A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems

Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

A New Branch-and-Price-and-Cut Algorithm for One-Dimensional Bin Packing Problems

Lijun Wei
Key Laboratory of Computer Integrated Manufacturing System, School of Electromechanical Engineering, Guangdong University of Technology, 510006, P.R. China, villagerwei@gmail.com

Zhixing Luo
School of Management and Engineering, Nanjing University, Nanjing 210093, P.R. China
Department of Industrial & Systems Engineering, National University of Singapore, Singapore, luozx.hkphd@gmail.com

Roberto Baldacci
Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna, 47521 Cesena, Italy, r.baldacci@unibo.it

Andrew Lim
Department of Industrial & Systems Engineering, National University of Singapore, Singapore
School of Management and Engineering, Nanjing University, Nanjing 210093, PR China, alim.china@gmail.com

In this paper, a new branch-and-price-and-cut algorithm is proposed to solve the one-dimensional bin packing problem (1D-BPP). The 1D-BPP is one of the most fundamental problems in combinatorial optimization and has been extensively studied for decades. Recently, Delorme et al. (2016) proposed 500 new test instances for the 1D-BPP; the best exact algorithm proposed in the literature can optimally solve 167 of these new instances, with a time limit of one hour imposed to each execution of the algorithm.

The exact algorithm proposed in this paper is based on the classical set-partitioning model for the 1D-BPP and the subset-row inequalities proposed by Jepsen et al. (2008). We describe an ad-hoc label-setting algorithm to solve the pricing problem, dominance and fathoming rules to speedup its computation and a new primal heuristic. The exact algorithm can easily handle some practical constraints, like the incompatibility between the items, and therefore we also apply it to solve the 1D-BPP with conflicts (1D-BPPC).

The proposed method is tested on a large family of 1D-BPP and 1D-BPPC classes of instances. For the 1D-BPP, the proposed method can optimally solve 237 instances of the new set of difficult instances; the largest instance involves 1003 items and bins of capacity 80,000. For the 1D-BPPC, the experiments show that the method is highly competitive with state-of-the-art methods, and successfully closed several open 1D-BPPC instances.

Key words: bin packing; bin packing with conflicts; branch-and-price-and-cut; exact algorithm
History:

1. Introduction

In this paper, we investigate the famous one-dimensional bin packing problem (1D-BPP) (Kantorovich 1960) and one of its classic variants, the one-dimensional bin packing problem with conflicts (1D-BPPC) (Gendreau et al. 2004).

Given a set $N = \{1, 2, \ldots, n\}$ of n items, each having an integer weight w_j, $j \in N$, and an unlimited number of identical bins of integer capacity c, the 1D-BPP requires to determine a minimum number of bins to pack all the items, subject to the constraint that the total weight of the items in a bin cannot exceed its capacity c. The 1D-BPPC extends the 1D-BPP by considering incompatibilities between pairs of items. That is, any pair of incompatible items cannot be packed into the same bin. In the 1D-BPPC, the incompatibilities of the items are usually given by an undirected conflict graph where a vertex represents an item and an edge between a pair of vertices represents the incompatibility of the pair of the vertices. Thus, the 1D-BPPC can be viewed as a combination of the 1D-BPP and the vertex coloring problem (Malaguti et al. 2008). Both the 1D-BPP and 1D-BPPC have attracted attentions from many researchers in the past few decades for their challenging combinatorial structures and wide-range applications in engineering, logistics and manufacturing (Johnson et al. 1974, Laporte and Desroches 1984, Jansen 1999).

The cutting stock problem (CSP) is another combinatorial optimization problem closely related to the 1D-BPP. Given an order for n types of rolls, each having an integer width w_j $(j = 1, \ldots, n)$ and an integer demand d_j $(j = 1, \ldots, n)$, and an unlimited number of larger rolls of integer width c, the objective of the CSP is to determine the minimum number of larger rolls cut into smaller widths to fulfill the order. The CSP originated from the steel industry (Wolfson (1965)), but it has then found applications in several other industries, such as paper and wood industries. According to the definition, the 1D-BPP is a special case of the CSP where the demand for each type of roll is fixed to 1. Therefore, solution approaches for the CSP can also be applied to solve the 1D-BPP.

In this paper, we propose a new branch-and-price-and-cut (BPC) algorithm for the 1D-BPP that can also solve the 1D-BPPC. Our BPC algorithm is based on the classic set-partitioning model of the 1D-BPP and the subset-row (SR) inequalities for strengthening the LP-relaxation of the set-partitioning formulation. The SR inequalities belong to famous Chvatal-Gomory rank-1 cuts and are valid for the general set-partitioning model. They were first proposed by Jepsen et al. (2008) for the vehicle routing problem with time windows

(VRPTW) to strengthen the LP-relaxation of the set-partitioning formulation. After that, the SR inequalities have been widely used in BPC algorithms for many vehicle routing problems (Archetti et al. 2011, Contardo and Martinelli 2014, Contardo et al. 2013) and other combinatorial optimization problems which can be formulated as set-partitioning models (Xue et al. 2015, Plum et al. 2014), since in many cases they substantially improve the lower bound yielded by the LP-relaxation of the set-partitioning formulation. However, to our knowledge, the SR inequalities have not been used to strengthen the set-partitioning model for the 1D-BPP, even though a number of branch-and-price (BP) algorithms have been proposed for the 1D-BPP or its variants in the literature.

1.1. Literature review

The literature on the 1D-BPP is vast and in this section we only present a review about exact algorithms for the 1D-BPP and the 1D-BPPC and, in particular, of the best exact methods for the 1D-BPP and the 1D-BPPC.

A comprehensive review on exact algorithms for the 1D-BPP can be found in Delorme et al. (2016). Exact algorithms for the 1D-BPP can be classified into branch-and-bound (BB) algorithms, constraint programming, pseudo-polynomial formulations based methods, and BP (or BPC) algorithms.

Eilon and Christofides (1971) proposed the first BB algorithm for the 1D-BPP, which utilizes lower bounds from the LP-relaxation of an assignment model and the general enumeration scheme proposed by Balas (1965). Martello and Toth (1990) improved the previous BB algorithm by several primal heuristics and reduction procedures. After that, a more powerful BB algorithm, referred to as the *BISON*, was proposed by Scholl et al. (1997) and was later improved by Schwerin and Wäscher (1998). BB algorithms have been also proposed by Mukhacheva et al. (2000), Korf (2002) and by Schreiber and Korf (2013).

Constraint programming solves the 1D-BPP through a set of pruning and propagation rules incorporating knapsack-based reasoning as well as lower bounds on the number of bins needed. Several researchers have successfully developed constraint programming based algorithms for the 1D-BPP, including Shaw (2004), Cambazard and O'Sullivan (2010), Dupuis et al. (2010) and Schaus et al. (2012).

Different pseudo-polynomial formulations have been proposed for the 1D-BPP. An effective CSP pseudo-polynomial formulation, denoted arc-flow, was presented by Valério de Carvalho, M. (1999). Very recently, Brandão and Pedroso (2016) proposed an arc-flow

formulation with side constraints that generalises the arc-flow formulation by Valério de Carvalho, M. (1999). Both Valério de Carvalho, M. (1999) and Brandão and Pedroso (2016) proposed exact methods for solving 1D-BPP and CSP problems.

BP (or BPC) algorithms have been the leading exact algorithms for the 1D-BPP in the past decade. Vance et al. (1994) proposed the first BP algorithm for the 1D-BPP, based on the traditional Gilmore-Gomory set-partitioning model for the CSP (Gilmore and Gomory 1961). At each node of the BB tree, the BP algorithm branches on a pair of items which are packed together by a fractional number of bins and forces the selected items to be packed either together or separately in the children nodes. Vance (1998) studied the influence of two branching schemes, one suggested in Barnhart et al. (1998) and the other in Vance et al. (1994), on BP algorithms and used the CSP as a case study. Valério de Carvalho, M. (2002) reviewed and analyzed LP models for the 1D-BPP and the CSP, and implemented several BP algorithms based on these LP models. Vanderbeck (1999) proposed a more efficient BP algorithm with several enhancement features such as variable fixing, cutting planes, early branching, and rounding heuristics. Degraeve and Schrage (1999) proposed a BP algorithm which branches on a column with fractional value for the CSP. This BP algorithm was improved later in Degraeve and Peeters (2003) by using heuristic algorithms, pruning rules, and a sub-gradient procedure to accelerate the convergence of column generation. Scheithauer et al. (2001) proposed a BPC algorithm with Chvátal-Gomory cuts (Chvátal 1973) to strengthen the LP-relaxation for the CSP. Belov and Scheithauer (2006) also proposed a BP algorithm for the CSP. The computational analysis reported in Delorme et al. (2016) shows that, among the different exact algorithms proposed for the 1D-BPP, the exact methods of Belov and Scheithauer (2006) and Brandão and Pedroso (2016) are the best ones.

The literature about exact algorithms for the 1D-BPPC is limited compared to the 1D-BPP literature. Fernandes Muritiba et al. (2010) proposed a hybrid multi-phase exact algorithm for the 1D-BPPC, whose core component is a BP algorithm that is applied in the last phase of the algorithm. The hybrid exact algorithm involves a population heuristic based on tabu search to find high-quality feasible solutions and several bounding procedures to tighten the lower bounds. Only if the bounding procedures cannot prove the optimality of the solution obtained by the heuristic, the BP is executed to find the final

optimal solution. Elhedhli et al. (2011) proposed another BP algorithm with two key features that greatly contribute to its efficiency. First, a special branching scheme based on the conflicting constraints, which is able to preserve the structure of the pricing problem after branching. Secondly, maximal clique inequalities that are generated according to the conflicting constraints and are added to the pricing problems. Sadykov and Vanderbeck (2013) developed a generic BP algorithm relying on generic branching schemes and primal heuristics. Depending on the structure of the conflict graph, the BP algorithm either uses a dynamic programming or a depth-first BB algorithm to solve the pricing problem. The computational results reported by Sadykov and Vanderbeck (2013) show that their BP algorithm outperforms the existing algorithms on instances from the literature. Brandão and Pedroso (2016) also presented the results obtained using their arc-flow formulation for several cutting and packing problems, including the 1D-BPPC.

1.2. Contributions of this paper

Our distinct contributions in this paper are as follows:

- We propose a new BPC algorithm to solve the 1D-BPP and the 1D-BPPC based on the set-partitioning model with SR inequalities.
- We describe a new label-setting algorithm to solve the pricing problem associated with the mathematical formulation and dominance and fathoming rules used to speed up its computation.
- We perform extensive computational experiments on both 1D-BPP and 1D-BPPC classes of instances proposed in the literature. In particular, for the 1D-BPP, we compare our results with state-of-the-art methods for the CSP that also represent the best exact methods for the 1D-BPP.

The results show that the SR inequalities can improve the lower bounds yielded by the LP-relaxation of the set-partitioning formulation and that the new BPC is highly competitive with state-of-the-art exact methods. For both the 1D-BPP and the 1D-BPPC the proposed method successfully closed several open instances.

The remainder of the paper is organized as follows. In Section 2, we introduce the mathematical formulation for the 1D-BPP and the 1D-BPPC together with a description of the corresponding pricing problems. The details of the BPC algorithm, including a label-setting algorithm used to solve the pricing problems, dominance and fathoming rules, a

primal heuristic and a branching strategy, are given in Section 3. Section 4 presents and analyses the computational results of the BPC algorithm on several classes of 1D-BPP and 1D-BPPC instances proposed in the literature. Finally, we conclude the paper and indicate future research directions in Section 5.

2. Set-partitioning formulation with SR inequalities

In this section, we introduce a set-partitioning formulation with SR inequalities for the 1D-BPP and the 1D-BPPC.

A *pattern* S is a subset of the item set N such that $\sum_{j\in S} w_j \leq c$. In the case of the 1D-BPPC, the set of items is also characterized by a conflict graph $G=(N,E)$, where E is a set of edges such that $\{i,j\}\in E$ when i and j are in *conflict*. For the 1D-BPPC, a pattern S cannot contain any pair of items i, j, $i\neq j$, if $\{i,j\}\in E$. Let $\mathscr{P}$ be the index set of all patterns for the 1D-BPP (1D-BPPC), and let N_p be the set of items of pattern $p\in\mathscr{P}$. We denote with a_{ip}, $i\in N$, $p\in\mathscr{P}$, a (0-1) coefficient equal to 1 if item $i\in N_p$, 0 otherwise.

Let $\mathscr{C}\subseteq\{S\subset N:|S|=3\}$ be a subset of all items triplets, and let $\mathscr{P}(S)\subseteq\mathscr{P}$ be the subset of the index set of all patterns containing at least two items in S (i.e.,$\mathscr{P}(S)=\{p\in\mathscr{P}:|N_p\cap S|\geq 2\}$). Let x_p, $p\in\mathscr{P}$, be a binary variable equal to 1 if and only if pattern p is in the optimal solution. The 1D-BPP (1D-BPPC) formulation based on the set-partitioning model and SR inequalities, hereafter called F, is

$$(F)\quad z(F)=\min\ \sum_{p\in\mathscr{P}} x_p \tag{1}$$

$$\text{s.t.}\sum_{p\in\mathscr{P}} a_{ip}x_p=1,\quad \forall i\in N \tag{2}$$

$$\sum_{p\in\mathscr{P}(S)} x_p\leq 1,\quad \forall S\in\mathscr{C} \tag{3}$$

$$x_p\in\{0,1\},\quad \forall p\in\mathscr{P}.$$

The objective (1) states to minimize the number of bins. Constraints (2) ensure that each item $i\in N$ has to be assigned to exactly one bin. Constraints (3) correspond to a subset of SR inequalities involving sets of items having cardinality equal to three.

Let LF be LP-relaxation of formulation F and let LF' be the LP-relaxation of formulation F without SR inequalities (3), denoted as formulation F'. We denote by $z(LF)$ and

by $z(LF')$ the optimal solution costs of problems LF and LF', respectively. A 1D-BPP instance is said to have the Integer Round-Up Property (IRUP) if the rounded up value of $z(LF')$ (i.e., $\lceil z(LF') \rceil$) is equal to $z(F')$. It was conjectured in the seventies that the IRUP held for all the 1D-BPP instances. However, this conjecture was disproved after Non-IRUP instances were discovered by Marcotte (1986). Scheithauer and Terno (1995, 1997) conjectured that a difference equal to one between the rounded up lower bound and the optimal solution holds for any 1D-BPP and CSP instance (Modified Integer Round-Up Property, MIRUP). The MIRUP conjecture is still open both for the 1D-BPP and the CSP, but a number of interesting results have been obtained while attempting to close it. In particular, Caprara et al. (2015) produced a large set of Non-IRUP instances by using a relationship between the 1D-BPP and the edge coloring problem. They also gave a method to transform an IRUP instance into a Non-IRUP one. The IRUP for the CSP has also been investigated by Kartak et al. (2015). The MIRUP does not hold for the Vertex Coloring Problem (see, for example, Malaguti et al. 2011), a special case of the 1D-BPPC, and therefore it does not hold also for the 1D-BPPC.

Relaxations LF' and LF can be solved by column generation by iteratively solving a *restricted master problem* (RMP) and the *pricing problem*, that determines whether there exists a variable $p \in \mathscr{P}$ to be added to RMP to improve its current solution. In the case of the 1D-BPP, the pricing problem associated with relaxation LF' is the well-known Knapsack Problem (KP) whereas in the case of the 1D-BPPC the pricing problem is a Knapsack Problem with Conflicts (KPC). SR inequalities can strengthen the lower bound obtained from LF' but the complexity of the corresponding pricing problem (KP or KPC) is sensitive to the addition of those cuts, since the values of the corresponding dual variables cannot be translated into subproblem costs.

2.1. The pricing problems

In this section, we describe the mathematical formulations associated with the pricing problems of formulation LF. In Section 3.1, we then describe an efficient algorithm for their solution based on dynamic programming.

The variables of the dual of problem LF are given by the vectors $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$ and $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_{|\mathscr{C}|})$, where $\mu_1, \mu_2, \ldots, \mu_n$ are associated with constraints (2), and $\boldsymbol{\lambda} \leq 0$

with constraints (3). The reduced cost of a pattern $p \in \mathscr{P}$ with respect to a dual solution $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ can be computed as

$$1 - \sum_{i \in N} a_{ip}\mu_i - \sum_{\substack{S \in \mathscr{C} \, s.t. \\ |N_p \cap S| \geq 2}} \lambda_S.$$

Let y_i, $i \in N$, be a binary variable equal to 1 if and only if item i is packed into the optimal pattern, and let z_S, $S \in \mathscr{C}$, be a binary variable equal to 1 if and only if the dual value λ_S of the SR inequality associated with set S is subtracted from the reduced cost of the optimal pattern. The pricing problem for the 1D-BPP is formulated as

$$(SP1) \quad z(SP1) = \min \; 1 - \sum_{i \in N} \mu_i y_i - \sum_{S \in \mathscr{C}} \lambda_S z_S \tag{4}$$

$$\text{s.t.} \sum_{i \in N} y_i w_i \leq c \tag{5}$$

$$z_S \geq y_i + y_j - 1, \quad \forall i, j \in S, i \neq j, \forall S \in \mathscr{C} \tag{6}$$

$$y_i \in \{0, 1\}, \quad \forall i \in N \tag{7}$$

$$z_S \in \{0, 1\}, \quad \forall S \in \mathscr{C}. \tag{8}$$

The objective (4) states to minimize the reduced cost of the pattern. Constraints (5) ensure that the total weight of the items in a pattern cannot exceed the capacity whereas constraints (6) are linking constraints between variables $\mathbf{y}$ and $\mathbf{z}$.

The formulation of the pricing problem associated with the 1D-BBPC can be derived from formulation $SP1$ as

$$(SP2) \quad z(SP2) = \min \; 1 - \sum_{i \in N} \mu_i y_i - \sum_{S \in \mathscr{C}} \lambda_S z_S$$

$$\text{s.t.} (5), (6), (7) \text{ and } (8)$$

$$y_i + y_j \leq 1, \quad \forall \{i, j\} \in E. \tag{9}$$

where the additional set of constraints (9) avoid joint assignments of items that are in conflict.

Section 3.1 describes a dynamic programming label-setting algorithm to solve the problems associated with formulations $SP1$ and $SP2$, and dominance and fathoming rules used to speed up the computation.

3. Branch-and-Price-and-Cut algorithm

In this section, we describe an exact algorithm for solving the 1D-BBP and the 1D-BBPC based on a BPC method.

The master problem at the root node of the enumeration tree is defined as the LP-relaxation of formulation F where the equality constraints (2) are relaxed to covering constraints and the set of patterns $\mathscr{P}$ and the set of SR inequalities $\mathscr{C}$ are substituted by sets $\overline{\mathscr{P}}$ and $\overline{\mathscr{C}}$, respectively. Set $\overline{\mathscr{P}}$ is defined as the set of single-item patterns, i.e., $\overline{\mathscr{P}} = \{1, 2, \ldots, n\}$, where $N_j = \{j\}$, $\forall j \in N$, and set $\overline{\mathscr{C}}$ is set to the empty set, i.e., $\overline{\mathscr{C}} = \emptyset$. Set $\overline{\mathscr{P}}$ also ensures that the master problem admits a feasible solution. SR inequalities (3) are separated by complete enumeration.

In the following section, we describe a label-setting algorithm used to solve the pricing problems described in Section 2.1 together with dominance and fathoming rules. Section 3.2 describes the primal heuristic used to find good feasible solutions whereas Section 3.3 delineates the branching scheme adopted in the algorithm.

3.1. Pricing algorithm

In this section, we describe an algorithm to solve the pricing problem $SP2$ associated with the 1D-BPPC (and thus the pricing problem $SP1$ associated with the 1D-BPP). The structure of the algorithm is such that branching rules based on whether a pair of items are required or forbidden to be packed in a same bin can be easily handled by the algorithm.

The pricing algorithm is based on the classic dynamic programming algorithm for the KP (see Martello and Toth 1990). In our case, the dynamic programming recursion needs to include additional resources to model the contributions of the dual variables associated with the SR inequalities to the computation of the reduced cost on a pattern.

For sake of description, we assume that a dummy item $n+1$, that is compatible with all the items of set N, is included in set N, i.e., $N = N \cup \{n+1\}$. Let $(\boldsymbol{\mu}, \boldsymbol{\lambda})$, $\boldsymbol{\mu} \geq 0$ and $\boldsymbol{\lambda} \leq 0$, be the current dual solution of the master problem.

A label is a tuple $L = (j, l, \overline{c}, V, O, \mathcal{R})$ and is composed of the following information:

- j: the index of the last item considered in the partial pattern;
- l: the total weight of the items in the partial pattern;
- $\overline{c}$: the reduced cost of the partial pattern;
- V: the set of items packed in the partial pattern, $V \subseteq \{1, 2, \ldots, j\}$;

- O: the set of items in $\{j+1,\ldots,n,n+1\}$ compatible with the items in V and such that $l+w_i \leq c$, $\forall i \in O$;
- $\mathcal{R}$: the set of binary resources associated with the binding SR inequalities in set $\overline{\mathscr{C}}$. For each SR inequality associated to a item subset S we consider one binary integer resource $\sigma : \mathscr{P} \to \{0,1\}$. We denote by $S(\sigma)$ and by $\lambda(\sigma)$ the item subset defining the SR inequality and the dual variable associated, respectively.

Let $F_j \subset N$ be the set of items incompatible with item j. The label extension rule for a label $L = (j, l, \overline{c}, V, O, \mathcal{R})$ is as follows. Let i be the item having the minimum index among the items in set O, i.e., $i = \arg\min\{h : h \in O(L)\}$. If $i = n+1$, then a single label $L = (n+1, l, \overline{c}, V, O, \mathcal{R})$ corresponding to the complete pattern $V(L)$ is created. Otherwise, two new labels L_1 and L_2 are created according to whether item i is packed into the partial pattern associated with L or not, respectively. The label L_1 is

$$
\begin{aligned}
j(L_1) &= i \\
l(L_1) &= l(L) + w_i \\
\overline{c}(L_1) &= \overline{c}(L) - \mu_i - \sum_{\substack{\sigma \in \mathcal{R} \text{ s.t.} \\ \sigma(L)=1,\, i \in S(\sigma)}} \lambda(\sigma) \\
V(L_1) &= V(L) \cup \{i\} \\
O(L_1) &= \{h : h \in O(L) \setminus F_i \setminus \{i\}, l(L) + w_i + w_h \leq c\} \\
\sigma(L_1) &= \begin{cases} \mathrm{mod}(\sigma(L)+1, 2), \text{ if } i \in S(\sigma) \\ \sigma(L), \text{ otherwise} \end{cases} \qquad \forall \sigma \in \mathcal{R},
\end{aligned}
$$

where the function $\mathrm{mod}(x, y)$ returns the remainder of dividing x by y, and the label L_2 is

$$
\begin{aligned}
j(L_2) &= i \\
l(L_2) &= l(L) \\
\overline{c}(L_2) &= \overline{c}(L) \\
V(L_2) &= V(L) \\
O(L_2) &= O(L) \setminus \{i\} \\
\sigma(L_2) &= \sigma(L), \forall \sigma \in \mathcal{R}.
\end{aligned}
$$

The algorithm starts from the initial label $(0, 0, 1.0, \emptyset, \{1, 2, \ldots, n+1\}, \{0\}_{\sigma \in \mathcal{R}})$.

To reduce the number of labels to extend, the following dominance rule is proposed to identify labels that can be safely discarded. Dominance rules for the VRPTW based on the SR inequalities have been investigated by Jepsen et al. (2008).

For a label L, let $\mathscr{P}(L) \subseteq \mathscr{P}$ be the index set of all feasible partial patterns with items in $O(L)$ that can be used to complete the partial pattern $V(L)$. Let $(V(L), N_p)$, $p \in \mathscr{P}(L)$, be a feasible pattern obtained by joining partial pattern $V(L)$ with a pattern N_p, $p \in \mathscr{P}(L)$, and let $\overline{c}(V(L), N_p)$ be the corresponding reduced cost computed according to the dual solution $(\boldsymbol{\mu}, \boldsymbol{\lambda})$.

DOMINANCE 1. Let L_1 and L_2 be two labels with $j(L_1) = j(L_2) = j$ and such that:

$$\left.\begin{array}{ll} \overline{c}(L_1) - \displaystyle\sum_{\substack{\sigma \in \mathcal{R}\ s.t. \\ \sigma(L_1)=1, \sigma(L_2)=0}} \lambda(\sigma) \leq \overline{c}(L_2) - \sum_{i \in O(L_2) \setminus O(L_1)} \mu_i & \text{(a)} \\ l(L_1) \leq l(L_2), & \text{(b)} \end{array}\right\} \tag{10}$$

then label L_1 dominates label L_2.

Proof. For any partial pattern $p_2 \in \mathscr{P}(L_2)$, we first construct another partial pattern N_{p_1} which consists of all the items in $N_{p_2} \cap O(L_1)$. From the definition of pattern p_1 and due to inequality (10)-b, we have $p_1 \in \mathscr{P}(L_1)$. The difference between the reduced cost of pattern $(V(L_1), N_{p_1})$ and $(V(L_2), N_{p_2})$ is

$$\begin{aligned} &\overline{c}(V(L_1), N_{p_1}) - \overline{c}(V(L_2), N_{p_2}) = \\ &\overline{c}(L_1) - \sum_{i \in N_{p_1}} \mu_i - \sum_{\substack{\sigma \in \mathcal{R}\ s.t \\ \sigma(L_1)=1, |N_{p_1} \cap S(\sigma)|=1}} \lambda(\sigma) - \sum_{\substack{\sigma \in \mathcal{R}\ s.t \\ |N_{p_1} \cap S(\sigma)| \geq 2}} \lambda(\sigma) - \\ &\overline{c}(L_2) + \sum_{i \in N_{p_2}} \mu_i + \sum_{\substack{\sigma \in \mathcal{R}\ s.t \\ \sigma(L_2)=1, |N_{p_2} \cap S(\sigma)|=1}} \lambda(\sigma) + \sum_{\substack{\sigma \in \mathcal{R}\ s.t \\ |N_{p_2} \cap S(\sigma)| \geq 2}} \lambda(\sigma). \end{aligned} \tag{11}$$

We have:

(i) $N_{p_2} \setminus N_{p_1} = N_{p_2} \setminus (N_{p_2} \cap O(L_1)) = N_{p_2} \setminus O(L_1)$.

(ii)
$$\begin{aligned} &\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, |N_{p_1} \cap S(\sigma)| = 1\} = \\ &\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 0, |N_{p_1} \cap S(\sigma)| = 1\} \cup \\ &\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 1, |N_{p_1} \cap S(\sigma)| = 1\} = \\ &(\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 0\} \setminus \{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 0, |N_{p_1} \cap S(\sigma)| \neq 1\}) \cup \\ &\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 1, |N_{p_1} \cap S(\sigma)| = 1\}. \end{aligned}$$

(iii)
$$\{\sigma \in \mathcal{R} : \sigma(L_2) = 1, |N_{p_2} \cap S(\sigma)| = 1\} =$$
$$\{\sigma \in \mathcal{R} : \sigma(L_1) = 0, \sigma(L_2) = 1, |N_{p_2} \cap S(\sigma)| = 1\} \cup$$
$$\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 1, |N_{p_2} \cap S(\sigma)| = 1\} =$$
$$\{\sigma \in \mathcal{R} : \sigma(L_1) = 0, \sigma(L_2) = 1, |N_{p_2} \cap S(\sigma)| = 1\} \cup$$
$$\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 1, |N_{p_1} \cap S(\sigma)| = 1\} \cup$$
$$\{\sigma \in \mathcal{R} : \sigma(L_1) = 1, \sigma(L_2) = 1, |(N_{p_2} \setminus N_{p_1}) \cap S(\sigma)| = 1\}.$$

(iv)
$$\{\sigma \in \mathcal{R} : |N_{p_2} \cap S(\sigma)| \geq 2\} =$$
$$\{\sigma \in \mathcal{R} : |N_{p_1} \cap S(\sigma)| \geq 2\} \cup \{\sigma \in \mathcal{R} : |N_{p_2} \cap S(\sigma)| \geq 2, |N_{p_1} \cap S(\sigma)| \leq 1\}.$$

Equation (11) then becomes:

$$\begin{aligned} \overline{c}(V(L_1), N_{p_1}) - \overline{c}(V(L_2), N_{p_2}) = \\ \overline{c}(L_1) - \overline{c}(L_2) + \sum_{i \in N_{p_2} \setminus O(L_1)} \mu_i - \sum_{\substack{\sigma \in \mathcal{R} \ s.t. \\ \sigma(L_1)=1, \sigma(L_2)=0}} \lambda(S) + \sum_{\substack{\sigma \in \mathcal{R} \ s.t. \\ \sigma(L_1)=1, \sigma(L_2)=0, |N_{p_1} \cap S(\sigma)| \neq 1}} \lambda(S) + \\ \sum_{\substack{\sigma \in \mathcal{R} \ s.t. \\ \sigma(L_1)=0, \sigma(L_2)=1, |N_{p_2} \cap S(\sigma)|=1}} \lambda(S) + \sum_{\substack{\sigma \in \mathcal{R} \ s.t. \\ \sigma(L_1)=1, \sigma(L_2)=1, |(N_{p_2} \setminus N_{p_1}) \cap S(\sigma)|=1}} \lambda(S) + \\ \sum_{\substack{\sigma \in \mathcal{R} \ s.t. \\ |N_{p_2} \cap S(\sigma)| \geq 2, |N_{p_1} \cap S(\sigma)| \leq 1}} \lambda(S). \end{aligned} \tag{12}$$

Since $\boldsymbol{\mu} \geq 0$, $\boldsymbol{\lambda} \leq 0$ and $N_{p_2} \subseteq O(L_2)$, from equation (12) we obtain:

$$\overline{c}(V(L_1), N_{p_1}) - \overline{c}(V(L_2), N_{p_2}) \leq \overline{c}(L_1) - \overline{c}(L_2) + \sum_{j \in O(L_2) \setminus O(L_1)} \mu_i - \sum_{\substack{\sigma \in \mathcal{R} \ s.t \\ \sigma(L_1)=1, \sigma(L_2)=0}} \lambda(\sigma) \leq 0,$$

and the last inequality holds due to inequality (10)-a.□

Condition (10)-b ensures that the feasible extensions of L_1 about the bin capacity are also feasible extensions for L_2. Inequality (10)-b aims at considering all resources from which a common extension of L_1 and L_2 would result in increasing the reduced cost of L_1 without increasing that of L_2. The inequality thus represents the impossibility for the reduced cost of L_1 to exceed that of L_2. It follows that L_2 can be safely discarded as it will never produce patterns better than those that can be produced from extending L_1.

The number of labels can be further reduced by removing any label L that cannot lead to any negative reduced cost pattern according to the following rule.

FATHOMING 1. Let $lb(L)$ be a lower bound on the reduced cost of any pattern $p \in \mathscr{P}(L)$. Any label L such that

$$\overline{c}(L) + lb(L) \geq 0$$

cannot lead to any negative reduced cost pattern.

Lower bound $lb(L)$ can be computed by ignoring the conflicting constraints and the dual variables associated with the SR inequalities as follows:

$$\begin{aligned} lb(L) = \min & \sum_{i=j(L)+1}^{n} -\mu_i z_i \\ \text{s.t.} & \sum_{i=j(L)+1}^{n} w_i z_i \leq c - l(L) \\ & z_i \in \{0,1\}, \forall i \in \{j(L)+1, \ldots, n\}, \end{aligned}$$

where $z_i \in \{0,1\}$, $\forall i \in \{j(L)+1, \ldots, n\}$, is a binary variable taking value 1 if item i is selected in solution and 0 otherwise.

Algorithm 1 illustrates the dynamic programming algorithm used to solve the pricing problem. In the algorithm, set $\mathcal{L}_j$, $j = 0, 1, \ldots, n$, represents the set of all labels at stage i of the algorithm. The algorithm returns a subset $\overline{\mathcal{L}}_{n+1}$ of set $\mathcal{L}_{n+1}$ of at most Δ negative reduced cost patterns with respect to the dual solution $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ and such that $\max_{L \in \overline{\mathcal{L}}_{n+1}} \overline{c}(L) \leq \min_{L \in \mathcal{L}_{n+1} \setminus \overline{\mathcal{L}}_{n+1}} \overline{c}(L)$, where Δ is a parameter defined a priori.

Implementation issues Each set $\mathcal{L}_i$ is stored using an array of data structures, where each data structure corresponds to a state or label with the corresponding information. Every time a new label is generated, its data are appended at the end of the corresponding array. In order to reduce the space requirements, the different sets of a label are encoded by bit strings, i.e., we use a *bitmask representation* where every single bit of a computer word is used separately to indicate whether an item is included in the set or not and bitset operations are used to implement the basic set operations of intersection and union.

During the execution of preliminary experiments, we observed that the Dominance rule 1 applied at Step 4 of Algorithm 1 can be time consuming, therefore, to speed up the whole solution process, we first sort the labels in set $\mathcal{L}_i$ for increasing values of the reduced costs. Then, only the pair of consecutive labels are compared to check the dominance rule. Although not all the dominated labels are removed under this strategy, we found it to be

Algorithm 1 *Algorithm for the solution of the pricing problem*

$L_0 \leftarrow (0, 0, 1.0, \emptyset, \{1, 2, \ldots, n+1\}, \{0\}_{\sigma \in \mathcal{R}})$, $\mathcal{L}_0 \leftarrow \{L_0\}$.
$\mathcal{L}_j \leftarrow \emptyset$, $j = 1, 2, \ldots, n+1$.
for all $j = 0, \ldots, n$ **do**
    Apply Dominance rule 1 and eventually delete labels in $\mathcal{L}_j$.
    for all $L \in \mathcal{L}_j$ **do**
        $i \leftarrow \arg\min\{h : h \in O(L)\}$.
        if $i = n+1$ **then**
            $\mathcal{L}_{n+1} \leftarrow \mathcal{L}_{n+1} \cup \{L\}$.
        else
            Extend L to create two new labels L_1 and L_2.
            if $\overline{c}(L_1) + lb(L_1) < 0$ ($\overline{c}(L_2) + lb(L_2) < 0$) **then**
                $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{L_1\}$ ($\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{L_2\}$).
            end if
        end if
    end for
end for
return Set $\overline{\mathcal{L}}_{n+1} \subseteq \mathcal{L}_{n+1}$ s. t. $|\overline{\mathcal{L}}_{n+1}| \leq \Delta$, $\max_{L \in \overline{\mathcal{L}}_{n+1}} \overline{c}(L) < 0$, $\max_{L \in \overline{\mathcal{L}}_{n+1}} \overline{c}(L) \leq \min_{L \in \mathcal{L}_{n+1} \setminus \overline{\mathcal{L}}_{n+1}} \overline{c}(L)$.

computationally convenient. We also found to be computationally convenient to directly sort the labels of each set $\mathcal{L}_i$ since it requires $O(|\mathcal{L}_i| \log |\mathcal{L}_i|)$ operations with respect to the option of maintaining an order list of labels, that requires $O(|\mathcal{L}_i|^2)$ operations.

The computation of function $lb(L)$ has been implemented as follows. Let $g(i, q)$ be the optimal solution cost of the KP defined by the items in $\{i, \ldots, n\}$ and bin capacity equal q. Then $lb(L)$ is computed as $lb(L) = g(j(L) + 1, c - l(L))$; functions $g(i, q)$ are computed before starting Algorithm 1 using a standard dynamic programming implementation (see Martello and Toth 1990). To further sped up the computation, during the execution of Algorithm 1 we also compute the minimum reduced cost among all the reduced costs of the labels generated so far, say $\overline{c}_{min}$ - each label L such that $\overline{c}(L) + lb(L) \geq \overline{c}_{min}$ can be discarded. In the computational results reported in Section 4, parameter Δ has been set equal to 10.

3.2. Primal heuristic

As shown by previous works (see, for example, Sadykov and Vanderbeck 2013), primal heuristics can help to improve the performance significantly. Sadykov and Vanderbeck (2013) used a generic diving heuristic that is a depth-first heuristic search in the BP tree,

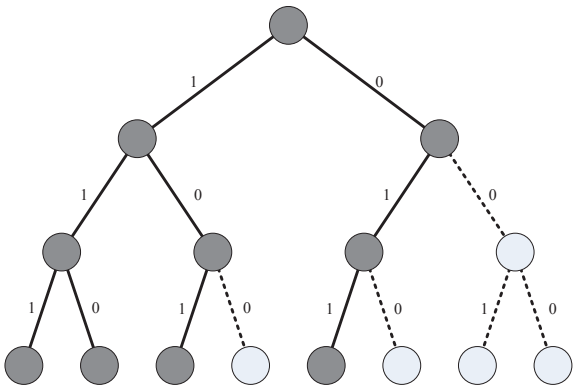

Figure 1 The search tree of the primal heuristic

based on the heuristic proposed in Joncour et al. (2010). The solution obtained through the initial depth-first exploration of the tree is considered as a reference incumbent solution and it is further explored using limited backtracking as a diversification mechanism and limited discrepancy search (Joncour et al. (2010)).

Similarly to Sadykov and Vanderbeck (2013), we designed a primal heuristic based on a BP tree, where we adopted different branching and search strategies. The BP enumeration is driven by a binary branching by fixing x_p variables either to 1 (*left branch*) or to 0 (*right branch*). At each BP node, the master is solved by column generation, then two branches are generated by using a greedy strategy. In a left branch, the master is updated by deleting rows associated to items already covered and deleting columns covering those items. In a right branch, the column associated with the selected variable is deleted from the master. When the master is reoptimized, the columns that have been deleted due to right branches are not added to the master problem even if they are regenerated during the solution of the pricing problem.

The tree is explored using a modified depth-first strategy. We restrict our search on a limited number of nodes by choosing at most one right child node in the path from the root node to any visited node. More precisely, suppose $(s_1, ..., s_i)$ be a node sequence where s_1 is the root node and node s_{j-1} is the father of node s_j, $j = 2, \ldots, i$. Then if s_j is the left child node of s_{j-1}, $j = 2, \ldots, i$, then the next node explored by the algorithm, i.e. node s_{i+1}, is the right child of s_i; otherwise, s_{i+1} can only be the left child node of s_i and the right child is discarded. Figure 1 illustrates the search strategy; in the figure, the nodes which are explored are darker than the nodes that are discarded.

During the execution of preliminary experiments, we found to be computationally convenient to apply the primal heuristic only at the root node of the BPC tree.

3.3. Branching strategy

According to the observation of Ryan and Foster (1981), when an optimal solution of the restrict master problem is fractional, there must exist a pair of item (i, j) such that $0 < \delta_{ij} < 1$, where $\delta_{ij} = \sum_{p \in \mathscr{P}} a_{ip} a_{jp} x_p$. Variable δ_{ij} can be interpreted as an auxiliary (0-1) variable indicating whether or not items i and j are packed in the same pattern.

Therefore, a natural branching scheme is the following. We first select the pair (i, j) whose δ_{ij} value is as close as possible to 0.5 - ties are broken by selecting the pair with maximum weight $w_i + w_j$. Then we branch by enforcing the selected pair of items being packed into the same bin or in different bins, i.e., $\delta_{ij} = 1$ or $\delta_{ij} = 0$, respectively.

The first child node, where item i and j must be assigned to the same bin, is implemented by setting the weight of item i equal to $w_i + w_j$ and by deleting item j, i.e., $N = N \setminus \{j\}$. The conflict graph $G = (V, E)$ is updated by setting $V = V \setminus \{j\}$ and by re-connecting all edges $\{j, k\} \in E$ to vertex i. The master is updated by deleting row j and by removing all columns p such that $a_{ip} = 1$ and $a_{jp} = 0$ or $a_{ip} = 0$ and $a_{jp} = 1$.

The second child, where item i and j must be assigned to different bins, is easily implemented by adding an incompatibility between item i and j, i.e., by setting $E = E \cup \{\{i, j\}\}$. All columns p such that $a_{ip} + a_{jp} > 1$ are removed from the master.

An important property of the above branching rule is that it preserves the structure of our pricing problem. It is worth mentioning that alternative branching schemes, that also preserve the structure of the pricing problem and that take a form closely related to the Ryan and Foster scheme, have been proposed by Vanderbeck (2011).

4. Computational results

The BPC algorithm described in Section 3 (hereafter called "EXM") was coded in Java using Sun Oracle JDK 1.7.0. The experiments were carried out on an Intel Xeon E5-1603 equipped with a 2.80 GHz (Quad Core) CPU and 8GB of RAM, running under Ubuntu 4 Linux operating system. The LP-relaxation of the restricted master problem was solved using the primal simplex optimizer of ILOG CPLEX 12.6.3 (IBM CPLEX 2016).

The source code of EXM and its results can be downloaded at the website `http://www.computational-logistics.org/orlib/`, section "1D Bin Packing Problems".

Table 1 1D-BPP and 1D-BPPC classes of instances proposed in literature

Problem	Class	$\#Inst$	Source	n	c
1D-BPP	Falkenauer U	80	Falkenauer (1996)	120-1000	150
	Falkenauer T	80		60-501	1000
	Scholl 1	720	Scholl et al. (1997)	50-500	100-150
	Scholl 2	480		50-500	1000
	Scholl 3	10		50-500	10,000
	Wäscher	17	Wäscher and Gau (1996)	57-239	10,000
	Schwerin 1	100	Schwerin and Wäscher (1998)	100	1000
	Schwerin 2	100		120	1000
	Hard28	28	Schoenfield (2002)	160-200	1000
	Random	3840	Delorme et al. (2016)	50-1000	50-1000
	AI	250		201-1002	2500-80,000
	ANI	250		201-1002	2500-80,000
1D-BPPC	t-FMIMT	540	Fernandes Muritiba et al. (2010)	60-501	1000
	u-FMIMT	540		120-1000	150
	t-ELGN	360	Elhedhli et al. (2011)	60-249	1000
	u-ELGN	360		120-500	150
	ta	360	Sadykov and Vanderbeck (2013)	60-501	1000
	ua	360		120-1000	150
	da	270		120-500	10,000
	d	270		120-500	10,000

4.1. Classes of instances

Test instances for the 1D-BPP have been proposed in the literature by different authors. Recently, Delorme et al. (2016) have generated two new classes of difficult instances. The first class of instances, called the *augmented Non-IRUP* (ANI) class, is characterized by instances without the IRUP property. The second class, called *augmented IRUP* (AI) class, has been derived from the ANI class by ensuring that the bins used in an optimal solution are always completely filled. It is worth mentioning that the optimal solutions of these two classes of instances are known based on the method used to generate them. The reader is referred to Delorme et al. (2016) for further details about the ANI and AI classes.

Table 1 summarizes the classes of 1D-BPP instances proposed in the literature and used in our experiments. In the table, for each class of instances, columns "n" and "c" give the ranges of the number of items and the capacities of the bins, respectively; column "#Inst" reports the number of instances of the corresponding class. The instances can be downloaded from `http://or.dei.unibo.it/library/bpplib` (see Delorme et al. 2017).

For the 1D-BPPC, instances were generated by Fernandes Muritiba et al. (2010), Elhedhli et al. (2011) and Sadykov and Vanderbeck (2013). The details of these classes of instances are also given in Table 1. In particular, in classes "t", "u" and "d", conflict graphs are interval graphs (see Sadykov and Vanderbeck (2013)). The instances are available at `https://www.math.u-bordeaux.fr/~rsadykov/tests/BPPC_test_instances.zip`.

4.2. Computational results on the 1D-BPP

In this section, we compare algorithm EXM with the following two exact methods that, according to Delorme et al. (2016), are accounted to be the best exact algorithms for the 1D-BPP:

- BELOV: the BP algorithm of Belov and Scheithauer (2006);
- VPSOLVER: the exact method of Brandão and Pedroso (2016).

It is worth mentioning that both BELOV and VPSOLVER are dedicated to the CSP. The computational results reported by Delorme et al. (2016) are used as a basis for comparing the results of EXM. Algorithms BELOV and VPSOLVER have been tested by Delorme et al. on an Intel Xeon 3.10 GHz with 8 GB RAM. According to the SuperPi (1M) benchmark (`http://www.superpi.net/`), the computing time in seconds of the machine used by Delorme et al. is equal to 9.97, and it is close to the computing time of our machine which is equal to 9.70. The computing time of SuperPi is an estimate of the single-thread speed of a CPU - the less computing time a CPU takes, the faster the CPU is. Therefore, our machine is as fast as the machine used in Delorme et al. (2016).

Delorme et al. preliminary computed lower and upper bounds through a simple procedure and BELOV and VPSOLVER were only executed on instances for which lower and upper bounds did not coincide. We therefore selected the same subset of instances to compare the results of EXM. Moreover, on classes of instances of Falkenauer, Scholl et al., Wäscher and Gau, Schwerin and Wäscher, and Schoenfield (hereafter called *Easy* class) and on Random class, Delorme et al. reported the results obtained by running the codes with a time limit of one minute and of 10 minutes.

Tables 2 and 3 summarize the results obtained with a time limit of one minute on the Easy and Random classes, respectively. In the tables, column "$\#Inst$" gives the number of instances for which the codes were executed, column "$\#opt$" gives the number of instances that were solved to proven optimality and column "t" reports the average computing time expressed in seconds. The last line of each table reports, for each method, the total number of solved instances and the average computing time.

Table 2 shows that EXM, within the time limit of one minute, solved almost all instances of the Easy class, BELOV and VPSOLVER solved 953 and 928 out of the 976 instances, respectively. On class Random, BELOV solved all instances and EXM all but one instance.

We also executed EXM with a time limit of 10 minutes. On class Random, BELOV, VPSOLVER, and EXM solved all instances. On the easy class, BELOV also solved all instances, VPSOLVER solved 969 out of 976 instances and EXM all but one instance (one instance of class Hard28). The corresponding results are given in tables 4 and 5. For methods BELOV and VPSOLVER, we report only the number of instances solved to optimality since the corresponding average computing times have not been reported by Delorme et al. (2016).

The results on the Easy and Random classes indicates that EXM outperforms VPSOLVER and is highly competitive against BELOV.

Table 2 Results on classes of instances of Falkenauer, Scholl et al., Wäscher and Gau, Schwerin and Wäscher, and Schoenfield

Class	$\#Inst$	BELOV		VPSOLVER		EXM	
		$\#opt$	t	$\#opt$	t	$\#opt$	t
Falkenauer U	74	74	0.0	74	0.1	74	2.0
Falkenauer T	80	57	24.7	80	0.4	80	3.8
Scholl 1	323	323	0.0	323	0.1	323	0.2
Scholl 2	244	244	0.3	208	14.0	244	7.0
Scholl 3	10	10	14.1	10	6.3	10	4.5
Wäscher	17	17	0.1	6	49.4	16	11.5
Schwerin 1	100	100	1.0	100	0.3	100	0.1
Schwerin 2	100	100	1.4	100	0.3	100	0.2
Hard28	28	28	7.3	27	14.2	26	8.5
	976	953	2.7	928	5.0	973	2.8

Table 3 Results on class Random of Deloorme et al.

n	$\#Inst$	BELOV		VPSOLVER		EXM	
		$\#opt$	t	$\#opt$	t	$\#opt$	t
50	165	165	0.0	165	0.0	165	0.0
100	271	271	0.0	271	0.1	271	0.0
200	359	359	0.0	359	0.3	359	0.1
300	393	393	0.1	393	0.6	393	0.2
400	425	425	0.2	425	0.8	425	0.4
500	414	414	0.2	413	1.7	414	0.6
750	433	433	0.4	431	2.4	433	1.7
1000	441	441	0.7	434	3.4	440	3.3
	2901	2901	0.2	2891	1.4	2900	0.8

Table 6 reports the results obtained on ANI an AI classes of instances. For these classes, a time limit of 3600 seconds was imposed by Delorme et al., therefore, the time limit of EXM has been set equal to 3600 seconds.

Table 4 Results on classes of instances of Falkenauer, Scholl et al., Wäscher and Gau, Schwerin and Wäscher, and Schoenfield with a time limit of 10 minutes

Class	$\#Inst$	BELOV	VPSOLVER	EXM	
		$\#opt$	$\#opt$	$\#opt$	t
Falkenauer U	74	74	74	74	2.0
Falkenauer T	80	80	80	80	3.8
Scholl 1	323	323	323	323	0.2
Scholl 2	244	244	242	244	7.0
Scholl 3	10	10	10	10	4.5
Wäscher	17	17	13	17	16.3
Schwerin 1	100	100	100	100	0.1
Schwerin 2	100	100	100	100	0.2
Hard28	28	28	27	27	8.5
	976	976	969	975	4.7

Table 5 Results on class Random of Deloorme et al. with a time limit of 10 minutes

n	$\#Inst$	BELOV	VPSOLVER	EXM	
		$\#opt$	$\#opt$	$\#opt$	t
50	165	165	165	165	0.0
100	271	271	271	271	0.0
200	359	359	359	359	0.1
300	393	393	393	393	0.2
400	425	425	425	425	0.4
500	414	414	414	414	0.6
750	433	433	433	433	1.7
1000	441	441	441	441	3.3
	2901	2901	2901	2901	0.8

In the table, column "$\#opt$" reports the number of instances solved to optimality by the methods and column "t" reports the corresponding average computing time in seconds. Table 6 shows that EXM outperform BELOV and VPSOLVER; EXM solved to optimality 97 and 140 instances of classes ANI and AI, respectively. Overall, EXM could solve 70 and 84 more instances than BELOV and VPSOLVER, respectively. In particular, two instances involving 1003 items and a bin capacity equal to 80,000 have been solved to optimality for the first time. The table also shows that on the instances solved to optimality by all methods, the average time of EXM is significantly lower.

The detailed results of EXM are summarized in Tables 7, 8 and 9. For sake of completeness, the results are reported for all the instances of the different classes. The columns of the tables report average values of the roundup lower bound without ("lb_{noCut}") and with ("lb") SR inequalities, the number of bins in the optimal solution ("$\#bin$"), the numbers of nodes explored by EXM ("$\#node$"), the number of nodes explored by the primal heuristic ("$\#node_p$"), the number of columns added to the restricted master problem ("$\#col$").

Table 6 Results on classes ANI and AI of Deloorme et al.

Class	$\#Inst$	n	c	BELOV		VPSOLVER		EXM	
				$\#opt$	t	$\#opt$	t	$\#opt$	t
ANI	50	201	2500	50	144.0	47	415.0	50	13.9
	50	402	10,000	1	3556.0	6	3304.0	47	436.2
	50	600	20,000	0	3602.0	0	3600.0	0	3602.7
	50	801	40,000	0	3602.0	0	3600.0	0	3605.9
	50	1002	80,000	-	-	-	-	0	3637.7
AI	50	202	2500	50	91.0	50	54.0	50	4.2
	50	403	10,000	45	699.0	42	1130.0	46	398.1
	50	601	20,000	21	2539.0	8	3509.0	27	1759.6
	50	802	40,000	0	3601.0	0	3600.0	15	2766.3
	50	1003	80,000	-	-	-	-	2	3546.1
	500			167		153		237	

Columns "t", "t_{lp}", "t_{primal}" and "t_{price}" report the total computing time in seconds of EXM, the primal simplex optimizer, the primal heuristic and of the label-setting algorithm, respectively. Times t_{lp} and t_{price} also include the times spent by the simplex optimizer and by the label-setting algorithm during the execution of the primal heuristic, respectively.

The last two columns "$\#cut_{root}$" and "$\#cut$" show the number of SR inequalities added to the restricted master problem at the root node and at all the explored nodes, respectively.

Table 7 Detailed results on classes of instances of Falkenauer, Scholl et al., Wäscher and Gau, Schwerin and Wäscher, and Schoenfield

Class	$\#opt$	lb_{noCut}	lb	$\#bin$	$\#node$	$\#node_p$	$\#col$	t	t_{lp}	t_{primal}	t_{price}	$\#cut_{root}$	$\#cut$
Falkenauer U	80	77.5	77.5	77.5	1.0	70.8	1353.6	3.8	3.3	0.2	2.2	11.7	11.7
Falkenauer T	80	188.1	188.1	188.1	1.0	8.6	1087.7	1.9	1.7	0.0	0.4	8.3	8.3
Scholl 1	720	108.9	108.9	108.9	1.0	0.5	292.7	0.2	0.1	0.0	0.0	1.2	1.2
Scholl 2	480	42.2	42.2	42.2	1.0	5.0	982.6	5.1	4.4	0.2	1.3	15.5	15.5
Scholl 3	10	56.2	56.2	56.2	1.0	9.9	972.6	4.5	0.5	3.8	0.5	15.1	15.1
Wäscher	17	17.3	17.4	17.4	424.1	14.4	1979.3	16.3	8.7	4.6	3.9	122.8	7328.5
Schwerin 1	100	18.0	18.0	18.0	1.0	0.2	409.3	0.1	0.1	0.0	0.0	7.1	7.1
Schwerin 2	100	21.9	21.9	21.9	1.0	0.3	576.4	0.2	0.2	0.0	0.0	7.1	7.1
Hard28	28	70.3	70.3	70.4	2596.4	166.8	1909.8	41.5	35.4	3.2	1.7	11.6	5837.3

The results reported in Table 9 shows that SR inequalities can improve the lower bounds on instances where the rounded lower bound is strictly smaller than the optimal solution cost. In Section 4.4 we also show that it is computationally convenient to use SR inequalities at the different nodes of the BCP tree.

Table 8 Detailed results on class Random of Deloorme et al.

Subclass	$\#opt$	lb_{noCut}	lb	$\#bin$	$\#node$	$\#node_p$	$\#col$	t	t_{lp}	t_{primal}	t_{price}	$\#cut_{root}$	$\#cut$
50	480	24.0	24.0	24.0	1.0	0.0	73.2	0.0	0.0	0.0	0.0	0.1	0.1
100	480	46.7	46.7	46.7	1.0	0.2	166.8	0.0	0.0	0.0	0.0	0.4	0.4
200	480	92.1	92.1	92.1	1.0	0.6	334.4	0.1	0.1	0.0	0.0	1.0	1.0
300	480	136.9	136.9	136.9	1.0	3.0	498.0	0.2	0.1	0.0	0.0	2.6	2.9
400	480	182.1	182.1	182.1	1.0	2.8	654.1	0.3	0.2	0.0	0.0	2.0	2.0
500	480	227.3	227.3	227.3	1.0	4.6	774.0	0.5	0.4	0.0	0.1	4.2	4.2
750	480	339.7	339.7	339.7	1.0	9.7	1081.3	1.5	1.2	0.1	0.3	9.4	9.4
1000	480	452.4	452.4	452.4	1.0	43.9	1365.1	3.6	2.8	0.1	1.3	10.2	10.3

Table 9 Detailed results on classes ANI and AI of Deloorme et al.

Class	n	c	$\#opt$	lb_{noCut}	lb	$\#bin$	$\#node$	$\#node_p$	$\#col$	t	t_{lp}	t_{primal}	t_{price}	$\#cut_{root}$	$\#cut$
	201	2500	50	65.0	65.8	66.0	51.0	137.5	3521.8	13.9	10.9	1.5	2.3	18.3	1591.1
	402	10,000	47	132.0	132.4	133.0	272.6	1948.5	9532.6	436.2	333.0	53.6	115.7	53.1	9418.3
ANI	600	20,000	0	198.0	198.0	199.0	1187.2	4402.1	22,213.7	3602.7	2900.4	514.9	624.7	48.6	29,046.6
	801	40,000	0	265.0	265.0	266.0	325.6	4476.7	22,118.8	3605.9	2632.3	861.9	1658.4	38.2	6082.0
	1002	80,000	0	332.0	332.0	333.0	1.9	3709.0	23,596.6	3637.7	2226.4	1399.1	2673.1	31.8	48.4
	202	2500	50	65.0	65.0	65.0	1.0	61.5	2965.4	4.2	3.3	0.7	1.4	26.2	26.2
	403	10,000	46	132.0	132.0	132.1	605.1	564.9	9538.8	398.1	290.5	62.8	33.9	30.7	8463.6
AI	601	20,000	27	198.0	198.0	198.5	715.5	2850.8	16,858.3	1759.6	1297.8	332.0	424.6	45.3	12,959.1
	802	40,000	15	265.0	265.0	265.7	168.2	4873.6	20,008.1	2766.3	1924.5	726.6	1601.4	36.8	3469.3
	1003	80,000	2	332.0	332.0	333.0	1.8	4228.4	23,664.5	3546.1	2030.5	1504.3	2560.6	32.8	48.1

4.3. Computational results on the 1D-BPPC

This section reports the results of EXM on 1D-BPPC instances. We compare EXM with the BP algorithm (hereafter called "GBP") proposed by Sadykov and Vanderbeck (2013) and with the exact method proposed by Brandão and Pedroso (2016) (called "VPSOLVER", as for the 1D-BPP) on the classes of 1D-BPPC instances reported in Table 1. The results reported by Sadykov and Vanderbeck (2013) and by Brandão and Pedroso (2016) show that GBP and VPSOLVER outperform the other algorithms proposed in the literature on classes "t-FMIMT", "u-FMIMT", "t-ELGN" and "u-ELGN".

According to the SuperPi (1M) benchmark, our computer is about 1.5 times faster than that of GBP, whose time limit is set to 3600 seconds. Therefore, we set time limit of EXM to be equal to 2400 seconds. In addition, the computer used by Brandão and Pedroso (2016) is about 1.4 faster than that of GBP.

Table 10 shows the results on classes of instances with interval conflict graphs, where all instances were solved to optimality by EXM, GBP and VPSOLVER. A symbol '-"

indicates that the class has not been considered by the corresponding method. For GBP and VPSOLVER, the table reports the average solution time ("t_{avg}") and the maximum solution time ("t_{max}"); for VPSOLVER, only column "t_{avg}" is reported. In the table, the computing times of EXM have been multiplied by 1.5 and those of VPSOLVER for 1.4, for sake of comparison with GBP.

The table shows that, on the different classes, the average times of EXM are significantly lower than that of GBP and that of VPSOLVER on classes t501, u500 and u1000.

Table 11 shows the results on "ta", "ua", and "da" instances. In the table, column "*#opt*" gives the number of instances solved to optimality for the corresponding class whereas column "t_{opt}" reports the average solution time only for the instances solved to optimality within the imposed time limit. Also in this table, the computing times of EXM have been multiplied by 1.5 for sake of comparison with GBP. No detailed computational results about these instances were reported by Brandão and Pedroso (2016). In their paper, Brandão and Pedroso mentioned that most of the instances were not solved to optimality with a time limit of 12 hours. The authors also observed that the combination of large capacities and long patterns is a limitation of their method.

The table shows that EXM could solve to optimality 35 more instances than GBP and that on 7 out of 11 classes EXM solved open 1D-BPPC instances; only on subclass "ua1000" EXM solved one instance less than GBP. Taking into account of the number of instances solved to optimality, EXM is generally faster than GBP.

The detailed computational results of EXM on the 1D-BPPC instances are summarized in tables 12 and 13.

According to the results, only in subclass "ta120" the SR inequalities can slightly improve the rounded lower bound, thus showing that their contribution on these classes of instances is very limited. Concerning the time spent by the different components of EXM, the tables show that a large portion of the total computing time is spent for the solution of the pricing problem.

4.4. Analysis of the different components of EXM

In this section, we analyse the impact of the primal heuristic, the pricing algorithm, the SR inequalities, and the Dominance 1 and Fathoming 1 rules on the performance of EXM.

We first compare the performance of EXM by disabling the use of the primal heuristic described in Section 3.2 (hereafter called "EXMnoPrimal"). Then, we conducted another

Table 10 Comparison results on 1D-BPPC classes of instances with interval conflict graphs

Class	Subclass	GBP		VPSOLVER	EXM	
		t_{avg}	t_{max}	t_{avg}	t_{avg}	t_{max}
d	d120	8.9	-	-	0.3	3.4
	d250	50.4	-	-	1.6	12.4
	d500	486.8	-	-	10.4	105.5
t-FMIMT	t60	0.8	6.5	0.1	0.1	0.7
	t120	37.8	2956.4	0.7	0.5	13.1
	t249	29.3	130.6	6.7	2.0	14.4
	t501	189.1	960.8	103.5	10.7	34.7
u-FMIMT	u120	2.8	26.2	0.6	0.2	6.3
	u250	12.5	35.9	5.4	5.8	4.7
	u500	70.3	154.9	63.2	6.1	20.7
	u1000	437.6	1133.1	1066.4	48.1	187.6
t-ELGN	t60	1.3	7.7	-	0.1	1.3
	t120	6.9	30.0	-	0.3	4.5
	t249	53.8	383.2	-	2.0	22.9
u-ELGN	u120	3.7	9.4	-	0.2	0.8
	u250	21.2	73.3	-	0.8	3.2
	u500	115.2	310.4	-	5.9	22.2

Table 11 Results on 1D-BPPC classes "ta", "ua" and "da"

Class	Subclass	GBP		EXM	
		#opt	t_{opt}	*#opt*	t_{opt}
ta	ta60	90	0.2	90	0.2
	ta120	90	4.2	90	0.9
	ta249	84	137.3	89	108.0
	ta501	65	392.6	74	170.1
ua	ua120	90	0.7	90	0.4
	ua250	88	9.0	90	7.8
	ua500	82	39.0	88	31.5
	ua1000	82	286.2	81	146.9
da	da120	67	23.2	78	64.8
	da250	50	23.3	52	148.3
	da500	49	137.6	50	129.2
		837		872	

experiment by replacing the label-setting algorithm described in Section 3.1 with our implementation of the branch-and-bound algorithm proposed by Bettinelli et al. (2017) for the knapsack problem with conflicts (hereafter called "EXM+BCM") also modified to consider the dual contributions given by the SR inequalities and to use a solution pool to store multiple solutions that are found during the exploration of the search tree (the maximum number of solutions generated is equal to parameter Δ, see Algorithm 1).

Table 12 Detailed results on 1D-BPPC classes of instances with interval conflict graphs

Class	Subclass	$\#opt$	lb_{noCut}	lb	$\#bin$	$\#node$	$\#node_p$	$\#col$	t	t_{primal}	t_{price}	$\#cut_{root}$	$\#cut$
d	d120	90	61.8	61.8	61.8	1.0	0.0	203.8	0.2	0.0	0.1	9.3	9.3
	d250	90	127.9	127.9	127.9	1.0	0.9	389.2	1.1	0.2	0.5	225.6	225.6
	d500	90	252.8	252.8	252.8	1.0	2.9	724.6	6.9	1.7	2.5	856.5	856.5
t-FMIMT	t60	90	33.4	33.4	33.4	1.0	0.4	200.8	0.1	0.0	0.0	0.6	0.6
	t120	90	66.1	66.1	66.1	3.9	11.5	407.8	0.3	0.1	0.1	5.5	22.4
	t249	90	135.8	135.8	135.8	1.0	36.7	812.5	1.3	0.6	0.4	10.5	10.5
	t501	90	275.7	275.7	275.7	1.0	66.8	1515.8	7.1	3.1	2.2	28.5	28.5
u-FMIMT	u120	90	70.4	70.4	70.4	1.0	54.1	346.1	0.2	0.1	0.0	1.6	1.7
	u250	90	143.7	143.7	143.7	1.0	11.1	733.8	0.6	0.1	0.2	5.4	5.4
	u500	90	286.0	286.0	286.0	1.0	55.4	1414.9	4.1	1.6	1.6	22.6	22.6
	u1000	90	571.9	571.9	571.9	1.0	191.9	2596.3	32.1	17.5	14.6	43.8	43.8
t-ELGN	t60	180	33.5	33.5	33.5	1.1	0.4	212.3	0.0	0.0	0.0	0.5	0.7
	t120	180	66.2	66.2	66.2	1.1	10.2	397.3	0.2	0.1	0.1	4.0	4.8
	t249	180	135.2	135.2	135.2	1.0	36.0	817.8	1.4	0.6	0.4	11.8	11.8
u-ELGN	u120	180	70.0	70.0	70.0	1.0	0.8	356.6	0.1	0.0	0.0	0.8	0.8
	u250	180	143.7	143.7	143.7	1.0	5.6	740.1	0.5	0.1	0.2	5.3	5.3
	u500	180	284.3	284.3	284.3	1.0	54.1	1420.6	4.0	1.5	1.6	20.9	20.9

Table 13 Detailed results on 1D-BPPC classes "ta", "ua", and "da"

Class	Subclass	$\#opt$	lb_{noCut}	lb	$\#bin$	$\#node$	$\#node_p$	$\#col$	t	t_{primal}	t_{price}	$\#cut_{root}$	$\#cut$
ta	ta60	90	21.8	21.8	21.9	1.1	1.2	260.6	0.1	0.0	0.0	0.6	0.6
	ta120	90	41.4	41.4	41.4	6.5	28.4	594.9	0.6	0.2	0.2	3.3	18.7
	ta249	89	83.8	83.8	83.8	1377.5	317.6	1564.6	97.8	5.3	35.7	6.0	5415.5
	ta501	74	167.4	167.4	167.6	1812.2	2324.7	3497.5	520.1	99.0	247.7	7.0	6840.9
ua	ua120	90	49.3	49.3	49.3	1.1	11.2	424.5	0.3	0.1	0.1	2.2	2.6
	ua250	90	100.6	100.6	100.6	56.1	110.6	895.0	5.2	1.3	2.4	4.9	161.0
	ua500	88	200.7	200.7	200.7	212.5	720.2	1655.1	73.9	22.7	41.2	8.1	670.5
	ua1000	81	399.9	399.9	400.0	1.0	3251.0	2904.3	328.6	309.2	149.8	16.6	16.6
da	da120	78	23.3	23.3	23.6	2714.1	36.0	2313.4	357.5	1.8	189.7	18.6	116,834.5
	da250	52	43.8	43.8	44.6	3544.4	306.2	3626.7	1070.6	38.0	663.2	11.9	85,094.0
	da500	50	82.5	82.5	84.1	507.2	1359.8	3989.6	1115.1	620.3	792.3	12.4	11,470.7

Table 14 summarizes the computational results obtained on the set of difficult instances of classes "ta", "ua", and "da". For each subclass, the table reports the number of instances solved to optimality ("$\#opt$") and the average solution time ("t") only for the instances solved to optimality. The results obtained clearly show the importance of the primal heuristic and the effectiveness of the label-setting algorithm when compared with our implementation of the branch-and-bound algorithm of Bettinelli et al. (2017) on the set of instances defined by our specific pricing problem. It is worth mentioning that the algorithm of Bet-

Table 14 Impact of the primal heuristic and of the pricing algorithm on EXM

Subclass	EXM		EXMnoPrimal		EXM+BCM	
	#opt	*t*	*#opt*	*t*	*#opt*	*t*
ta60	90	0.1	90	0.1	90	2.5
ta120	90	0.6	90	0.6	90	20.5
ta249	89	72.0	89	42.5	83	254.8
ta501	74	113.4	64	179.1	62	463.3
ua120	90	0.3	90	0.2	90	0.5
ua250	90	5.2	90	7.2	90	12.7
ua500	88	21.0	84	61.5	87	34.9
ua1000	81	97.9	76	148.4	79	159.8
da120	78	43.2	73	143.6	55	1155.5
da250	52	98.9	40	147.3	4	3514.6
da500	50	86.1	32	191.4	0	0.0
	872		818		730	

tinelli et al. has also been tested on different classes of KPC and that a complete comparison with the method of Bettinelli et al. is out of the scope of this paper. Our dynamic programming algorithm is therefore not proved to be better than that of Bettinelli et al. for solving the KPC, but, at least for the type of instances associated with our pricing problem, it turns out to be particular efficient.

We analyse the impact of SR inequalities on EXM by disabling the use of SR inequalities (we call the corresponding version "EXMNoCut"). For the 1D-BPP, our experiments shows that SR inequalities can improve the rounded lower bounds for instances of classes *Hard28* and *Wäscher*, therefore we selected some of the corresponding instances in order to test EXMNoCut. Table 15 reports the results obtained on the selected set of instances about EXM and EXMNoCut. For each version, column "*#node*" reports the final number of nodes explored whereas column "*t*" shows the computing time in seconds; concerning EXM, column "*#nodeCut*" reports the number of nodes explored where SR inequalities were useful in strengthening the lower bound. From the table, we can see that, on the selected set of instances, the use of SR inequalities significantly improves the performance of EXM.

We also run EXMNoCut on 1D-BPPC classes "ta", "ua", and "da", and the corresponding results are shown in Table 16. The table shows that, even if the average lower bounds at the root node cannot be improved by SR inequalities (see Table 13), it is computationally convenient to separate SR inequalities at the different nodes of the enumeration tree.

In order to have some details on the number of states or labels generated by Algorithm 1, Table 16 also gives for the basic version "EXMNoCut" some statistics about the number of

labels generated at the root node of the enumeration tree. More precisely, for each instance, we computed the minimum, average and maximum number of labels generated during the different calls to Algorithm 1. Columns $\#l_{min}$, $\#l_{avg}$, and $\#l_{max}$ report average values of the minimum, average and maximum values of the different instances, respectively.

Table 15 Effect of SR inequalities on 1D-BPP selected instances

Class	Name	n	EXM			EXMNoCut	
			$\#node$	t	$\#nodeCut$	$\#node$	t
Hard28	BPP716	180	65,369	966.3	7727	278,960	3600.0
	BPP419	200	256	14.1	71	338	12.7
	BPP359	180	4715	78.5	600	29,605	424.3
	BPP175	200	11	3.7	4	117	5.4
	BPP119	200	1	0.6	1	7	4.1
	BPP60	160	618	18.1	151	3443	58.6
	BPP40	160	54	4.6	11	56	4.3
	BPP14	160	1637	45.0	290	2799	54.6
Wäscher	TEST0065	60	7193	142.6	1415	13403	177.6

Table 16 Effect of SR inequalities on 1D-BPPC classes "ta", "ua", and "da"

Subclass	EXM		EXMNoCut				
	$\#opt$	t	$\#opt$	t	$\#l_{min}$	$\#l_{avg}$	$\#l_{max}$
ta60	90	0.1	90	0.1	31.4	156.0	367.6
ta120	90	0.6	90	0.6	42.2	498.6	2,008.7
ta249	89	72.0	89	70.5	45.0	1,564.3	15,703.5
ta501	74	113.4	74	132.8	42.4	5,071.8	109,151.4
ua120	90	0.3	90	0.2	37.4	287.9	878.5
ua250	90	5.2	90	5.7	72.7	584.4	2,389.1
ua500	88	21.0	89	42.1	95.7	838.4	4,206.1
ua1000	81	97.9	80	101.4	109.6	1,102.2	5,947.9
da120	78	43.2	77	59.2	156.8	3,674.3	14,785.3
da250	52	98.9	51	46.4	635.5	14,327.1	61,945.3
da500	50	86.1	50	94.1	1,776.0	43,531.2	189,230.9
	872		870				

We finally analyse the impact of Dominance 1 and Fathoming 1 rules on Algorithm 1. We selected a subset of difficult 1D-BPP instances (subclasses ANI201, ANI402, AI202 and AI403) and a subset of difficult 1D-BPPC instances (subclasses ta60, ta120, ua120, ua250, da120 and da250) for a total of 740 instances (200 1D-BPP instances and 540 1D-BPPC instances). We run EXM by disabling only the use of Dominance 1 (we call the corresponding version "EXMNoDom1") and also by disabling only the use of Fathoming

Table 17 Effectiveness of Dominance 1 and Fathoming 1 rules

Problem	Subclass	EXM		EXMNoDom1		EXMNoFath1	
		#opt	*t*	*#opt*	*t*	*#opt*	*t*
1D-BPP	ANI201	50	13.9	50	15.2	50	16.3
	ANI402	47	436.2	47	543.0	46	505.2
	AI202	50	4.2	50	6.3	50	6.1
	AI403	46	398.1	43	556.6	44	488.6
1D-BPPC	ta60	90	0.1	90	0.6	90	0.8
	ta120	90	0.6	90	1.9	90	5.5
	ua120	90	0.3	90	0.9	90	2.5
	ua250	90	5.2	90	7.9	90	61.8
	da120	78	43.2	75	46.3	57	890.3
	da250	52	98.9	54	52.6	13	2295.2
		683	100.1	679	123.1	620	427.2

1 (version "EXMNoFath1"). As for EXM, we imposed a time limit of 3600 seconds and of 2400 seconds to the executions of EXMNoDom1 and EXMNoDom1 on 1D-BPP and 1D-BPPC instances, respectively. The results obtained on the selected set of instances are given in Table 17 - the table shows, in addition to the results of EXM, the results of the two versions EXMNoDom1 and EXMNoFath1. The meaning of columns "*#opt*" and "*t*" is as described above. The last line of the table gives totals and averages computer over the values reported in the different columns.

Table 17 shows that the combined use of the two rules is particularly useful as EXM outperforms EXMNoDom1 and EXMNoFath1 both for the number of instances solved to optimality and the corresponding average computing time. Fathoming 1 is particularly effective, as shown by the results obtained without using the rule (version EXMNoFath1). Dominance 1 is also effective but its use can be time consuming, as shown by the fact that on subclass da250 of the 1D-BPP, EXM solved 52 instances whereas EXMNoDom1 solved 54 instances.

5. Conclusions and future research

In this paper, we considered the one-dimensional bin packing problem (1D-BPP) and the 1D-BPP with conflicts (1D-BPPC).

We designed a new branch-and-price-and-cut algorithm based on the set partitioning formulation with additional constraints that correspond to a subset of the subset-row inequalities. In particular, we implemented an ad-hoc label-setting algorithm and dominance and fathoming rules used to speed up its computation.

Our algorithm was tested on several classes of 1D-BPP and 1D-BPPC instances from the literature and the results compared with those obtained with state-of-the-art algorithms. For the 1D-BPP, the proposed method successfully solved 70 difficult instances open so far; the largest instance involves 1003 items and bins of capacity 80,000. For the 1D-BPPC, the experiments show that the method is highly competitive with state-of-the-art algorithms specifically designed for the CSP and that also represent the best exact methods for the 1D-BPPC, and successfully closed several open 1D-BPPC instances.

Future work will address other variants of the 1D-BPP that have been proposed to consider different constraints arising from practical applications, such as the variable sized bin packing problem, the bin packing with color constraints, the ordered open-end bin packing problem, the bin packing problem with precedence constraints, to name a few.

Acknowledgments

The authors would like to thank the anonymous reviewers and associate editor for their extremely helpful suggestions and very thorough review of the paper.

This research was partially supported by the National Natural Science Foundation of China (Grant Nos. 71501091, 71531009, 71571077, 71501075), Guangdong Natural Science Funds for Distinguished Young Scholar (Grant No. 2015A030306007), NRF Singapore (Grant No. NRF-RSS2016-004) and MOE-AcRF-Tier 1 (Grants Nos. R-266-000-096-133, R-266-000-096-731, R-266-000-100-646).

References

Archetti C, Bouchard M, Desaulniers G (2011) Enhanced branch and price and cut for vehicle routing with split deliveries and time windows. *Transp. Sci.* 45(3):285–298.

Balas E (1965) An additive algorithm for solving linear programs with zero-one variables. *Oper. Res.* 13(4):517–546.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Belov G, Scheithauer G (2006) A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *Eur. J. Oper. Res.* 171(1):85–106.

Bettinelli A, Cacchiani V, Malaguti E (2017) A Branch-and-bound Algorithm for the knapsack problem with conflict graph. *INFORMS J. Comput.* 1–24.

Brandão F, Pedroso JP (2016) Bin packing and related problems: General arc-flow formulation with graph compression. *Compters & Oper. Res.* 69:56–67.

Cambazard H, O'Sullivan B (2010) Propagating the bin packing constraint using linear programming. *Int. Conf. Princ. Pract. Constraint Program.*, 129–136 (Springer).

Caprara A, Dell'Amico M, Díaz-Díaz JC, Iori M, Rizzi R (2015) Friendly bin packing instances without Integer Round-up Property. *Math. Program.* 150(1):5–17.

Chvátal V (1973) Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Math.* 4(4):305–337.

Contardo C, Cordeau JF, Gendron B (2013) An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS J. Comput.* 26(1):88–102.

Contardo C, Martinelli R (2014) A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discret. Optim.* 12:129–146.

Degraeve Z, Peeters M (2003) Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS J. Comput.* 15(1):58–81.

Degraeve Z, Schrage L (1999) Optimal integer solutions to industrial cutting stock problems. *INFORMS J. Comput.* 11(4):406–419.

Delorme M, Iori M, Martello S (2016) Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Eur. J. Oper. Res.* 255(1):1–20.

Delorme M, Iori M, Martello S (2017) Bpplib: a library for bin packing and cutting stock problems. *Optimization Letters* .

Dupuis J, Schaus P, Deville Y (2010) Consistency check for the bin packing constraint revisited. *Int. Conf. Integr. Artif. Intell. Oper. Res. Tech. Constraint Program.*, 117–122 (Springer).

Eilon S, Christofides N (1971) The loading problem. *Manage. Sci.* 17(5):259–268.

Elhedhli S, Li L, Gzara M, Naoum-Sawaya J (2011) A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS J. Comput.* 23(3):404–415.

Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. *J. Heuristics* 2(1):5–30.

Fernandes Muritiba AE, Iori M, Malaguti E, Toth P (2010) Algorithms for the bin packing problem with conflicts. *Informs J. Comput.* 22(3):401–415.

Gendreau M, Laporte G, Semet F (2004) Heuristics and lower bounds for the bin packing problem with conflicts. *Comput. & Oper. Res.* 31(3):347–358.

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Oper. Res.* 9(6):849–859.

IBM CPLEX (2016) *IBM ILOG CPLEX 12.6.3 callable library.*

Jansen K (1999) An approximation scheme for bin packing with conflicts. *J. Comb. Optim.* 3(4):363–377.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.* 3(4):299–325.

Joncour C, Michel S, Sadykov R, Sverdlov D, Vanderbeck F (2010) Column Generation based Primal Heuristics. *ISCO 2010 - Int. Symp. Comb. Optim.* 36:695–702.

Kantorovich LV (1960) Mathematical methods of organizing and planning production. *Management Science, English translation of a 1939 paper written in Russian* 6:366–422.

Kartak VM, Ripatti AV, Scheithauer G, Kurz S (2015) Minimal proper non-irup instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics* 187:120–129.

Korf RE (2002) A new algorithm for optimal bin packing. *AAAI/IAAI*, 731–736.

Laporte G, Desroches S (1984) Examination timetabling by computer. *Comput. & Oper. Res.* 11(4):351–360.

Malaguti E, Monaci M, Toth P (2008) A metaheuristic approach for the vertex coloring problem. *INFORMS J. Comput.* 20(2):302–316.

Malaguti E, Monaci M, Toth P (2011) An exact approach for the vertex coloring problem. *Discrete Optimization* 8(2):174–190.

Marcotte O (1986) An instance of the cutting stock problem for which the rounding property does not hold. *Oper. Res. Lett.* 4(5):239–243.

Martello S, Toth P (1990) *Knapsack Problems: Algorithms and Computer Implementations* (Chichester: John Wiley & Sons).

Mukhacheva EA, Belov GN, Kartack VM, Mukhacheva AS (2000) Linear one-dimensional cutting-packing problems: numerical experiments with the sequential value correction method (SVC) and a modified branch-and-bound method (MBB). *Pesqui. Operacional* 20(2):153–168.

Plum CEM, Pisinger D, Salazar-González JJ, Sigurd MM (2014) Single liner shipping service design. *Comput. & Oper. Res.* 45:1–6.

Ryan D, Foster B (1981) An integer programming approach to scheduling. *Comput. Sched. public Transp. urban Passeng. Veh. crew Sched.* 269–280.

Sadykov R, Vanderbeck F (2013) Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS J. Comput.* 25(2):244–255.

Schaus P, Régin JC, Van Schaeren R, Dullaert W, Raa B (2012) Cardinality reasoning for bin-packing constraint: application to a tank allocation problem. *Princ. Pract. Constraint Program.*, 815–822 (Springer).

Scheithauer G, Terno J (1995) The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research* 84:562–571.

Scheithauer G, Terno J (1997) Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters* 20:93–100.

Scheithauer G, Terno J, Müller A, Belov G (2001) Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm. *J. Oper. Res. Soc.* 52(12):1390–1401.

Schoenfield J (2002) Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command,Huntsville,Alabama,USA.

Scholl A, Klein R, Jürgens C (1997) BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Comput. & Oper. Res.* 24(7):627–645.

Schreiber EL, Korf RE (2013) Improved Bin Completion for Optimal Bin Packing and Number Partitioning. *IJCAI*, 651–658.

Schwerin P, Wäscher G (1998) *A new lower bound for the bin-packing problem and its integration into MTP* (Martin-Luther-Univ. Halle-Wittenberg, Wirtschaftswiss. Fak.).

Shaw P (2004) A constraint for bin packing. *Int. Conf. Princ. Pract. Constraint Program.*, 648–662 (Springer).

Valério de Carvalho, M J (1999) Exact solution of bin-packing problems using column generation and branch-and-bound. *Ann. Oper. Res.* 86(0):629–659.

Valério de Carvalho, M J (2002) LP models for bin packing and cutting stock problems. *Eur. J. Oper. Res.* 141(2):253–273.

Vance PH (1998) Branch-and-price algorithms for the one-dimensional cutting stock problem. *Comput. Optim. Appl.* 9(3):211–228.

Vance PH, Barnhart C, Johnson EL, Nemhauser GL (1994) Solving binary cutting stock problems by column generation and branch-and-bound. *Comput. Optim. Appl.* 3(2):111–130.

Vanderbeck F (1999) Computational study of a column generation algorithm for bin packing and cutting stock problems. *Math. Program.* 86(3):565–594.

Vanderbeck F (2011) Branching in branch-and-price: a generic scheme. *Math. Program.* 130(2):249–294.

Wäscher G, Gau T (1996) Heuristics for the integer one-dimensional cutting stock problem: A computational study. *Operations-Research-Spektrum* 18(3):131–144.

Wolfson ML (1965) Selecting the best lengths to stock. *Oper. Res.* 13(4):570–585.

Xue L, Luo Z, Lim A (2015) Two exact algorithms for the traveling umpire problem. *Eur. J. Oper. Res.* 243(3):932–943.