	142
NGCUT	12	12	11	12	2	12	2	12	3	12	2	12	2	12	1	12	0	12	0
BKW	13	6	2241	8	1889	8	1620	7	2005	8	1670	7	1799	5	2229	5	2363	5	2360
GCUT	13	5	2220	5	2220	5	2219	5	2217	5	2217	5	2216	7	1705	7	1694	7	1705
CLASS1	50	41	724	44	712	40	879	36	1109	39	966	38	1041	50	105	50	63	50	30
CLASS2	50	42	714	38	1193	38	1092	29	1655	27	1783	27	1830	49	122	47	240	48	192
CLASS3	50	20	2166	21	2142	21	2096	18	2308	18	2312	19	2246	36	1022	36	1042	36	1034
CLASS5	50	21	2128	22	2089	22	2059	21	2107	21	2112	21	2116	34	1158	34	1160	34	1157
CLASS7	50	48	174	48	154	47	217	47	217	47	217	47	217	50	1	50	0	50	1
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS10	50	7	3101	7	3137	7	3104	6	3183	6	3177	6	3213	14	2594	15	2542	15	2543
Total	410	271	1268	272	1333	267	1326	244	1503	247	1484	246	1503	326	762	325	772	327	755

These experiments showed that, despite being useful in theory, the reviewed symmetry-breaking constraints do not necessarily improve (and can even deteriorate) the performance of the tested ILP-based approaches. As far as CP-based approaches are concerned, it seems like preventing symmetric solutions in which identical items are permutated is useful for both the 1CBP and the P|*cont*|$\mathrm{C_{max}}$. Based on these results, we decided to not add any symmetry-breaking constraints to the ILP models and to only use the symmetry-breaking constraints related to permutations of identical items for the CP models.

6.5 Destructive bounds and reduced-cost variable fixing

We report in Tables 16 and 17 the outcome of the experiments aimed at evaluating the impact of destructive bounds (referred to as "DB" in the table) and reduced-cost variable fixing (referred to as "RCVF" in the table). For the P|*cont*|$\mathrm{C_{max}}$ (resp., for the 1CBP), an approach using destructive bounds tries to find a solution for a problem instance in which the bin capacity (resp., the number of bins) is fixed to a given lower bound LB. If the approach finds a feasible solution, then it is proven to be optimal as the solution has an objective value that is equal to a lower bound. If the approach proves that there isn't any solution with objective value LB, then LB+1 becomes a valid lower bound and the approach is called again. Destructive bounds are often used in the SPP literature [27, 37]. An ILP-based approach that uses reduced-cost variable fixing works similarly. The only difference is that, at each iteration, every variable that cannot appear (i.e., that cannot take value 1 or above) in a solution with objective value LB is deactivated. To do so, such an approach starts by solving the LP-relaxation of the model (say with objective value $\hat{z}$) and computes the reduced cost of every variable in the model. One can show [34] that any LP solution in which a variable v with reduced cost $\hat{s}_v - \epsilon \geq \mathrm{LB} - \hat{z}$ takes value 1 or above must have objective value strictly above LB (ϵ is a very small number used to avoid precision errors). Therefore, one can deactivate variable v for a given LB (i.e., set the variable to 0 or not creating it at all) if $\hat{s}_v - \epsilon \geq \mathrm{LB} - \hat{z}$. Reduced-cost variable fixing was shown to be sometimes useful when solving ILP models with a very tight LP-relaxation such as the cycle formulation for the kidney exchange problem [34] or the arcflow formulation for the bin packing problem [35].

From the implementation perspective, we initialized LB with the LP-relaxation value of model **BM** rounded up for **BM** and **CP-PCC**, with the LP-relaxation value of model **FLOW-PCC** rounded up for **FLOW-PCC**, and with $L_2^{CBP'}$ for **APTP**, **FLOW-CBP**, and **CP-CBP**. We also outline that, if one does not need to minimize the number of bins, several simplifications occur in **APTP** and **FLOW-CBP**. In fact, **APTP** with destructive bounds is identical to **BM** with destructive bounds and **FLOW-CBP** with destructive bounds is identical to **FLOW-PCC** with destructive bounds.

For the P|*cont*|$\mathrm{C_{max}}$, it appears that using destructive bounds is useful for both **FLOW-PCC** and **CP-**

Table 16: Results of **BM**, **FLOW-PCC**, **CP-PCC** with/without DB and RCVF

Dataset	#inst	BM+NOR+ INT						FLOW-PCC + NOR + INT						CP-PCC+S_I			
		B		DB		DB+RCVF		B		DB		DB+RCVF		B		DB	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	3	153	2	1200	2	1200	2	1229	2	1200	2	1201	1	2400	1	2400
HT	9	9	8	9	113	9	343	9	324	9	31	9	284	8	447	9	23
BENG	10	10	125	10	52	10	52	10	591	10	23	10	23	10	1	10	0
NGCUT	12	12	1	12	0	12	0	12	12	12	1	12	1	12	0	12	0
BKW	13	10	1001	7	1670	8	1423	8	1471	10	968	10	958	3	2965	5	2233
GCUT	13	12	280	12	290	12	298	10	831	10	832	10	831	9	1108	9	1111
CLASS1	50	50	0	50	0	50	0	50	0	50	0	50	0	47	224	50	0
CLASS2	50	50	23	50	4	50	4	50	124	50	44	50	45	50	1	50	1
CLASS3	50	47	284	45	380	45	388	43	505	45	392	46	376	31	1393	42	607
CLASS5	50	48	145	48	147	48	147	46	291	48	186	48	189	38	868	43	576
CLASS7	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS10	50	24	1915	24	1884	24	1901	22	2134	23	1973	23	1984	19	2249	24	1937
Total	410	375	334	369	369	370	370	362	476	369	384	370	389	328	734	355	505

Table 17: Results of **APTP**, **FLOW-CBP**, **CP-CBP** with/without DB and RCVF

Dataset	#inst	APTP+NOR+INT						FLOW-CBP+NOR+INT						CP-CBP+S_I			
		B		DB		DB+RCVF		B		DB		DB+RCVF		B		DB	
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)
CGCUT	3	2	1202	2	1201	2	1201	2	1202	2	1201	2	1201	2	1286	2	1272
HT	9	9	11	9	11	9	11	9	10	9	8	9	3	8	418	9	5
BENG	10	6	1601	5	1991	5	1991	2	2880	4	2270	4	2270	10	142	9	362
NGCUT	12	12	2	12	1	12	1	12	3	12	1	12	1	12	0	12	0
BKW	13	8	1606	9	1396	9	1396	7	2005	9	1339	10	1059	5	2360	4	2516
GCUT	13	5	2220	5	2430	5	2408	5	2217	5	2260	5	2260	7	1705	7	1929
CLASS1	50	40	755	39	864	39	864	36	1109	35	1226	35	1217	50	30	41	720
CLASS2	50	40	910	39	927	39	926	29	1655	26	1863	26	1863	48	192	47	339
CLASS3	50	20	2173	22	2159	22	2159	18	2308	18	2315	18	2315	36	1034	31	1376
CLASS5	50	21	2128	21	2158	21	2158	21	2107	21	2116	21	2116	34	1157	34	1168
CLASS7	50	48	174	47	217	47	217	47	217	47	217	47	217	50	1	50	1
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	50	0	50	0
CLASS10	50	7	3101	7	3227	7	3227	6	3183	5	3255	5	3255	15	2543	13	2668
Total	410	268	1296	267	1344	267	1343	244	1503	243	1519	244	1509	327	755	309	924

PCC, but not for **BM**. Interestingly, we observe that the performance of **FLOW-PCC** with destructive bounds is almost comparable to the performance of **BM** with destructive bounds. For the 1CBP, using destructive bounds does not seem to bring any empirical improvement for **APTP** and **FLOW-CBP**, and significantly deteriorates the performance of **CB-CBP**. This may come as a surprise considering the positive results observed for **CP-PCC**. A possible explanation could be that proving infeasibility for a CP approach is more difficult for an instance with a large makespan and a resource available in small quantity compared to an instance with a small makespan and a resource available in large quantity (indeed, most of the tested instances had W significantly smaller than the optimal solution value). When it comes to reduced-cost variable fixing, it seems like using the technique brings a marginal improvement compared to using destructive bounds alone for both the P|*cont*|C_{max} and the 1CBP.

6.6 Approach combination and randomness impact

In total, we tested 35 approaches for the P|*cont*|C_{max} and 32 approaches for the 1CBP (the 3 variations using MIM patterns were not suitable for the 1CBP). For the P|*cont*|C_{max}, the best approach was **BM** with normal patterns and integer variables with a total of 375 instances solved. For the 1CBP, the best approach was **CP-CBP** with symmetry-breaking constraints forbidding the permutation of identical items with a total of 327 instances solved. However, we observed that, if we consider the results of all

tested methods, the total number of P|*cont*|C_{max} instances solved to optimality was 383 and the total number of 1CBP instances solved to optimality was 343. This means that the 2 mentioned approaches did not clearly dominate the others on all datasets. Therefore, an interesting research question would be to identify a combination of approaches solving the maximum number of instances. To do so, we solved the following *Maximum Coverage Problem* (MCP). For each of the tested approaches, we created sub-variations simulating the results that would have been obtained by the approach if a certain time limit of x seconds was used, where x varied between 1 and 3600. For a given approach a and a given instance i, we say that i is covered by sub-variation v_x of a if a solves i in less than x seconds. We also define the budget of a sub-variation v_x to be the amount of time spent running v_x (i.e., x). The objective is to find a subset of sub-variations covering the maximum number of instances with a budget of 3600 seconds.

For the P|*cont*|C_{max}, 382 instances could be covered in one hour by using three sub-variations: a variant of **BM**, a variant of **FLOW-PCC**, and a variant of **CP-PCC**. If we restrict the number of sub-variations to two, then 381 instances could be covered by using **BM** with normal patterns and integer variables for 2000 seconds and **CP-PCC** with symmetry-breaking constraints forbidding the permutation of identical items for 1000 seconds. For the 1CBP, 338 instances could be covered in one hour by using six sub-variations: four variants of **CP-CBP** and two variants of **FLOW-CBP**. If we restrict the number of sub-variations to two, then 334 instances could be covered by using **CP-CBP** with mirror symmetry-breaking constraints (63) for 2500 seconds and **FLOW-CBP** with normal patterns, integer variables, and reduced-cost variable fixing for 1000 seconds. We point out that: (i) the 3600s budget was not always entirely used, (ii) we ran the MCP model for up to 3600 sub-variations per approach, but the aforementioned results were not improved, (iii) the MCP model could always be solved to optimality in less than 60 seconds, and (iv) we provide the MCP model used in our experiments in Appendix E for the interested reader.

A last question one may wonder is whether the results reported in our experiments are strongly impacted by solver randomness. Indeed, it is common knowledge that solver randomness may influence the performance of an approach, therefore, it is possible that one of the small improvements observed when using a specific feature was more due to solver randomness than to the impact of said feature. To do so, we ran each component of the best pair of approaches for both the P|*cont*|C_{max} and the 1CBP with 5 different random seeds, which, according to the solver's documentation, "acts as a small perturbation to the solver, and typically leads to different solution paths" [47]. These results, which also include experiments on CLASS4, CLASS6, and CLASS8, are reported in Tables 18 and 19 for the P|*cont*|C_{max} and in Appendix F for the 1CBP. In these tables, the column labeled "# solved by x seeds" indicates the number of instances solved by exactly x out of the 5 tested random seeds. If solver randomness was non-existent, then only the columns "# solved by 0 seeds" and "# solved by 5 seeds" would contain non-zero values.

Table 18: Results of **BM** with normal patterns and integer variables for P|*cont*|C_{max} instances

Dataset	#inst	**Seed-1**		**Seed-2**		**Seed-3**		**Seed-4**		**Seed-5**		**#solved by x seeds**					
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	5	4	3	2	1	0
CGCUT	3	3	766	2	1200	3	1115	3	90	3	196	2	1	0	0	0	0
HT	9	9	186	8	407	9	70	9	220	8	425	8	0	1	0	0	0
BENG	10	10	156	9	374	9	368	10	48	10	9	9	0	1	0	0	0
NGCUT	12	12	0	12	0	12	1	12	0	12	1	12	0	0	0	0	0
BKW	13	10	1054	8	1389	9	1311	9	1228	10	979	8	0	1	1	1	2
GCUT	13	12	280	12	279	12	279	12	279	12	279	12	0	0	0	0	1
CLASS1	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS2	50	50	4	50	11	50	20	50	6	50	4	50	0	0	0	0	0
CLASS3	50	48	247	45	396	47	310	47	291	46	343	45	1	1	0	1	2
CLASS4	50	0	3600	1	3539	0	3600	0	3600	0	3600	0	0	0	0	1	49
CLASS5	50	48	147	48	146	48	146	48	146	48	146	48	0	0	0	0	2
CLASS6	50	0	3600	0	3600	0	3600	0	3600	0	3600	0	0	0	0	0	50
CLASS7	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS8	50	4	3324	5	3260	4	3316	6	3233	5	3266	4	0	1	0	1	44
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS10	50	24	1901	24	1904	24	1954	24	1915	24	1911	24	0	0	0	0	26
Total	560	380	1186	374	1206	377	1206	380	1182	378	1186	372	2	5	1	4	176

Table 19: Results of **CP-PCC** with symmetry-breaking constraints forbidding the permutation of identical items and DB for P|*cont*|C_{max} instances

Dataset	#inst	Seed-1		Seed-2		Seed-3		Seed-4		Seed-5		#solved by x seeds					
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	5	4	3	2	1	0
CGCUT	3	1	2400	1	2400	1	2400	1	2400	1	2400	1	0	0	0	0	2
HT	9	9	91	9	7	9	32	9	14	9	19	9	0	0	0	0	0
BENG	10	10	0	10	0	10	0	10	0	10	0	10	0	0	0	0	0
NGCUT	12	12	0	12	0	12	0	12	0	12	0	12	0	0	0	0	0
BKW	13	5	2364	5	2220	5	2222	5	2259	5	2241	5	0	0	0	0	8
GCUT	13	9	1112	9	1111	9	1111	9	1112	9	1112	9	0	0	0	0	4
CLASS1	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS2	50	50	1	50	1	50	0	50	3	50	0	50	0	0	0	0	0
CLASS3	50	41	708	42	669	42	675	41	654	41	665	41	0	0	0	2	7
CLASS4	50	20	2228	19	2273	21	2169	20	2298	20	2257	18	1	1	1	1	28
CLASS5	50	43	516	42	592	43	538	42	589	43	519	42	0	1	0	0	7
CLASS6	50	10	2910	10	2911	10	2908	10	2911	10	2912	10	0	0	0	0	40
CLASS7	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS8	50	12	2761	12	2763	12	2762	12	2763	12	2762	12	0	0	0	0	38
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS10	50	24	1955	24	1963	24	1960	24	1956	24	1958	24	0	0	0	0	26
Total	560	396	1084	395	1088	398	1074	395	1089	396	1080	393	1	2	1	3	160

As anticipated, both the number of instances solved to optimality and the average running time are influenced by solver randomness for the P|*cont*|C_{max}. However, the extent of such an influence is fairly limited as the tested version of **BM** solved between 374 and 380 instances in total, among which 372 instances were solved by all 5 tested random seeds whereas the tested version of **CP-PCC** solved between 395 and 398 instances in total, among which 393 instances were solved by all 5 tested random seeds. This indicates that randomness alone could explain a small increase/decrease of the number of optimal solutions found by a given approach, which is why we only kept in our experiments the model enhancements that brought an empirical improvement that was either consistent (i.e., that could be observed on most of the tested approaches) or significant (i.e., that was well above the margin of error). The difficulty of datasets CLASS4, CLASS6, and CLASS8 was, once more, empirically demonstrated as the tested version of **BM** mostly failed to find any optimal solution within the time limit. Interestingly, the tested version of **CP-PCC** showed a surprising good performance on these three datasets as it could solve all instances with 20 items.

As far as the 1CBP is concerned, similar comments can be made regarding CLASS4, CLASS6, and CLASS8. We observe, however, that solver randomness seems slightly more pronounced as the tested version of **FLOW-CBP** (resp. **CP-CBP**) solved 30 (resp. 18) instances with one random seed but not with (at least) one other. For the P|*cont*|C_{max}, these numbers were 12 for the tested version of **BM** and 7 for the tested version of **CP-PCC**. We still point out that aggregated results do not show a large disparity in the range of solved instances: between 252 and 258 instances solved for the tested version of **FLOW-CBP** and between 360 and 364 instances solved for the tested version of **CP-CBP**. These range magnitudes are comparable to what was observed for the P|*cont*|C_{max}.

6.7 Lower bounds

In Tables 20 and 21, we present the outcome of the experiments aimed at assessing the quality of the P|*cont*|C_{max} and 1CBP lower bounds introduced in Sections 5.1 and 5.2. All three preprocessing techniques described in Section 4.3 were applied before calling the lower bounds. Note, however, that the preprocessing applied for the P|*cont*|C_{max} relaxation of one instance is not necessarily the same as the preprocessing applied for the 1CBP relaxation of that same instance. The columns in the tables provide, for each dataset and for each lower bound, the number of instances in the dataset for which that lower bound was the best lower bound (column "#eq."), the average deviation from the best lower bound (column "dev."), and the average CPU time needed to run the lower bound (column "T(s)"). For a given instance, we call “best lower bound" the highest lower bound obtained in our experiments for that instance (i.e., the maximum among the 30+ lower bounds obtained by our exact approaches in one hour and the tested lower bounds).

Table 20: Performance of the P|*cont*|C_{max} lower bounds

		L_1			L_2^{PCC}			$L_2^{PCC'}$			L_3^{PCC}			$L_3^{PCC'}$			L_4^{PCC}			$L_4^{PCC'}$		
Dataset	#inst	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)
CGCUT	3	0	8	0	1	7	1	1	7	0	2	6.7	176	2	6.7	1	1	1.7	0	1	1.7	0
HT	9	9	0	0	9	0	0	9	0	0	9	0	0	9	0	0	9	0	7	9	0	0
BENG	10	0	1	0	10	0	0	10	0	0	10	0	293	10	0	1	10	0	0	10	0	0
NGCUT	12	1	4	0	4	1.2	0	4	1.2	0	7	0.8	0	7	0.8	0	2	2.8	0	2	4	0
BKW	13	13	0	0	13	0	0	13	0	0	13	0	569	13	0	39	13	0	591	13	0	0
GCUT	13	3	69.8	0	3	62.6	98	3	62.6	4	4	55.2	662	4	55.2	141	6	15.2	277	3	27.5	1
CLASS1	50	6	1.5	0	31	0.7	0	31	0.7	0	31	0.7	13	31	0.7	0	45	0.1	0	41	0.2	0
CLASS2	50	0	1	0	50	0	0	50	0	0	50	0	84	50	0	0	50	0	0	50	0	0
CLASS3	50	6	4.7	0	16	3.7	1	16	3.7	0	17	3.5	460	17	3.5	4	35	0.8	1	29	1.1	0
CLASS5	50	14	13.2	0	20	11.5	15	20	11.5	0	21	11.3	640	21	11.3	17	28	2.3	73	20	3.3	0
CLASS7	50	19	7.9	0	31	4.4	0	31	4.4	0	35	2.9	0	35	2.9	0	30	2.8	0	30	3	0
CLASS9	50	44	2.5	0	49	0.1	0	49	0.1	0	50	0	0	50	0	0	32	4.4	0	32	4.4	0
CLASS10	50	1	8.8	0	11	7.7	15	11	7.7	1	12	7.6	1678	12	7.6	30	32	1.5	1380	22	2.3	0
Total	410	116	7.2	0	248	5.5	7	248	5.5	0	261	5	398	261	5	12	293	2	205	262	2.8	0

As one could expect, both the quality and the computation time of a P|*cont*|C_{max} lower bound increase as the relative complexity of the bound increases. The trivial lower bound L_1 yields the worst performance, even though it is very fast to compute. Lower bound L_2^{PCC} (obtained by solving the P||C_{max}) obtained slightly better results while also being relatively fast to compute. Lower bound L_3^{PCC} (obtained by solving the P||C_{max} with conflicts) obtained even better results, but at the expense of a significant increase in computation time. Interestingly, the quality of lower bounds $L_2^{PCC'}$ and $L_3^{PCC'}$ were as good as the quality of L_2^{PCC} and L_3^{PCC}, respectively, while being significantly faster to compute. Finally, L_4^{PCC} (obtained by solving a model inspired by the reflect formulation [35]) achieved the best results, with 293 instances out of 410 for which the approach obtained the best lower bound. However, the computation time required to achieve such results was significant (205 seconds on average). In contrast, $L_4^{PCC'}$ (obtained by solving the LP-relaxation of the model used in L_4^{PCC}) also produced relatively good results in a much shorter time.

Table 21: Performance of the 1CBP lower bounds

		L_1			L_2^{CBP}			$L_2^{CBP'}$			L_3^{CBP}			$L_3^{CBP'}$		
Dataset	#inst	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)	#eq.	dev.	T(s)
CGCUT	3	0	6.7	0	2	0.3	0	2	0.3	0	2	0.3	0	2	0.3	0
HT	9	9	0	0	9	0	0	9	0	0	9	0	0	9	0	0
BENG	10	0	1	0	10	0	0	10	0	0	10	0	4	10	0	0
NGCUT	12	1	4	0	2	4	0	2	4	0	3	2.9	0	3	2.9	0
BKW	13	13	0	0	13	0	0	13	0	0	13	0	298	13	0	34
GCUT	13	3	62.7	0	6	20.3	0	6	20.3	0	8	16.5	0	8	16.5	0
CLASS1	50	6	1.5	0	41	0.2	0	41	0.2	0	45	0.1	0	45	0.1	0
CLASS2	50	0	1	0	50	0	0	50	0	0	50	0	1	50	0	0
CLASS3	50	5	5.8	0	36	1	0	36	1	0	38	0.4	0	38	0.4	0
CLASS5	50	12	18.6	0	28	2.9	0	28	2.9	0	36	1.7	0	36	1.7	0
CLASS7	50	19	7.9	0	29	3.1	0	29	3.1	0	32	2.5	0	32	2.5	0
CLASS9	50	44	2.5	0	32	4.4	0	32	4.4	0	46	1.6	0	46	1.6	0
CLASS10	50	1	8.4	0	29	1.9	0	29	1.9	0	44	0.4	0	44	0.4	0
Total	410	113	7.8	0	287	2.4	0	287	2.4	0	336	1.4	10	336	1.4	1

For the 1CBP too, both the quality and the computation time of a lower bound increase as the relative complexity of the bound increases. Compared to the P|*cont*|C_{max} lower bounds, 1CBP lower bounds are faster to compute and have better quality on average. The former remark could be explained by the fact that 1CBP lower bounds do not require an iterative process, unlike some P|*cont*|C_{max} lower bounds.

Overall, these findings are significant as they outline that it is possible to derive good quality lower bounds for both the P|*cont*|C_{max} and the 1CBP. We even observed that, for some instances, the best lower bound among the ones tested in these experiments was higher than the best lower bound obtained by the approaches tested in the previous experiments. These results also suggest that the tested lower bounds should perform well for the SPP, confirming previous observations on the matter [2, 27].

7 Conclusion

We studied the parallel processor scheduling problem with contiguity constraints (P|*cont*|C_{max}) and the one-dimensional bin packing problem with contiguity constraints (1CBP). These two problems are relevant for the research community because they are used as components of effective decomposition algorithms for various two-dimensional cutting and packing problems. After enumerating the main approaches proposed in the literature to solve the P|*cont*|C_{max} and the 1CBP exactly, we reviewed existing mathematical formulations for the two problems together with some model enhancements. We also investigated lower bounding techniques for the two problems and assessed whether recent advances in the cutting and packing area such as reduced-cost variable fixing, meet-in-the-middle patterns, and the reflect formulation could be used to solve the problems more efficiently. We then empirically evaluated the performance of each model when solved with a state-of-the-art solver on a large set of instances from the literature. We also tested the reviewed model enhancements and lower bounding techniques and made a number of relevant observations.

From the problem point of view, it is interesting to know that (i) the most important feature determining whether an approach can solve a given instance is the model used by the approach, (ii) the second most important feature is whether or not the approach uses a preprocessing step aimed at prepacking large items and adjusting the instance dimensions and, for the 1CBP, whether or not a lower bound is given to the approach, (iii) less important features, but which are still relevant overall for integer linear programming (ILP) models, are the use of normal (or meet-in-the-middle) patterns to reduce the number of variables involved and the exploitation of item multiplicity, (iv) features that do not appear to be empirically useful are the use of symmetry breaking constraints, destructive bounds, and reduced-cost variable fixing, (v) approaches **BM** and **CP-PCC** obtain very good empirical results, even without any model enhancement techniques, (vi) lower bounds obtained by solving the continuous relaxation of flow models, including reflect, are both fast to compute and effective, (vii) on some notoriously difficult datasets, it seems beneficial to use an approach based on constraint programming (CP) over an approach based on ILP, and (viii) for most strip packing instances, the (numerical instance resulting from the) P|*cont*|C_{max} relaxation seems easier to solve than the (numerical instance resulting from the) 1CBP relaxation. As far as the latter comment is concerned, we emphasized that both relaxations are relevant in practice because they do not necessarily lead to the same objective solution value, as demonstrated in Appendix G. For example, for SPP instance "cl_06_020_06", the optimal solution value of the 1CBP relaxation was 206 whereas the optimal solution value of the P|*cont*|C_{max} relaxation was 208. In total, we identified at least 4 instances for which the 1CBP relaxation and the P|*cont*|C_{max} relaxation did not have the same optimal solution value.

From the optimization point of view, it is interesting to observe that (i) even though some model enhancements such as symmetry-breaking constraints are useful in theory, they do not necessarily translate into any empirical improvement, (ii) whereas solver randomness only has a marginal effect on empirical results, that effect does exist and should be taken into consideration when assessing the performance of an approach, (iii) the goal of finding the best unique approach to solve a given combinatorial optimization problem might sometimes be overvalued given the fact that running a few sequential approaches for a limited time period may provide better empirical results than running a unique approach for the same duration.

As far as challenging and open problems are concerned, we identified three interesting research directions. Based on the outcomes of this survey, a first natural research direction is the search for more effective exact algorithms to solve the P|*cont*|C_{max} and the 1CBP as there are still dozens of medium-sized instances that could not be solved to optimality within one hour of computation time. To the best of our knowledge, no major breakthrough has happened in the area since the **FLOW-PCC** formulation, whereas it is possible that alternative ILP models or CP formulations could obtain better empirical results. Considering that the 1CBP is very close to the bin packing problem (BPP), which was extensively studied

in the recent years, it would also be interesting to determine whether any of the main features utilized by the effective approaches recently proposed for the BPP [32, 70, 79] could be extended to tackle the 1CBP. Other methodologies to solve the P|*cont*|C_{max} and the 1CBP more effectively include the search for good quality and fast to compute bounding procedures. We reviewed and empirically assessed various relaxation-based lower bounding procedures for both problems and our results showed that there was still some room for improvement. Regarding upper bounding procedures, we could not find many methods tailored to the P|*cont*|C_{max} or to the 1CBP besides adaptations of the first-fit, best-fit, and worst-fit heuristics suggested by Martello et al. [61] and Côté at al. [27]. Indeed, as these two problems are mostly solved as components of a decomposition approach, existing heuristics are usually aimed at finding good quality solutions for the overall problem, not for individual components. One could certainly adapt the many existing strip packing heuristics [1, 18, 59, 80] to tackle the P|*cont*|C_{max} and the 1CBP, but it would probably be more interesting to understand whether problem-specific structures can be exploited to derive more effective ad hoc heuristics. We also identify as promising the search for automated hyper-algorithms able to select which method (or sequence of methods) among the ones reviewed in this work is better suited to solve a given instance based on its features (following the recent trend that aims at incorporating machine learning techniques into optimization algorithms [11]).

A second research direction concerns the integration of the results found in this review into the decomposition framework proposed by Côté at al. [27], where the main problem is (a form of) 1CBP or P|*cont*|C_{max}, and how this framework could be enhanced. For example, to the best of our knowledge, callback functions allowing the user to add constraints during the solver execution are not available in the CP solver of Cplex whereas these are available (and were shown to be useful) in ILP solvers. It would be interesting to see, given the surprisingly good results displayed by CP on certain classes of P|*cont*|C_{max} and 1CBP instances, how **CP-PCC** and **CP-CBP** could be integrated into the decomposition framework. Côté at al. [27] also outlined that no-good cuts could not be used with integer variables, whereas these cuts are needed in the decomposition framework to forbid solutions of the main problem that were previously shown to be infeasible. We pointed out that item multiplicity could be exploited using binary expansion instead. However, to the best of our knowledge, the problem of strengthening a no-good-cut by finding the so-called "minimum infeasible subset" (the minimum subset of items causing infeasibility, also known as "MIS") and applying a lifting procedure was never thoroughly studied in the context of a binary expansion. It would also be interesting to conduct an empirical study focusing on the secondary problem of the decomposition framework, a scheduling problem with non-overlapping constraints (sometimes referred to as "y-check" in the literature). So far, the problem has been solved with either a branch-and-bound algorithm or CP. Even though the computing times displayed in the literature show that most of the computational effort is spent solving the main problem, it sometimes occurs for very large size instances that solving the secondary problem also takes a significant amount of time. Considering that the secondary problem often needs to be solved multiple times (especially if one computes minimum infeasible subsets), any significant improvement in the solving time of the secondary problem would be relevant. Finally, the way of finding the minimum infeasible subsets is also worth investigating. Côté at al. [27] suggested various heuristic strategies that were shown to be effective in general, but those can be time consuming in practice, especially for large size instances. It would be interesting to determine whether one strategy is more suitable for instances displaying certain features or if the set of strategies used should depend on the number of solutions that has been tested so far in the secondary problem. Machine learning techniques seem particularly suitable to answer such questions, but these approaches usually require very large datasets for the training phase, motivating the need for supplementary benchmark instances and dataset generators.

A third research direction concerns the extension of Côté at al.'s [27] decomposition framework to

other two-dimensional cutting and packing problems. To the best of our knowledge, such a framework was used to solve the strip packing problem [27, 37, 65], the two-dimensional bin packing problem [29], and practical packing problems in logistics [28, 38, 85]. The same framework could also be used to solve the two-dimensional knapsack problem [20] and any of the aforementioned problems that incorporates additional components that can be handled in the main or secondary problem such as orthogonal rotation [37], guillotine constraints [63], multiple and/or variable-sized containers [42], loading-unloading constraints [28], and defects [64]. That being said, all reviewers would not necessarily agree that solely applying an existing framework to another, closely related, optimization problem is enough in terms of original research to warrant publication. Hyperbolically, if one considers the 4 aforementioned classes of two-dimensional packing problems (strip packing, knapsack, orthogonal packing, and bin packing) and the 5 aforementioned supplementary features (orthogonal rotation, guillotine constraints, multiple and/or variable-sized containers, loading-unloading constraints, and defects), one could theoretically write $4 \times 2^5 = 128$ research articles. Therefore, it would appear to be beneficial if each framework extension also came with significant novelties, either in the form of real-world problem specificities, better ways to solve the main and/or the secondary problem, original ways to integrate the problem features into the main and/or the secondary problem, among others. Finally, the question remains open as to whether a similar decomposition framework could also be used to solve three-dimensional cutting and packing problems [73], and if yes, whether the resulting computational behaviors would be competitive.

References

[1] R. Alvarez-Valdés, F. Parreño, and J Tamarit. Reactive GRASP for the strip-packing problem. *Computers & Operations Research*, 35:1065–1083, 2008.

[2] R. Alvarez-Valdés, F. Parreño, and J. Tamarit. A branch and bound algorithm for the strip packing problem. *OR Spectrum*, 31:431–459, 2009.

[3] E. Anand and R. Panneerselvam. Literature review of open shop scheduling problems. *Intelligent Information Management*, 7:33, 2015.

[4] C. Arbib, F. Marinelli, F. Rossi, and F. Di Iorio. Cutting and reuse: An application from automobile component manufacturing. *Operations Research*, 50:923–934, 2002.

[5] R. Baldacci and M.A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183:1136–1149, 2007.

[6] R. Baldacci, M.A. Boschetti, M. Ganovelli, and V. Maniezzo. Algorithms for nesting with defects. *Discrete Applied Mathematics*, 163:17–33, 2014.

[7] M. Barkel and M. Delorme. Arcflow formulations and constraint generation frameworks for the two bar charts packing problem. *INFORMS Journal on Computing*, 35:475–494, 2023.

[8] J.E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[9] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33:49–64, 1985.

[10] G. Belov, V. Kartak, H. Rohling, and G. Scheithauer. One-dimensional relaxations and LP bounds for orthogonal packing. *International Transactions in Operational Research*, 16:745–766, 2009.

[11] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290:405–421, 2021.

[12] B.-E. Bengtsson. Packing rectangular pieces—a heuristic approach. *The computer journal*, 25:353–357, 1982.

[13] J.O. Berkey and P.Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, 38:423–429, 1987.

[14] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the two-dimensional level strip packing problem. *4OR*, 6:361–374, 2008.

[15] M.A. Boschetti, A. Mingozzi, and E. Hadjiconstantinou. New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[16] M.A. Boschetti and L. Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58:1774–1791, 2010.

[17] F. Brandao and J.P. Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.

[18] E.K. Burke, M.R. Hyde, and G. Kendall. A squeaky wheel optimisation methodology for two-dimensional strip packing. *Computers & Operations Research*, 38:1035–1044, 2011.

[19] E.K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52:655–671, 2004.

[20] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32:5–14, 2004.

[21] P.M. Castro and I.E. Grossmann. From time representation in scheduling to the solution of strip packing problems. *Computers & Chemical Engineering*, 44:45–57, 2012.

[22] P.M. Castro and J.F. Oliveira. Scheduling inspired models for two-dimensional packing problems. *European Journal of Operational Research*, 215:45–56, 2011.

[23] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[24] A.M. Chugay and A.V. Zhuravka. Packing optimization problems and their application in 3D printing. In *Advances in Computer Science for Engineering and Education III 3*, pages 75–85, 2021.

[25] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183:1196–1211, 2007.

[26] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35:944–959, 2008.

[27] J.-F. Côté, M. Dell'Amico, and M. Iori. Combinatorial Benders' cuts for the strip packing problem. *Operations Research*, 62:643–661, 2014.

[28] J.-F. Côté, M. Gendreau, and J.Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, 62:1126–1141, 2014.

[29] J.-F. Côté, M. Haouari, and M. Iori. Combinatorial Benders' decomposition for the two-dimensional bin packing problem. *INFORMS Journal on Computing*, 33:963–978, 2021.

[30] J.-F. Côté and M. Iori. The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, 30:646–661, 2018.

[31] V.L. de Lima, C. Alves, F. Clautiaux, M. Iori, and J.M. Valério de Carvalho. Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research*, 296:3–21, 2022.

[32] V.L. de Lima, M. Iori, and F.K. Miyazawa. Exact solution of network flow models with strong relaxations. *Mathematical Programming*, 197:813–846, 2023.

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

[34] M. Delorme, S. García, J. Gondzio, J. Kalcsics, D. Manlove, and W. Pettersson. New algorithms for hierarchical optimisation in kidney exchange programs. *Operations Research,* to appear, 2023.

[35] M. Delorme and M. Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32:101–119, 2020.

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

[37] M. Delorme, M. Iori, and S. Martello. Logic based benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, 78:290–298, 2017.

[38] M. Delorme and J. Wagenaar. Exact decomposition approaches for a single container loading problem with stacking constraints and medium-sized weakly heterogeneous items. *Omega*, 125:103039, 2024.

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

[40] A. Erzin, G. Melidi, S. Nazarenko, and R. Plotnikov. Two-bar charts packing problem. *Optimization Letters*, 15:1955–1971, 2021.

[41] I. Friedow and G. Scheithauer. Using contiguous 2D-feasible 1D cutting patterns for the 2D strip packing problem. In K.F. Dörner, I. Ljubic, G. Pflug, and G. Tragler, editors, *Operations Research Proceedings 2015*, pages 71–77. Springer International Publishing, 2017.

[42] F. Furini and E. Malaguti. Models for the two-dimensional two-stage cutting stock problem with multiple stock size. *Computers & Operations Research*, 40:1953–1962, 2013.

[43] R. Gedik, C. Rainwater, H. Nachtmann, and E.A. Pohl. Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals. *European Journal of Operational Research*, 251:640–650, 2016.

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

[45] S. Grandcolas and C. Pinto. A SAT encoding for multi-dimensional packing problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7*, pages 141–146, 2010.

[46] A. Grange, I. Kacem, and S. Martin. Algorithms for the bin packing problem with overlapping items. *Computers & Industrial Engineering*, 115:331–341, 2018.

[47] Gurobi optimization. Gurobi online documentation. `https://www.gurobi.com/documentation/current/refman/seed.html`, 2024. accessed 26 January 2024.

[48] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83:39–56, 1995.

[49] J.C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[50] E.B.C.H. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128:34–57, 2001.

[51] W. Huang and K. He. On the weak computability of a four dimensional orthogonal packing and time scheduling problem. *Theoretical Computer Science*, 501:1–10, 2013.

[52] M. Iori, V.L. de Lima, S. Martello, F.K. Miyazawa, and M. Monaci. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research*, 289:399–415, 2021.

[53] M. Iori, S. Martello, and M. Monaci. *Metaheuristic Algorithms for the Strip Packing Problem*, pages 159–179. Springer US, Boston, MA, 2003.

[54] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[55] 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.

[56] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38:3–13, 2011.

[57] D. Kowalczyk and R. Leus. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20:355–372, 2017.

[58] O.R. Letelier, F. Clautiaux, and R. Sadykov. Bin packing problem with time lags. *INFORMS Journal on Computing*, 34:2249–2270, 2022.

[59] S.C.H. Leung, D. Zhang, and K.M. Sim. A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research*, 215:57–69, 2011.

[60] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[61] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15:310–319, 2003.

[62] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management science*, 44:388–399, 1998.

[63] M. Martin, E.G. Birgin, R.D. Lobato, R. Morabito, and P. Munari. Models for the two-dimensional rectangular single large placement problem with guillotine cuts and constrained pattern. *International Transactions in Operational Research*, 27:767–793, 2020.

[64] M. Martin, R. Morabito, and P. Munari. Two-stage and one-group two-dimensional guillotine cutting problems with defects: a CP-based algorithm and ILP formulations. *International Journal of Production Research*, 60:1854–1873, 2022.

[65] K. Matsushita, Y. Hu, H. Hashimoto, S. Imahori, and M. Yagiura. Exact algorithms for the rectilinear block packing problem. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 12:JAMDSM0074–JAMDSM0074, 2018.

[66] M. Mesyagutov, E. Mukhacheva, G. Belov, and G. Scheithauer. Packing of one-dimensional bins with contiguous selection of identical items: An exact method of optimal solution. *Automation and Remote Control*, 72:141–159, 2011.

[67] J.F. Oliveira, A. Neuenfeldt Júnior, E. Silva, and M.A. Carravilla. A survey on heuristics for the two-dimensional rectangular strip packing problem. *Pesquisa Operacional*, 36:197–226, 2016.

[68] C. Paquay, M. Schyns, and S. Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23:187–213, 2016.

[69] F. Parreño, M.T. Alonso, and R. Alvarez-Valdés. Solving a large cutting problem in the glass manufacturing industry. *European Journal of Operational Research*, 287:378–388, 2020.

[70] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183:483–523, 2020.

[71] C.N. Potts. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10:155–164, 1985.

[72] E. Silva, J.F. Oliveira, T. Silveira, L. Mundim, and M.A. Carravilla. The floating-cuts model: a general and flexible mixed-integer programming model for non-guillotine and guillotine rectangular cutting problems. *Omega*, 114:102738, 2023.

[73] E.F. Silva, T.A.M. Toffolo, and T. Wauters. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. *Computers & Operations Research*, 109:12–27, 2019.

[74] M. Sindelar, R.K. Sitaraman, and P. Shenoy. Sharing-aware algorithms for virtual machine colocation. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 367–378, 2011.

[75] T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima. A SAT-based method for solving the two-dimensional strip packing problem. *Fundamenta Informaticae*, 102:467–487, 2010.

[76] T. Strecker and L. Hennig. Automatic layouting of personalized newspaper pages. In Bernhard Fleischmann, Karl-Heinz Borgwardt, Robert Klein, and Axel Tuma, editors, *Operations Research Proceedings 2008*, pages 469–474. Springer Berlin Heidelberg, 2009.

[77] 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.

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

[79] L. Wei, Z. Luo, R. Baldacci, and A. Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32:428–443, 2020.

[80] L. Wei, W.-C. Oon, W. Zhu, and A. Lim. A skyline heuristic for the 2d rectangular packing and strip packing problems. *European Journal of Operational Research*, 215:337–346, 2011.

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

[82] L.A. Wolsey. Valid inequalities, covering problems and discrete dynamic programs. In *Annals of Discrete Mathematics*, volume 1, pages 527–538. 1977.

[83] H. Xiong, S. Shi, D. Ren, and J. Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

[84] J. Yang and J. Leung. The ordered open-end bin-packing problem. *Operations Research*, 51:759–770, 2003.

[85] X. Zhang, L. Chen, M. Gendreau, and A. Langevin. A branch-and-cut algorithm for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 302:259–269, 2022.

APPENDIX

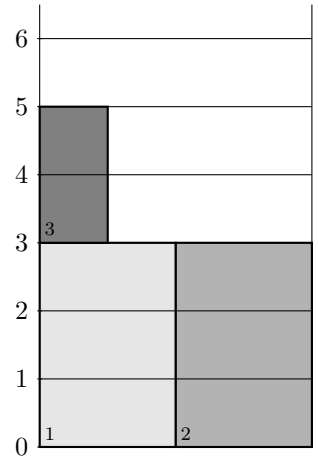
A MIM patterns cannot easily be used for the 1CBP

A direct application of the MIM patterns introduced by Côté and Iori [30] to the 1CBP is not feasible. Consider the scenario depicted in Figure 5a, featuring a simple instance with 3 items. If we assume that a valid upper bound UB is 6, we show in Figure 5b the optimal 1CBP solution obtained using normal patterns, with objective value 5. If we use the MIM patterns in which the threshold t is set to 2, we show in Figure 5c that an optimal 1CBP solution has objective value 6. This is due to the fact that MIM patterns assume that the items packed after the threshold (i.e., above the bold line in the figure) are pushed as much to the top as possible, removing any solution with objective value 5.

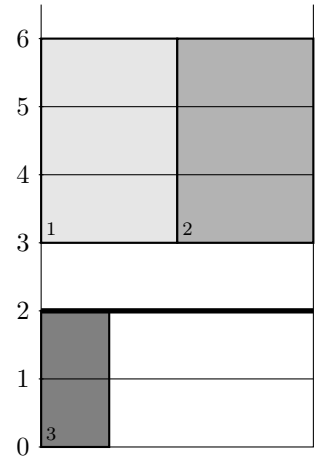
Figure 5: Optimal solution of a 1CBP instance using normal patterns versus MIM patterns

items	w	h	NP	MIM
item 1	2	3	{0,2,3}	{0,3}
item 2	1	2	{0,3}	{0,4}
item 3	2	3	{0,2,3}	{0,3}
Strip with $W = 4$ and UB=6				

(a) Instance description



(b) Optimal packing using normal patterns



(c) Optimal packing using MIM patterns with $t = 2$

B Model REFLECT

In this section of the appendix, we first introduce two reflect-based models tailored to the P$|cont|$C$_{\max}$ and we illustrate why an optimal solution of such models cannot necessarily be transformed into an equivalent (feasible) **BM** or **FLOW-PCC** solution. In all models, we assume W to be an even number. The case where W is an odd number can be handled using the techniques proposed by Delorme and Iori [35].

B.1 Model FLOW-REFLECT:

Model **FLOW-REFLECT** uses graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where vertex set $\mathcal{V} = \{0, 1, \ldots, \frac{W}{2}\}$ and where arc set $\mathcal{A}$ is composed of:

- standard item arcs $\mathcal{A}_1^s, \ldots, \mathcal{A}_n^s$ where $\mathcal{A}_i^s$ contains every arc $(d, d + w_i, s)$ such that $d \in \mathcal{W}_i$ and $d + w_i \leq \frac{W}{2}$; a binary decision variable x_{ide}^s is associated with each standard item arc;
- reflected item arcs $\mathcal{A}_1^r, \ldots, \mathcal{A}_n^r$ where $\mathcal{A}_i^r$ contains every arc $(d, W - d - w_i, r)$ such that $d \in \mathcal{W}_i$ and $d + w_i > \frac{W}{2}$; a binary decision variable x_{ide}^r is associated with each reflected item arc;
- standard loss arcs $\mathcal{A}_0^s$ containing every arc (d, e, s) such that $d \in \mathcal{W}, d < \frac{W}{2}, e = \min\{e' \in \mathcal{W} : e' > d\}$ an integer variable ψ_{de} is associated with each standard loss arc;
- reflected loss arc $(\frac{W}{2}, \frac{W}{2}, r)$; an integer variable Ψ is associated with the reflected loss arc;

If selected in a solution, item arc $(d, e, \{s, r\})$ carries h_i units of flow from node d to node e. If selected in a solution, loss arc $(d, e, \{s, r\})$ carries 1 unit of flow from node d to node e (note that a loss arc may be selected multiple times). A feasible **FLOW-REFLECT** solution can then be defined as a set of z pairs of colliding paths (i.e., two paths both starting in zero and ending in the same vertex but only one of the two contains a reflected arc). The model can be defined as follows:

$$\min \quad \sum_{i\in\mathcal{N}} \sum_{(d,e,r)\in A_i^r} h_i x_{ide}^r + \Psi \tag{66}$$

$$\text{s.t.} \quad \sum_{(d,e)\in A_i^s} h_i x_{ide}^s + \sum_{(d,e)\in A_i^r} h_i x_{ide}^r = 1 \qquad i \in \mathcal{N}, \tag{67}$$

$$\sum_{i\in\mathcal{N}} \sum_{(0,e,s)\in\mathcal{A}_i^s} h_i x_{i0e}^s + \sum_{i\in\mathcal{N}} \sum_{(0,e,r)\in\mathcal{A}_i^r} h_i x_{i0e}^r + \sum_{(0,e,s)\in\mathcal{A}_0^s} \psi_{0e} = 2\left(\sum_{i\in\mathcal{N}} \sum_{(d,e,r)\in A_i^r} h_i x_{ide}^r + \Psi\right), \tag{68}$$

$$\sum_{i\in\mathcal{N}} \sum_{(d,e,s)\in\mathcal{A}_i^s} h_i x_{ide}^s + \sum_{(d,e,s)\in\mathcal{A}_0^s} \psi_{de} = \sum_{i\in\mathcal{N}} \sum_{(e,f,s)\in\mathcal{A}_i^s} h_i x_{ief}^s + \sum_{i\in\mathcal{N}} \sum_{(e,f,r)\in\mathcal{A}_i^r} h_i x_{ief}^r + \sum_{(e,f)\in\mathcal{A}_0^s} \psi_{ef} \qquad e \in \mathcal{V}\setminus\{0, \frac{W}{2}\}, \tag{69}$$

$$\sum_{i\in\mathcal{N}} \sum_{(d,e,s)\in\mathcal{A}_i^s} h_i x_{ide}^s + \sum_{(d,e,s)\in\mathcal{A}_0^s} \psi_{de} = 2\Psi \qquad e = \frac{W}{2}, \tag{70}$$

$$x_{ide}^r \in \{0,1\} \qquad i \in \mathcal{N}, (d,e,r) \in \mathcal{A}_i^r, \tag{71}$$

$$x_{ide}^s \in \{0,1\} \qquad i \in \mathcal{N}, (d,e,s) \in \mathcal{A}_i^s, \tag{72}$$

$$\psi_{de} \in \mathbb{N}_0 \qquad (d,e,s) \in \mathcal{A}_0^s, \tag{73}$$

$$\Psi \in \mathbb{N}_0. \tag{74}$$

B.2 Model BM-REFLECT

Model **BM-REFLECT** is a hybridization of models **BM** and **FLOW-REFLECT**. The model considers a column set $\mathcal{W} = \{0, \ldots, \frac{W}{2} - 1\}$. For each item, it also considers two subsets of columns in which the first slice of a given $i \in \mathcal{N}$ item can be packed:

- Columns $\mathcal{W}_i^s = \{0, \ldots, \frac{W}{2} - w_i\}$; if the first slice of item i is packed in column $p \in \mathcal{W}_i^s$, then a slice of item i is also packed in columns $p+1, p+2, \ldots, p+w_i-1$;
- Columns $\mathcal{W}_i^r = \left\{ \max\{0, \frac{W}{2} - w_i + 1\}, \ldots, \min\{\frac{W}{2} - 1, W - w_i\}\right\}$; if the first slice of item i is packed in column $p \in \mathcal{W}_i^r$, then a slice of item i is also packed in:
  - columns $p+1, p+2, \ldots, \frac{W}{2} - 1$ to account for pre-reflection;
  - columns $\frac{W}{2} - 1, \frac{W}{2} - 2, \ldots, W - p - w_i$ to account for post-reflection;

We also adapt set $\mathcal{W}_i(p)$ to **BM-REFLECT** as follows:

- $\mathcal{W}_i^s(p) = \{p' \in \mathcal{W}_i^s : 0 \leq p - w_i + 1 \leq p' \leq p\}$, for all $i \in \mathcal{N}$ and for all $p \in \mathcal{W}$;
- $\mathcal{W}_i^{r1}(p) = \{p' \in \mathcal{W}_i^r : 0 \leq \frac{W}{2} - w_i + 1 \leq p' \leq p\}$, for all $i \in \mathcal{N}$ and for all $p \in \mathcal{W}$;
- $\mathcal{W}_i^{r2}(p) = \{p' \in \mathcal{W}_i^r : 0 \leq W - p - w_i \leq p' \leq \frac{W}{2} - 1\}$, for all $i \in \mathcal{N}$ and for all $p \in \mathcal{W}$;

The model uses two sets of binary decision variables:

- x_{ip}^s taking value 1 if the first slice of item $i \in \mathcal{N}$ is packed in column $p \in \mathcal{W}_i^s$;
- x_{ip}^r taking value 1 if the first slice of item $i \in \mathcal{N}$ is packed in column $p \in \mathcal{W}_i^r$;

together with integer decision variable z indicating the maximum column height.

BM-REFLECT can be defined as follows:

$$\min \quad z \tag{75}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{W}_i^s} x_{ip}^s + \sum_{p \in \mathcal{W}_i^r} x_{ip}^r = 1 \qquad i \in \mathcal{N}, \tag{76}$$

$$\sum_{i \in \mathcal{N}} h_i \left(\sum_{p' \in \mathcal{W}_i^s(p)} x_{ip'}^s + \sum_{p' \in \mathcal{W}_i^{r1}(p)} x_{ip'}^r + \sum_{p' \in \mathcal{W}_i^{r2}(p)} x_{ip'}^r \right) \leq 2z \qquad p \in \mathcal{W}, \tag{77}$$

$$x_{ip}^s \in \{0,1\} \qquad i \in \mathcal{N}, p \in \mathcal{W}_i^s \tag{78}$$

$$x_{ip}^r \in \{0,1\} \qquad i \in \mathcal{N}, p \in \mathcal{W}_i^r. \tag{79}$$

B.3 Transforming a FLOW-REFLECT solution into a FLOW-PCC solution

A feasible **FLOW-REFLECT** solution with objective value z cannot always be converted into a feasible **FLOW-PCC** solution with the same objective value. Consider, for example, the instance described in Table 6a. An optimal solution obtained by **FLOW-PCC** with objective value 5 is depicted in Figure 6b whereas an optimal solution obtained by **FLOW-REFLECT** with objective value 4 is depicted in Figure 6c. Following the strategy described by Delorme and Iori [37], the **FLOW-REFLECT** solution can be decomposed into the following pairs of colliding paths:

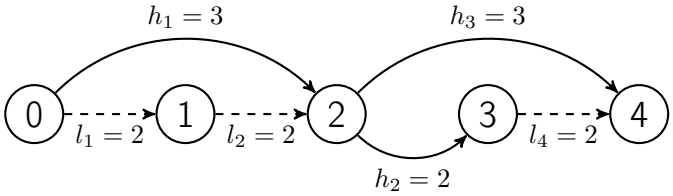
- 3 times $0 \to 2 \to 2$ and $0 \to 2$;
- 1 time $0 \to 1 \to 2 \to 2$ and $0 \to 1 \to 2$.

In P|*cont*|C_{max} terms, such a solution packs the unique slice of item 2 into two columns at the same time: a first half in the first column and the other half in the fourth column, which is why a solution with objective value 4 can be reached. This problem occurs in the reflect-based models tailored to the P|*cont*|C_{max} because of the presence of the h_i coefficients in the flow-conservation constraints (which did not appear in the original reflect formulation for the cutting stock problem). One could still reconstruct a feasible P|*cont*|C_{max} solution from an optimal **FLOW-REFLECT** solution by "unfolding" the solution graph and duplicating each standard arc, as shown in Figure 6d. One can then solve a simplified version of **FLOW-PCC** in which at most two arcs per item are considered. Preliminary experiments indicated that the resulting two-step matheuristic approach was not promising.

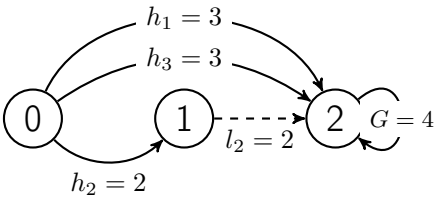
Figure 6: Comparison of different solutions

	w	h
strip	4	∞
item 1	2	3
item 2	1	2
item 3	2	3

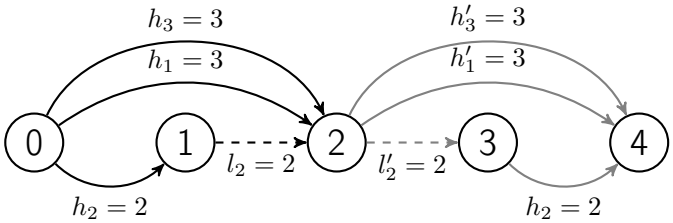
(a) Instance description



(b) Optimal **FLOW-PCC** solution



(c) Optimal **REFLECT** solution



(d) Extended **REFLECT** solution

C Supplementary tables evaluating the baseline version of the tested models

Table 22: Results of **BKRS**, **BM**, **FLOW-PCC**, and **CP-PCC** without preprocessing techniques

Dataset	#inst	**BKRS**			**BM**			**FLOW-PCC**			**CP-PCC**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	1	2400	0	2	1201	1	2	1236	54	1	2400	0
HT	9	6	1355	232	8	404	5	9	39	39	8	414	16
BENG	10	8	1232	640	10	8	8	6	1582	234	10	1	1
NGCUT	12	12	39	39	12	1	1	12	40	40	12	1	1
BKW	13	3	3007	1029	5	2222	15	4	2666	565	3	2942	751
GCUT	13	4	2842	1136	11	611	66	9	1128	29	10	1054	290
CLASS1	50	50	1	1	50	0	0	50	0	0	39	851	75
CLASS2	50	47	515	319	50	7	7	49	209	139	50	1	1
CLASS3	50	34	1240	130	48	250	110	43	505	1	31	1369	2
CLASS5	50	34	1315	240	48	148	4	46	331	47	38	876	16
CLASS7	50	50	4	4	50	0	0	50	10	10	50	0	0
CLASS9	50	50	0	0	50	0	0	50	0	0	50	0	0
CLASS10	50	11	2887	362	23	2045	218	19	2331	260	18	2305	2
Total	410	310	991	150	367	406	32	349	583	55	320	812	28

Table 23: Results of **APT**, **APTP**, **FLOW-CBP**, and **CP-CBP** without a valid LB

Dataset	#inst	**APT**			**APTP**			**FLOW-CBP**			**CP-CBP**		
		#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)	#opt	T(s)	T_{opt}(s)
CGCUT	3	2	1206	8	2	1205	7	2	1314	171	1	2400	0
HT	9	9	31	31	9	18	18	9	81	81	8	413	14
BENG	10	6	2079	1065	8	1095	468	2	2880	2	10	37	37
NGCUT	12	12	10	10	12	13	13	12	9	9	12	1	1
BKW	13	3	2798	123	5	2504	751	4	2524	104	6	2242	658
GCUT	13	4	2534	136	4	2618	409	3	2770	1	8	1677	475
CLASS1	50	42	792	258	36	1043	49	26	1782	103	36	1037	41
CLASS2	50	41	846	242	41	786	169	30	1677	396	49	123	52
CLASS3	50	14	2627	126	19	2240	22	13	2668	16	24	1892	39
CLASS5	50	21	2116	67	20	2161	3	20	2161	4	24	1875	3
CLASS7	50	46	346	63	41	668	24	42	642	78	50	13	13
CLASS9	50	50	1	1	50	9	9	50	6	6	50	0	0
CLASS10	50	3	3384	6	6	3185	141	3	3384	1	11	2812	19
Total	410	253	1463	137	253	1429	83	216	1752	93	289	1097	49

Table 24: Model-specific metrics for **APT**, **APTP**, and **FLOW-CBP** without a valid LB

Dataset	#inst	**APT**				**APTP**				**FLOW-CBP**			
		cont.	#var	#const	#nz	cont.	#var	#const	#nz	cont.	#var	#const	#nz
CGCUT	3	129.7	18 421	18 455	15 730 516	240.1	18 420	681	456 687	240.1	18 745	359	55 261
HT	9	15.2	581	604	20 401	21.7	580	76	3219	21.7	608	51	1740
BENG	10	49.5	10 184	10 274	1 403 530	89.4	10 183	281	78 720	89.4	10 280	187	30 550
NGCUT	12	29.4	430	442	24 621	36.3	429	89	4841	36.3	469	52	1289
BKW	13		255 108	255 462	236 366 157	177.7	255 107	733	3 758 896	177.7	255 298	545	765 323
GCUT	13		81 919	68 631	368 278 939	4240.5	81 918	5833	27 124 252	4240.5	84 817	2928	245 736
CLASS1	50	154.8	3858	3886	700 154	186.6	3857	192	24 444	186.6	3940	112	11 571
CLASS2	50	33.9	4587	4647	438 127	60.1	4586	190	29 018	60.1	4653	127	13 759
CLASS3	50		18 344	18 379	12 477 412	502.2	18 343	666	341 375	502.2	18 660	352	55 030
CLASS5	50		32 543	32 569	52 822 092	1613.6	32 542	1391	1 613 877	1613.6	33 217	709	97 609
CLASS7	50	1544.7	1357	1363	484 073	1580.9	1356	365	66 804	1580.9	1527	187	4048
CLASS9	50	3340.5	167	169	41 092	3340.2	166	136	7801	3340.2	206	70	441
CLASS10	50		53 069	53 120	65 441 994	909.7	53 068	1662	1 904 061	909.7	53 875	858	159 204
Total	410	306	24 988	24 606	35 469 121	1145	24 987	786	1 470 961	1145	25 355	414	74 949

D Supplementary table evaluating the impact of variable reduction techniques

Table 25: Model-specific metrics for **APTP** and **FLOW-CBP** with/without normal patterns

Dataset	#inst	APTP				FLOW-CBP			
		Baseline		Normal patterns		Baseline		Normal patterns	
		#var	#const	#var	#const	#var	#const	#var	#const
CGCUT	3	18 420	681	18 007(-10.4%)	681(0%)	18 745	359	18 323(-10.3%)	359(0%)
HT	9	580	76	568(-2.4%)	76(0%)	608	51	596(-2.4%)	51(0%)
BENG	10	10 183	281	10 183(-0%)	281(0%)	10 280	187	10 280(-0%)	187(0%)
NGCUT	12	429	89	322(-31.8%)	89(0%)	469	52	349(-32.1%)	52(0%)
BKW	13	255 107	794	254 895(-2.4%)	733(0%)	255 298	545	255 084(-2.4%)	545(0%)
GCUT	13	81 918	6319	70 677(-24.4%)	5833(0%)	84 817	2928	72 976(-26.5%)	2928(0%)
CLASS1	50	3857	192	3850(-6.1%)	192(0%)	3940	112	3933(-6.6%)	112(0%)
CLASS2	50	4586	190	4585(-0.2%)	190(0%)	4653	127	4652(-0.2%)	127(0%)
CLASS3	50	18 343	679	18 268(-12.6%)	666(0%)	18 660	352	18 577(-13.4%)	352(0%)
CLASS5	50	32 542	1512	32 240(-14.7%)	1391(0%)	33 217	709	32 881(-16.4%)	709(0%)
CLASS7	50	1356	389	807(-39.9%)	365(0%)	1527	187	888(-43.5%)	187(0%)
CLASS9	50	166	200	89(-13.7%)	136(0%)	206	70	96(-21.3%)	70(0%)
CLASS10	50	53 068	1662	52 907(-4.3%)	1662(0%)	53 875	858	53 705(-4.6%)	858(0%)
Total	410	24 987	830	24 475(-13.1%)	786(0%)	25 355	414	24 802(-14.9%)	414(0%)

E Model for the maximum coverage problem

The following model determines a combination of approaches that solves the maximum number of instances. It uses:

- set $\mathcal{A}$ containing every considered approach;
- set $\mathcal{I}$ containing every considered instances;
- set $\mathcal{T}$ containing every considered time limit;
- parameter s_{iat} is equal to 1 if instance i is solved by approach a in t seconds or less ($i \in \mathcal{I}, a \in \mathcal{A}, t \in \mathcal{T}$);
- binary decision variable x_{at} taking value 1 if the selected approach combination includes approach a with a time limit of t seconds (called a sub-variation) ($a \in \mathcal{A}, t \in \mathcal{T}$);
- binary decision variables y_i taking value 1 if the selected approach combination covers (or solves) instance i ($i \in \mathcal{I}$).

The model is as follows:

$$\max \quad \sum_{i \in \mathcal{I}} y_i \tag{80}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}} \sum_{t \in \mathcal{T}} x_{at} \leq \beta, \tag{81}$$

$$y_i \leq \sum_{a \in \mathcal{A}} \sum_{t \in \mathcal{T}} s_{iat} x_{at} \qquad i \in \mathcal{I}, \tag{82}$$

$$\sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} t x_{at} \leq 3600, \tag{83}$$

$$y_i \in \{0, 1\} \qquad i \in \mathcal{I}, \tag{84}$$

$$x_{at} \in \{0,1\} \qquad a \in \mathcal{A}, t \in \mathcal{T}. \tag{85}$$

Objective function (80) maximizes the total number of instances solved, constraint (81) limits the number of sub-variations included in the approach combination, constraints (82) make sure that an instance is counted as covered if it is solved by at least one sub-variation in the approach combination, and constraint (83) limits the total time budget of the approach combination to 3600 seconds.

F Supplementary tables evaluating the impact of solver randomness

Table 26: Results of **FLOW-CBP** with normal patterns, integer variables, and RCVF for 1CBP instances

Dataset	#inst	**Seed-1**		**Seed-2**		**Seed-3**		**Seed-4**		**Seed-5**		**#solved by x seeds**					
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	5	4	3	2	1	0
CGCUT	3	2	1200	2	1200	2	1210	2	1200	2	1201	2	0	0	0	0	1
HT	9	9	3	9	66	9	18	9	10	9	5	9	0	0	0	0	0
BENG	10	2	2881	4	2564	3	2770	3	2520	2	2880	2	0	0	2	0	6
NGCUT	12	12	1	12	1	12	1	12	1	12	1	12	0	0	0	0	0
BKW	13	10	1172	7	1668	9	1259	10	1098	8	1527	6	2	2	0	0	3
GCUT	13	5	2216	5	2217	5	2217	5	2217	5	2217	5	0	0	0	0	8
CLASS1	50	34	1278	34	1211	35	1174	34	1221	35	1170	33	0	1	1	2	13
CLASS2	50	28	1780	25	1941	26	1788	27	1743	23	1969	22	0	1	4	8	15
CLASS3	50	19	2242	17	2383	18	2312	18	2337	17	2413	17	0	1	0	1	31
CLASS4	50	8	3080	9	3086	8	3081	7	3099	8	3093	7	1	0	0	1	41
CLASS5	50	21	2103	22	2033	21	2102	21	2102	21	2103	21	0	0	0	1	28
CLASS6	50	6	3185	7	3153	7	3216	6	3216	7	3193	6	0	1	0	0	43
CLASS7	50	47	217	47	217	47	217	47	217	48	215	47	0	0	0	1	2
CLASS8	50	0	3600	0	3600	0	3600	0	3600	0	3600	0	0	0	0	0	50
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS10	50	5	3241	5	3241	5	3258	5	3261	5	3241	5	0	0	0	0	45
Total	560	258	1987	255	2006	257	1989	256	1985	252	2020	244	3	6	7	14	286

Table 27: Results of **CP-CBP** with mirror symmetry-breaking constraints (63) for 1CBP instances

Dataset	#inst	**Seed-1**		**Seed-2**		**Seed-3**		**Seed-4**		**Seed-5**		**#solved by x seeds**					
		#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	#opt	T(s)	5	4	3	2	1	0
CGCUT	3	1	2400	1	2400	1	2400	1	2400	1	2400	1	0	0	0	0	2
HT	9	8	420	8	415	9	107	8	432	9	329	8	0	0	1	0	0
BENG	10	10	18	10	20	10	34	10	35	10	15	10	0	0	0	0	0
NGCUT	12	12	0	12	0	12	0	12	0	12	0	12	0	0	0	0	0
BKW	13	6	2015	6	2200	5	2239	5	2321	5	2250	5	0	0	1	0	7
GCUT	13	7	1786	7	1788	7	1711	7	1692	7	1701	7	0	0	0	0	6
CLASS1	50	50	34	50	28	50	52	50	12	49	101	49	1	0	0	0	0
CLASS2	50	47	246	48	175	48	192	46	304	47	222	46	0	1	1	1	1
CLASS3	50	36	1037	36	1051	36	1064	36	1051	36	1020	36	0	0	0	0	14
CLASS4	50	15	2611	15	2615	16	2510	20	2467	16	2534	14	0	1	4	1	30
CLASS5	50	35	1145	34	1157	34	1158	34	1156	35	1144	34	0	0	0	2	14
CLASS6	50	9	2970	10	2896	10	2927	9	3011	9	2998	8	1	1	0	0	40
CLASS7	50	50	0	50	0	50	1	50	2	50	1	50	0	0	0	0	0
CLASS8	50	11	2926	11	3020	11	2891	11	2965	10	2924	10	1	0	0	0	39
CLASS9	50	50	0	50	0	50	0	50	0	50	0	50	0	0	0	0	0
CLASS10	50	15	2551	15	2541	15	2562	15	2572	14	2595	14	1	0	0	0	35
Total	560	362	1315	363	1316	364	1299	364	1323	360	1319	354	4	3	7	4	188

G An SPP instance for which the 1CBP and the P|*cont*|C_{max} relaxations do not produce the same bound

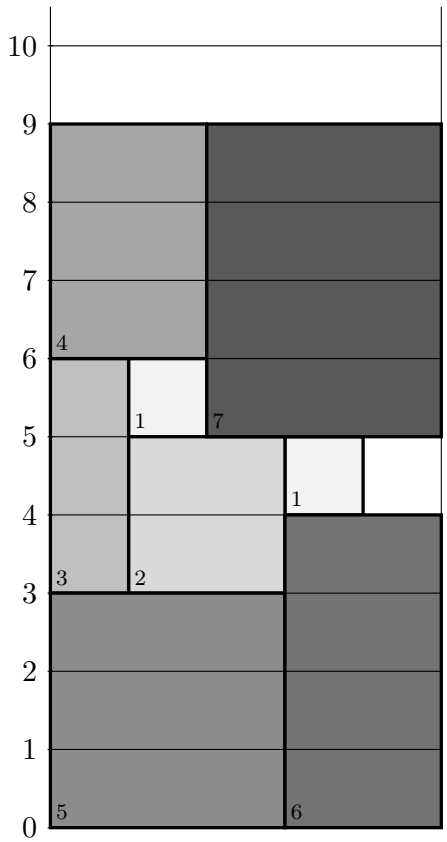
For a given SPP instance, the bound derived from its 1CBP relaxation is not necessarily the same as the bound derived from its P|*cont*|Cmax relaxation. For example, we provide in Figure 7a an SPP instance

for which the optimal solution of its 1CBP relaxation has value 9 (see Figure 7b) whereas the optimal solution of its P|*cont*|C_{max} relaxation has value 10 (see Figure 7c).

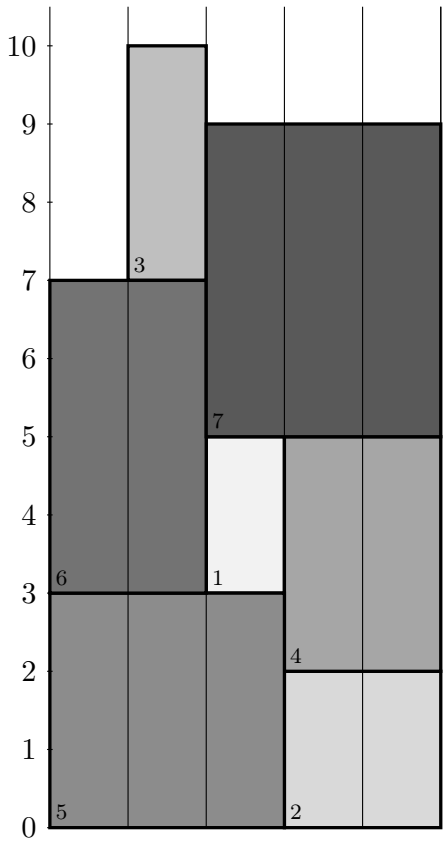
Figure 7: An SPP instance and the optimal solution of its 1CBP and P|*cont*|C_{max} relaxations

	w	h
strip	5	∞
item 1	1	2
item 2	2	2
item 3	1	3
item 4	2	3
item 5	3	3
item 6	2	4
item 7	3	4

(a) SPP instance



(b) Optimal 1CBP solution



(c) Optimal P|*cont*|C_{max} solution