

# A lexicographic pricer for the fractional bin packing problem

Stefano Coniglio, Fabio D'Andreagiovanni and Fabio Furini

Department of Mathematical Sciences, University of Southampton, University Road, SO17 1BJ, Southampton, UK

French National Center for Scientific Research (CNRS), France, Sorbonne Universités, Université de Technologie de Compiègne, CNRS, Heudiasyc UMR 7253, CS 60319, 60203 Compiègne Cedex, France

LAMSADE, Université Paris-Dauphine, 75775, Paris, France

## ARTICLE INFO

### Keywords:

fractional bin packing problem  
column generation  
dynamic programming  
lexicographic optimization

## ABSTRACT

We propose an exact lexicographic dynamic programming pricing algorithm for solving the Fractional Bin Packing Problem with column generation. The new algorithm is designed for generating maximal columns of minimum reduced cost which maximize, lexicographically, one of the measures of maximality we investigate. Extensive computational experiments reveal that a column generation algorithm based on this pricing technique can achieve a substantial reduction in number of columns and computing time, also when combined with a classical smoothing technique from the literature.

## 1. Introduction

Given a set  $N = \{1, \dots, n\}$  of items with positive integer weights  $w_1, \dots, w_n$  and an unlimited number of bins with a positive integer capacity  $C$ , the *Bin Packing Problem* (BPP) asks to compute the minimum number of bins needed to pack all the items. The BPP is a key problem in combinatorial optimization. It is  $\mathcal{NP}$ -hard and also very challenging to solve from a computational viewpoint. For recent works on the problem and for a survey, we refer the reader to [9, 11, 12, 15].

In this paper, we focus on solving the *Fractional Bin Packing Problem* (FBPP)—a linear programming relaxation of the BPP which is known to provide very strong dual bounds and whose variables correspond to sets of items (or patterns) fitting into a bin [7, 15]. State-of-the-art methods for solving the FBPP rely on a *Column Generation* (CG) approach by which the patterns (columns) are generated one at a time.

As inclusion-wise maximal columns of the FBPP dominate nonmaximal ones (see Section 3 for more details), enforcing the generation of maximal columns is advisable when solving the FBPP with CG. Since the problem often admits many columns of minimum reduced cost which are also maximal (see Example 1 in Section 3), it is natural to ask oneself whether, among many such maximal columns, one should be preferred to the another ones in order to obtain a speed up in the convergence of the CG algorithm.

We investigate this question by studying three different *measures of maximality* and by proposing an exact *lexicographic dynamic programming* pricing algorithm capable of generating maximal columns of minimum reduced cost which also maximize, lexicographically, one of the three measures.

## 2. Preliminaries

We call *pattern* any feasible set of items  $S \subseteq N$  with  $\sum_{j \in S} w_j \leq C$ . Let  $\mathcal{S}$  be the collection of all patterns, namely,  $\mathcal{S} := \{S \subseteq N : \sum_{j \in S} w_j \leq C\}$ . Let  $A \in \{0, 1\}^{n \times |\mathcal{S}|}$  be a binary matrix with  $a_{jS} = 1$  if and only if item  $j \in N$  belongs to pattern  $S \in \mathcal{S}$ , and let  $e$  be the all-one vector of  $n$  components.


Upon introducing a nonnegative variable  $\xi_S$  for each pattern  $S \in \mathcal{S}$ , the FBPP amounts to solving the following linear programming problem:

$$(\text{FBPP}) \quad \min_{\xi \geq 0} \left\{ \sum_{S \in \mathcal{S}} \xi_S : A\xi \geq e \right\}. \quad (1)$$

The dual bound provided by the FBPP is very tight, as the absolute difference (gap) between the optimal solution values of the FBPP and BPP is, typically, smaller or equal to 1. Since classes of BPP instances in which this difference is exactly 1 are known in the literature [8], the FBPP does not enjoy the *integer round-up property* (we consider an instance with a gap of 1 in Section 5). To the best of our knowledge, no instance with a gap strictly greater than 1 has been found. It is indeed conjectured (*modified integer round-up property*) that the gap can never be larger than 1 [8].

The FBPP is usually solved via a CG method. We call *Restricted Master Problem* (RMP) a restriction of Formulation (1) to a subset of columns  $\tilde{\mathcal{S}}$  which admits a feasible solution ( $\tilde{\mathcal{S}}$  comprises an initial pool of columns plus all those which have been generated up to the current iteration). Iteratively, the CG method solves the RMP and the corresponding *Pricing Problem* (PP) so to determine a new pattern (column) to be added to  $\tilde{\mathcal{S}}$ . The procedure is iterated until no more columns with a negative reduced cost are found and, thus, the RMP is solved to optimality.

At a given CG iteration, let  $\pi^* \in \mathbb{R}_+^n$  be a vector of optimal dual variables corresponding to an optimal solution  $\xi^*$  of the RMP. The PP can be formulated as the following 0-1 *knapsack problem*, where  $x_j = 1$  if and only if the new

 s.coniglio@soton.ac.uk (S. Coniglio);

d.andreagiovanni@hds.utc.fr (F. D'Andreagiovanni);

fabio.furini@dauphine.fr (F. Furini)

ORCID(s): 0000-0001-9568-4385 (S. Coniglio); 0000-0003-0872-3636 (F. D'Andreagiovanni); 0000-0002-1839-5827 (F. Furini)

pattern contains the item of index  $j \in N$ :

$$(PP) \quad \max_{x \in \{0,1\}^n} \left\{ \sum_{j \in N} \pi_j^* x_j : \sum_{j \in N} w_j x_j \leq C \right\}. \quad (2)$$

Let  $x^*$  be an optimal solution to the PP. If  $\sum_{j \in N} \pi_j^* x_j^* > 1$  (i.e., the reduced cost of the new pattern is negative), the pattern  $S^* := \{j \in N : x_j^* = 1\}$  is added to  $\mathcal{S}$  and the procedure is reiterated.

We remark that the dual of Formulation (1) contains a constraint of type  $\sum_{j \in S} \pi_j \leq 1$  for each  $S \in \mathcal{S}$ , and that generating a column of minimum reduced cost ( $1 - \sum_{j \in S^*} \pi_j^*$ ) corresponds to separating, in the dual of the RMP, an inequality of maximum violation.

In the context of CG approaches for solving the BPP/FBPP problems, state-of-the-art algorithms for the solution of Problem (2) rely on *Dynamic Programming* (DP) [25, 30]. These algorithms are based on the following recursive equation, which is satisfied for all  $j \in N$  and  $s \in \{0, \dots, C\}$ :

$$\phi_j(s) := \max\{\phi_{j-1}(s), \phi_{j-1}(s - w_j) + \pi_j^*\}, \quad (3)$$

where  $\phi_j(s)$  is defined as the optimal solution value of a restriction of the problem to items in  $\{1, \dots, j\}$  and a capacity of  $s \leq C$ . We assume  $\phi_0(s) := 0$  for all  $s \in \{0, \dots, C\}$  and  $\phi_0(s) := -\infty$  for every  $s < 0$ . The optimal solution value of the PP corresponds to  $\phi_n(C)$ .

For every  $S \subseteq N$ , we define  $w(S) := \sum_{j \in S} w_j$  and  $\pi^*(S) := \sum_{j \in S} \pi_j^*$ . The DP algorithm relies on a list  $F$  of maximum size  $C$  which, at each iteration  $j$ , contains the value of  $\phi_j(s)$  in each position indexed by  $s \leq C$ . Each of the values of  $F$  is computed in  $O(1)$  thanks to Equation (3). The algorithm runs in  $O(nC)$ . An optimal solution can be obtained in  $O(n)$  by storing a binary matrix  $U$  satisfying  $U(j, s) = 1$  if and only if item  $j$  is added to the optimal solution found by the DP algorithm of the problem restricted to items in  $\{1, \dots, j\}$  and to capacity  $s \leq C$ . Equation (3) corresponds to a *dominance rule* according to which, whenever two solutions  $S', S''$  of the same total weight  $w(S') = w(S'')$  are found, only the one with larger profit (either  $\pi^*(S')$  or  $\pi^*(S'')$ ) is stored.

In the next section, we show how to extend this DP algorithm so to guarantee the generation of a column of minimum reduced cost which also maximizes, lexicographically, a measure of maximality. The overall aim is obtaining a method which exhibits, in practice, a faster convergence.

We remark that CG methods are well known to suffer from a number of computational issues, including dual oscillations, tailing-off effects, and primal degeneracy of the master problem [16]. After the seminal work of [26], several stabilization techniques have been proposed in the literature [31, 10, 21, 5]. While the method that we propose here is aimed at accelerating the practical convergence rate of the CG algorithm, it is not a stabilization technique of the dual variables. It is, rather, a parameter-free way of obtaining a similar effect without having to resort to often complicated stabilization techniques that typically require a good degree of parameter tuning [27].

### 3. The search for maximal columns

A pattern  $S \in \mathcal{S}$  and the corresponding column are called *maximal* if, for all the items  $j \in N \setminus S$ ,  $S \cup \{j\} \notin \mathcal{S}$ . By inspecting the dual of Formulation (1), one can see that any nonmaximal pattern  $S$  originates a dual inequality which is *dominated* by at least another inequality corresponding to a maximal pattern  $S' \supset S$ . It follows that nonmaximal columns/patterns can be discarded.

Maximal columns can be achieved by applying an *a posteriori* greedy procedure which, after the PP is solved, adds to the generated pattern items  $j \in N$  with  $\pi_j^* = 0$  (in any order) until either the capacity  $C$  is saturated or no other item fits. This is equivalent to applying a *sequential lifting* procedure on the corresponding dual constraint, see [29], which suffices to show that maximal columns are facet-defining for the dual of the FBPP.

As we mentioned (see also Example 1), the FBPP may admit many different *maximal* columns even when restricting ourselves to columns of minimum reduced cost. This is clear if  $\pi^*$  is sparse, which is often the case as, due to complementary slackness,  $\pi_j^* = 0$  whenever  $\sum_{S \in \mathcal{S}: j \in S} \xi_S > 1$ . The natural question is then: *should some of these maximal columns be preferred to the other ones?* We address this question in the remainder of the paper.

#### 3.1. Measures of maximality

We consider the following three measures of maximality: (i) *Weight*, (ii) *Diversity*, and (iii) *Density*.

**Weight.** This measure is equal to the total weight of the items in the pattern. Formally:

$$g_w(x) := \sum_{j \in N} w_j x_j. \quad (4)$$

A pattern  $S \subseteq N$  has maximum weight  $\sum_{j \in S} w_j$  if and only if it has minimum waste  $C - \sum_{j \in S} w_j$ . It is known that, in the integer case, any solution with more than one bin of waste larger than  $\frac{C}{2}$  cannot be optimal (see Chapter 8.3.2 of [25]). This can be extended to the case of the FBPP as follows:

**Proposition 1.** *Given any two distinct patterns  $S_1, S_2$  each of waste greater or equal than  $\frac{C}{2}$ , a solution to the FBPP with  $\xi_{S_1} > 0$  and  $\xi_{S_2} > 0$  cannot be optimal.*

**Proof:** Since  $S_1$  and  $S_2$  have waste greater or equal than  $\frac{C}{2}$ , they can be merged into a new pattern  $S' := S_1 \cup S_2$ . Letting  $\xi_{S'} := \max\{\xi_{S_1}, \xi_{S_2}\}$ ,  $\xi_{S_1} := 0$ , and  $\xi_{S_2} := 0$ , we obtain another feasible solution with an objective function value smaller than that of the original one, precisely we have  $\xi_{S_1} + \xi_{S_2} - \max\{\xi_{S_1}, \xi_{S_2}\} > 0$ .  $\square$

The following proposition establishes a bijection between optimal solutions of the FBPP and those of an alternative master problem in which the total waste is minimized:

**Proposition 2.** *The optimal solutions of the FBPP as defined in (1) and those of the following alternative problem*

coincide:

$$\arg \min_{\xi \geq 0} \left\{ \sum_{S \in \mathcal{S}} (C - \sum_{j \in S} w_j) \xi_S : \sum_{S \in \mathcal{S}: j \in S} \xi_S = 1, \forall j \in N \right\}.$$

**Proof:** Due to the packing constraint, the objective function of the alternative problem is obtained by a nonnegative affine transformation of the objective function of the FBPP. Indeed, the former can be rewritten as follows:

$$\sum_{S \in \mathcal{S}} (C - \sum_{j \in S} w_j) \xi_S = C \sum_{S \in \mathcal{S}} \xi_S - \sum_{S \in \mathcal{S}} \sum_{j \in S} w_j \xi_S = C \sum_{S \in \mathcal{S}} \xi_S - \sum_{j \in N} w_j \sum_{S \in \mathcal{S}: j \in S} \xi_S.$$

As  $\sum_{S \in \mathcal{S}: j \in S} \xi_S = 1$  for each  $j \in N$ , the last expression is equal to  $C \sum_{S \in \mathcal{S}} \xi_S - \sum_{j \in N} w_j$ : an affine function of the FBPP's objective function. The claim follows since, w.l.o.g., in an optimal FBPP solution the covering constraint is satisfied as an equation.  $\square$

Since any solution which only contains patterns of zero waste is optimal for the alternative problem (as it achieves an objective function value of 0), Proposition 2 implies that such solution is also optimal for the FBPP. As we discuss in Subsection 5.3, this is often the case in many of the instances of the literature. In the same subsection, we also show that even instances that do not admit optimal solutions with patterns of zero waste contain, on average, patterns whose waste is very small. This suggests that by favoring the generation of columns of large weight we can introduce columns which are more likely to be contained in an optimal FBPP solution earlier during the execution of the CG algorithm.

**Diversity.** In the cutting plane literature, a number of papers have observed the benefits of generating diversified cuts, see [20, 6, 32, 3]. Since a (primal) column generation method can be seen as a cutting plane method for the dual, investigating the benefits of generating diversified columns is reasonable. This has already been observed (with different techniques than those proposed in this paper) in [28, 24, 23].

We consider, here, a measure originally proposed in [2, 3] in the context of a cutting plane method to generate, among all cuts of maximum violation, one which is maximally diverse with respect to the previously generated ones. In the column generation setting, the measure is equal to the 1-norm distance  $\|x - \tilde{s}\|_1$  between the column  $x$  to be generated and the average  $\tilde{s}$  of the previously generated columns, additively combined with trade-off factor  $\delta > 0$  with the density  $\|x\|_1$  of  $x$ . Formally:

$$g_c(x) := \|x - \tilde{s}\|_1 + \delta \|x\|_1. \quad (5)$$

With  $\delta > 1$ , one can show that only maximal columns will be produced [2, 3]. Since  $x$  is binary,  $\|x - \tilde{s}\|_1 + \delta \|x\|_1$  can be rewritten as  $\sum_{j \in N} x_j - 2 \sum_{j \in N} \tilde{s}_j x_j + \sum_{j \in N} \tilde{s}_j + \delta \sum_{j \in N} x_j$ . When letting  $\delta = 2$  (as suggested in [3]), we have:

$$g_c(x) = \sum_{j \in N} (3 - 2\tilde{s}_j) x_j + \sum_{j \in N} \tilde{s}_j.$$

The second term, being a constant, can be dropped.

Differently from [3], which reformulates the lexicographic problem where  $g_c$  is maximized as a single-objective problem employing an exact trade-off factor  $\epsilon$  (with the drawback that numerical issues may arise if  $\epsilon$  is too small), the

approach we propose in this paper relies on a suitably designed lexicographic dynamic programming method which is entirely parameter-free and does not suffer from numerical issues (see Section 4).

**Density.** The last measure we consider is equal to the number of items in the pattern  $S = \{j \in N : x_j = 1\}$ . Formally:

$$g_d(x) := \|x\|_1 = \sum_{j \in N} x_j = |S|. \quad (6)$$

While being a quite intuitive way of obtaining a maximal column, maximizing  $g_d$  may have a negative impact as dense columns lead to a dense RMP and dense linear programs typically take longer to be reoptimized [1].

We now illustrate with an example that, while any column maximizing any of these three functions is maximal, maximizing the three quantities leads to different columns:

**Example 1.** Consider an instance with  $n = 6$  items,  $w = (50, 8, 9, 49, 26, 25)$ , and  $C = 100$ . Assume that the RMP contains seven patterns: all the singleton patterns  $S_j = \{j\}$  for all  $j \in N$  and the maximal pattern  $S_7 = \{1, 2, 3, 6\}$ . Let  $\pi^* = (1, 0, 0, 1, 1, 0)$  be an optimal dual solution. We have  $\tilde{s} = \left(\frac{2}{7}, \frac{2}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{7}, \frac{2}{7}\right)$  and  $3e - 2\tilde{s} = \left(\frac{17}{7}, \frac{17}{7}, \frac{17}{7}, \frac{19}{7}, \frac{19}{7}, \frac{17}{7}\right)$ . Consider the following three maximal patterns of minimum reduced cost (equal to  $-1$ ):  $S_8 = \{1, 2, 3, 5\}$  (with  $g_d = 4$ ,  $g_w = 93$ ,  $g_c = 10$ ),  $S_9 = \{2, 3, 4, 5\}$  (with  $g_d = 4$ ,  $g_w = 92$ ,  $g_c = \frac{72}{7}$ ), and  $S_{10} = \{4, 5, 6\}$  (having  $g_d = 3$ ,  $g_w = 100$ ,  $g_c = \frac{55}{7}$ ).  $S_{10}$  is of maximum weight, but not of maximum density nor diversity.  $S_8$  is of maximum density, but not of maximum weight nor diversity.  $S_9$  is also of maximum diversity (and also of maximum density), but not of maximum weight.

### 3.2. Maximality by design of the DP algorithm

We note that, for the FBPP, the lifting procedure used to produce a maximal pattern *a posteriori* is not needed. One can, indeed, always achieve a maximal pattern by a suitable implementation of the DP algorithm:

**Proposition 3.** Consider a DP algorithm with look-up table  $U$ . If the algorithm is implemented so that  $U(j, s) := 1$  if and only if  $\phi_{j-1}(s) \leq \phi_{j-1}(s - w_j) + \pi_j^*$  (i.e., so that  $j$  is added to the solution even if  $\phi_{j-1}(s) = \phi_{j-1}(s - w_j) + \pi_j^*$ ) for every pair  $(j, s)$ , then the algorithm always produces a maximal pattern.

**Proof:** Let  $x \in \{0, 1\}^n$  be the solution produced by the algorithm and assume, by contradiction,  $x$  nonmaximal. Pick any  $j \in N$  with  $x_j = 0$  such that  $x' := x + e_j$  is optimal (note that the optimality of  $x^*$  implies  $\pi_j^* = 0$ ). W.l.o.g., assume  $j = n$  (if not, ignore all the items from  $j + 1$  to  $n$  and redefine  $n := j$  and  $C := C - \sum_{i=j+1}^n w_i x_i$ ). Let the optimal profit be  $\Pi := \sum_{i=1}^j \pi_i^* x_i = \sum_{i=1}^j \pi_i^* x'_i$ . Since  $x$  is optimal and  $x_j = 0$ , by construction we have  $\phi_{j-1}(C) = \Pi$ . Since  $x'_j =$

1, item  $n$  fits in  $x$ . Since  $x'$  is optimal,  $x'_j = 1$ , and  $\pi_j^* = 0$ , by definition of  $\phi$  we have  $\phi_{j-1}(C - w_j) + \pi_j^* = \phi_{j-1}(C - w_j) = \Pi$ . Since  $x_j = 0$ , we have  $U(j, C) = 0$ . Since, by implementation hypothesis, this implies  $\phi_{j-1}(C) > \phi_{j-1}(C - w_j)$ , we have a contradiction.  $\square$

Generating a maximal pattern of minimum reduced cost this way presents the drawback of offering no control over which maximal pattern is produced.

#### 4. A novel lexicographic pricing problem

We introduce the following *Lexicographic Pricing Problem* (LPP) (a special case of a multi-objective problem with two objectives) which, among all columns of minimum reduced cost, finds one which maximizes one of the three maximality measures  $g_d$ ,  $g_w$ , and  $g_c$ :

$$(LPP) \quad \max_{x \in \{0,1\}^n} \text{lex} \left\{ (f(x), g(x)) : \sum_{j \in N} w_j x_j \leq C \right\}.$$

In the LPP,  $f(x) := \sum_{j \in N} \pi_j^* x_j$  and  $g(x)$  (equal to either  $g_d$ ,  $g_w$ , or  $g_c$ ) is optimized as a second-priority objective function over all the solutions which maximize  $f(x)$ . We now show how to solve the LPP with a *Lexicographic Dynamic Programming* (Lex-DP) in  $O(nC)$  time.

For each item  $j \in N$  and integer value  $s \in \{0, \dots, C\}$ , let  $(\phi_j(s), \gamma_j(s))$  be the values  $f$  and  $g$  of an optimal solution to the problem restricted to items in  $\{1, \dots, j\}$  and capacity  $s \leq C$ . Let us consider the partial order  $\geq$  defined such that, given  $(a, b), (a', b') \in \mathbb{R}^2$ ,  $(a, b) \geq (a', b')$  if and only if either  $a > a'$  or, if  $a = a'$ ,  $b \geq b'$ .

W.l.o.g., we rewrite  $g(x)$  as the linear function  $\sum_{j \in N} c_j x_j$ , for some  $c \in \mathbb{R}^n$ . For the three cases of density, weight, and diversity, i.e.,  $g_d$ ,  $g_w$ , and  $g_c$ , we have, respectively,  $c_j = 1$ ,  $c_j = w_j$ , and  $c_j = 3 - 2s_j$  for each  $j \in N$ .

The pair  $(\phi_j(s), \gamma_j(s))$  enjoys the following recursive structure, for all  $j \in N$  and  $s \in \{0, \dots, C\}$ :

$$(\phi_j(s), \gamma_j(s)) = \begin{cases} (\phi_{j-1}(s), \gamma_{j-1}(s)) & \text{if } (\phi_{j-1}(s), \gamma_{j-1}(s)) \geq (\phi_{j-1}(s - w_j) + \pi_j^*, \gamma_{j-1}(s - w_j) + c_j) \\ (\phi_{j-1}(s - w_j) + \pi_j^*, \gamma_{j-1}(s - w_j) + c_j) & \text{if } (\phi_{j-1}(s), \gamma_{j-1}(s)) < (\phi_{j-1}(s - w_j) + \pi_j^*, \gamma_{j-1}(s - w_j) + c_j) \end{cases}.$$

For each  $j \in N$ , we assume  $\phi_j(s) = \gamma_j(s) = -\infty$  for every  $s < 0$ , and  $\phi_0(s) = \gamma_0(s) = 0$  for every  $s \in \{0, \dots, C\}$ . Starting from  $s = 1$  and  $j = 1$ , one can use the previous equation to compute, recursively, the values of  $\phi_j(s)$  and  $\gamma_j(s)$  for all combinations of  $j \in N$  and  $s \in \{0, \dots, C\}$ . An optimal lexicographic solution has value  $(\phi_n(C), \gamma_n(C))$ .

For every  $S \subseteq N$ , let  $c(S) := \sum_{j \in S} c_j$ . In the lexicographic case, one can extend the dominance rule between two partial solutions  $S'$  and  $S''$  by defining that  $S'$  dominates  $S''$  if:

$$w(S') = w(S'') \quad \text{and} \quad (\pi^*(S'), c(S')) \geq (\pi^*(S''), c(S'')).$$

The complexity of the Lex-DP algorithm is  $O(nC)$ —the same as for the non-lexicographic case. The only extra computational burden is due to the need for storing the values of

$\gamma$  alongside those of  $\phi$ . As it is shown in the computational results section, for the instances we considered this overhead is completely negligible.

#### 5. Computational experience

We assess the impact of the newly proposed method by measuring the reduction in the number of columns that are generated when solving the FBPP to optimality as well as the reduction in the total computing time spent reoptimizing the RMP and generating columns.

The experiments are run on a single core of an Intel i7-3770 at 3.40 GHz, equipped with 16 GB of RAM and running Ubuntu 14.04. All the algorithms we consider are implemented in C and compiled with the gcc compiler, version 4.8.4. We use the primal simplex algorithm implemented in CPLEX 12.7.0 (in single-threaded mode) to solve the RMP at each iteration of the CG procedure. All its parameters are set to their default value.

We initialize the RMP with  $n$  singleton columns, one per item, and we always generate a single column per CG iteration (as a consequence, in our experiments the number of columns is equal to the number of CG iterations). While different choices (e.g., adopting a heuristic initial set of columns and generating more columns at a time) may speed up the computations, we opt for a clean setting to better assess the impact of our proposed method. This is in line with other investigations in the literature, see [32, 3].

We use a testbed comprising 9 different classes of BPP instances proposed in the literature: Falkenauer T and U, Hard 28, Random, School 1, 2, and 3, Schwerin 1 and 2, and Wäscher. They contain, respectively, 80, 80, 28, 720, 480, 10, 100, 100, and 17 instances, with  $n$  ranging from 57 to 1,000 and  $C$  from 100 to 100,000. We discard 130 easy and small instances of class School 1 since they are all solved by generating less than 100 columns, thus obtaining a testbed of 1,475 instances in total.

We compare four CG algorithms (all generating maximal columns of minimum reduced cost):

1. DP-STD: it employs the nonlexicographic DP algorithm by which the PP is solved so to guarantee the maximality of the resulting column according to Proposition 3.
2. LEX-DENS: it employs the novel LEX-DP algorithm for solving the LPP with  $g(x) = g_d(x)$  (i.e., using *Density* as second-level objective function).
3. LEX-W: it employs the novel LEX-DP algorithm for solving the LPP with  $g(x) = g_w(x)$  (i.e., using *Weight* as second-level objective function).
4. LEX-DIV: it employs the novel LEX-DP algorithm for solving the LPP with  $g(x) = g_c(x)$  (i.e., using *Diversity* as second-level objective function).

All the item indices are permuted once at random before the experiments are run.



**Table 1**

Estimate of the number of times the PP admits two different optimal solutions and of the last iteration in which two different optimal solutions are found.

class	different solutions [%]			last iter. diff. sol. [%]		
	Min	Avg	Max	Min	Avg	Max
Falkenauer T	39.17	57.63	71.34	40.83	69.94	99.16
Falkenauer U	9.08	21.46	52.03	58.22	83.85	99.52
Hard 28	16.48	25.42	41.85	45.16	86.14	99.85
School 1	0.00	46.71	100.00	0.00	80.25	99.96
School 2	1.19	22.43	93.42	14.00	42.80	99.87
School 3	28.00	30.54	32.98	38.25	87.02	98.82
Schwerin 1	14.00	18.30	22.22	22.11	32.92	99.48
Schwerin 2	11.57	18.29	25.69	21.63	36.24	98.62
Wäscher	3.01	10.81	25.48	12.74	37.62	99.45

### 5.1. On the presence of multiple optimal solutions in the PP

We first estimate, computationally, how often the PP admits multiple optimal solutions as a function of the iterations. The experiments are conducted by employing DP-STD at each iteration and, before adding the column it generates to the RMP, generating another column (which is afterwards discarded) with LEX-W. The two columns are then inspected to verify whether they are different. This gives a conservative estimate on the number of times the PP admits multiple optimal solutions (note that there may be multiple optima even if the two columns we produced were equal).

We summarize these results in Table 1 (each entry is averaged over the instances in the corresponding class). The "different solutions [%]" columns report the minimum, average, and maximum number of different solutions found when solving the PP as a percentage of the total number of generated columns. The "last iter. diff. sol. [%]" columns report the earliest, average, and last iteration in which two different solutions to the PP are found as a percentage of the total number of iterations.

The table shows the average percentage of iterations in which different (optimal) solutions are found ranges from 10.81% (achieved on the Wäscher instances) to 57.63% (which is achieved on the Falkenauer T instances). The table also shows that the last iteration (in percentage) in which two different solutions are found ranges, on average, from 32.92% (achieved on the Schwerin 1 instances) to 87.02% (achieved on the School 3 instances). While the table indicates that the PP is, on average, more likely to admit multiple optimal solutions during the early iterations of the CG algorithm, its last column, whose values are all very close to 99%, shows that different solutions can still be found until we are extremely close to convergence.

### 5.2. Comparing the four CG methods

Table 2 illustrates the difference in terms of the total number of columns that are generated to solve the FBPP to optimality when employing the four pricing algorithms. The results are presented in aggregation over each class of

**Table 2**

Average number of columns for DP-STD and percentage variation for LEX-DIV, LEX-W, and LEX-DENS per class of instances.

class	# Cols	Percentage Column Variation		
		DP-STD	LEX-DIV	LEX-W
Falkenauer T	596.5	-4.0	-44.4	0.1
Falkenauer U	1117.0	0.6	-6.5	0.8
Hard 28	760.0	1.8	-5.7	3.8
School 1	611.1	-12.0	-4.2	4.3
School 2	485.1	-1.9	-12.7	-0.2
School 3	618.6	-4.9	-22.7	0.0
Schwerin 1	213.7	1.1	-9.2	-0.1
Schwerin 2	248.0	-5.0	-7.9	-0.1
Wäscher	540.2	0.0	-5.1	2.4

instances using the arithmetic mean. Due to being identical for all the methods, the initial set of columns (one per item) is not taken into account. We report the average number of generated columns for DP-STD and the percentage variation w.r.t. this number for the other three algorithms LEX-DIV, LEX-W, and LEX-DENS. A negative number corresponds to a reduction, whereas a positive number indicates an increase.

Table 2 shows that LEX-DIV allows for a reduction in the number of columns in most of the instances (the only exceptions are the Falkenauer U, Hard 28, and Schwerin 1 classes). While the reduction can be quite large (see the School 1 instances, with a reduction of 12%), the increase in the number of columns does not exceed 1.8% (see the Hard 28 instances).

LEX-DENS leads to an overall increase in the number of columns in almost all the instances (up to 4.3% on the School 1 class), with only three exceptions (the School 2, Schwerin 1, and Schwerin 2 instances). Such improvement is, nevertheless, very modest (the largest, equal to 0.2%, is observed on the School 2 instances).

Differently, LEX-W yields an improvement which can be quite substantial. The average reduction ranges from 4.2% (observed on the School 1 class) to 44.4% (observed on the Falkenauer T class).

On the whole testbed, the total number of columns that are generated is 808,451 for DP-STD, 752,136 for LEX-DIV, 733,644 for LEX-W, and 823,839 for LEX-DENS. W.r.t. DP-STD, LEX-W allows for a reduction in the number of columns by 9.25% on average.

Table 3 illustrates the difference in terms of the total computing time needed to solve the FBPP to optimality with the four CG algorithms. The results are presented as the total time taken to solve all of the instances in each class.

As the table shows, LEX-DIV yields a reduction in computing time on some instances, but also an increase on some others. The largest reduction is found on the School 3 class, where it is equal to 16.5%. The largest increase, by 7.7%, is on the Schwerin 1 class.

LEX-DENS leads to an overall increase in computing time on almost all the instances. The sole exception is the Falkenauer U class, where we observe an improvement, albeit

**Table 3**

Total computing time (in seconds) for DP-STD and percentage variation for LEX-DIV, LEX-W, and LEX-DENS per class of instances.

class	Time (s)		Percentage Time Variation		
	DP-STD	LEX-DIV	LEX-W	LEX-DENS	
Falkenauer T	90.2	1.0	-58.2	2.5	
Falkenauer U	766.7	-5.3	-4.2	-0.4	
Hard 28	18.1	5.4	-4.6	7.0	
School 1	508.0	0.6	-0.9	6.5	
School 2	1672.2	-3.1	-14.8	0.9	
School 3	405.9	-16.5	-35.3	0.4	
Schwerin 1	12.7	7.7	-12.5	0.7	
Schwerin 2	18.2	-6.1	-11.0	2.7	
Wäscher	83.5	-1.6	-5.2	1.7	

**Table 4**

Average computing time (ms) per CG iteration spent solving the RMP and the PP/LPP.

class	DP-STD		LEX-DIV		LEX-W		LEX-DENS	
	RMP	PP	RMP	LPP	RMP	LPP	RMP	LPP
Falkenauer T	0.8	0.4	0.8	0.5	0.5	0.5	0.8	0.5
Falkenauer U	4.6	0.1	4.3	0.1	4.6	0.1	4.6	0.1
Hard 28	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3
School 1	0.7	0.1	0.7	0.1	0.7	0.1	0.7	0.1
School 2	3.2	0.5	3.1	0.5	2.9	0.5	3.2	0.5
School 3	0.6	65.0	0.6	57.0	0.6	54.4	0.6	65.3
Schwerin 1	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.3
Schwerin 2	0.4	0.3	0.4	0.3	0.4	0.3	0.4	0.3
Wäscher	2.2	4.7	2.1	4.7	2.1	4.6	2.2	4.7

very small (by 0.4%).

In line with the reduction in the number of columns, LEX-W yields a reduction in computing time on all the instance classes. The smallest reduction is by 0.9%, observed on the School 1 instances, whereas the largest one is on the Falkenauer T class, where it is equal to 67.1%.

Overall, the results in terms of computing times are in line with those on the number of columns. In particular, the total computing time is 3,575.47 seconds for DP-STD, 3,420.25 seconds for LEX-DIV, 3,085.87 seconds for LEX-W, and 3,628.03 seconds for LEX-DENS. W.r.t. DP-STD, the reduction in computing time obtained with LEX-W is by 13.69% on average.

Table 4 reports the average computing time in milliseconds spent per CG iteration when solving the RMP and the PP/LPP in the four CG algorithms we consider.

The table shows that the time spent solving the PP/LPP is almost comparable over the four algorithms. In particular, it shows that the lexicographic pricing problems do not introduce any measurable overhead in terms of computing time. The same applies to the time spent reoptimizing the RMP, which is comparable over the four methods.

Overall, the three tables confirm that LEX-W allows for a reduction in the total number of columns which is often quite substantial. Due to the fact that the computing times per iteration are almost the same as those for DP-STD, this translates

into a comparable reduction in computing time and, overall, into a faster CG algorithm. Finally, it is worth mentioning that a smaller number of columns also results in a smaller RMP, a feature which may become relevant in a branch-and-price algorithm where a large number of instances of the RMP need to be solved.

When compared to either DP-STD or LEX-DENS, LEX-W generates the smallest number of columns and it takes the smallest computing time in more than 80% of the instances. When compared to LEX-DIV, LEX-W generates the smallest number of columns and takes the smallest computing time in, respectively, slightly less than 80% and slightly more than 70% of the instances.

Overall, our tests show that DP-STD and LEX-DIV are completely dominated by LEX-W and that, when considering LEX-W and LEX-DIV, LEX-W should be preferred in the majority of the cases, confirming that LEX-W is the best method among the four.

### 5.3. Computational illustration of the performance of LEX-W

In Section 3, we have pointed out that feasible solutions to the RMP featuring only columns (patterns) of zero waste are necessarily optimal. We illustrate this property on the following BPP instance with  $n = 15$ ,  $w = (1, 3, 6, 8, 12, 16, 33, 66, 80, 132, 144, 160, 264, 288, 320)$ , and  $C = 511$ . This is one of the smallest BPP instances described in [8] which does not enjoy the integer round-up property (the optimal FBPP solution value is 3, whereas the optimal solution value of the BPP is 4).

All four of the methods we consider produce the same optimal solution. It consists of the following six patterns:  $S_1 = \{1, 3, 9, 12, 13\}$ ,  $S_2 = \{1, 5, 8, 11, 14\}$ ,  $S_3 = \{2, 4, 9, 10, 14\}$ ,  $S_4 = \{6, 7, 8, 10, 13\}$ ,  $S_5 = \{2, 5, 6, 12, 15\}$ ,  $S_6 = \{3, 4, 7, 11, 15\}$ , all having total weight  $g_w$  equal to the bin capacity 511. The patterns are selected with  $\xi_{S_i} = 0.5$  for  $i = 1, \dots, 6$ . In this solution, each item is covered by exactly two patterns.

While the four methods produce the same optimal solution, the sets of columns that they generate over the iterations are different and have different cardinalities. DP-STD generates 47 columns, LEX-DIV 45, LEX-DENS 50, and LEX-W 23. In particular, LEX-W is the only method which manages to obtain a substantial reduction (by about 50%) w.r.t. DP-STD.

Let us now inspect the range of iterations within which each method generates a column which is contained in the final optimal solution. The range is [25, 42] for DP-STD, [23, 42] for LEX-DIV, [23, 46] for LEX-DENS, and [12, 23] for LEX-W. This shows that, for this instance, LEX-W generates columns belonging to an optimal solution much earlier than the other methods.

Let us now consider the whole testbed. For each instance and each CG iteration, we call *basic-column item-weight* the average over all the columns which are basic in that iteration of their total item weight divided by the bin capacity. For each instance, we then divide the CG iterations in 10 percentage intervals. As an example, if the total number of iteration is 100, the first interval is [0, 10), the second is

[10, 20), and so on until the interval [90, 100]. For a given interval, we call *basic-column item-weight index* the average of the basic-column item-weight over all instances and all iterations in that interval.

As our results show, the index achieves quite large values, exceeding 95% already in the [50,60) interval and reaching 98.5% in the interval [90,100]. This indicates that, even if the FBPP asks for minimizing the number of bins directly rather than for maximizing the overall item weight, the item weight of the columns contained in an optimal solution monotonically increases over the iterations and that, the closer we are to convergence, the higher the weight of the basic columns is. This suggests that, by generating columns of minimum reduced cost which are also of maximum weight, we favor the generation of columns which are more likely to be featured in an optimal solution. The experiments also show that, in line with the illustration given above for the numerical BPP instance, in 649 instances out of a total of 1,475 the optimal basic solution we find only contains columns of item weight exactly equal to the bin capacity (i.e., of zero waste and basic-column item-weight index equal to 100%).

#### 5.4. Combination with a stabilization technique

We conclude by experimenting with the combination of one of the classical stabilization techniques in the literature with LEX-W, focusing on the *smoothing* technique proposed in [31] (also see the recent survey [27]). The method is based on solving the PP by using a *smoothed* dual variable vector  $\tilde{\pi}$  rather than  $\pi^*$ , defined as  $\tilde{\pi}_j := \hat{\pi}_j + (1 - \alpha)(\pi_j^* - \hat{\pi}_j)$ ,  $j \in N$ . The vector  $\hat{\pi}$  is the *stability center*: an estimate of an optimal feasible solution to the dual of the FBPP.

We construct  $\hat{\pi}$  along the lines of the *Farley bound* [19, 4], relying on a feasible solution  $\tilde{\pi}$ . Letting  $S^*$  be the (optimal) solution found by solving the pricing problem using  $\tilde{\pi}$  as vector of objective function coefficients, we construct the dual solution  $\tilde{\pi}_j := \frac{\tilde{\pi}_j}{\sum_{j \in S^*} \tilde{\pi}_j}$ ,  $j \in N$ . Since, by construction,  $\sum_{j \in S^*} \tilde{\pi}_j$  is equal to the largest value taken by the left-hand side of the constraints of the dual of the FBPP (which read  $\sum_{j \in S} \pi_j \leq 1$ , for  $S \in \mathcal{S}$ ) for the given  $\tilde{\pi}$ ,  $\tilde{\pi}$  is guaranteed to satisfy all the dual constraints. Starting with  $\hat{\pi} = 0$ , we set  $\hat{\pi} := \tilde{\pi}$  whenever  $\tilde{\pi}$  becomes the new best solution.

Adopting  $\tilde{\pi}$  rather than  $\pi^*$  can result in a so-called *mispricing* step in which the pricing problem does not produce a negative reduced-cost column even when one exists, i.e.,  $\sum_{j \in S^*} \tilde{\pi}_j < 1$ . If this happens, we switch the smoothing technique off and solve the standard pricing problem with a nonsmoothed  $\pi^*$  until convergence.

We combine the smoothing technique of [31] with LEX-W in LEX-W+S, which corresponds to using LEX-W adopting  $\tilde{\pi}$  in lieu of  $\pi^*$  and switching to  $\pi^*$  after the first mispricing step takes place. For comparison purposes, we also experiment with the smoothing technique applied in combination with the standard pricing problem DP-STD. We refer to the resulting method as DP-STD+S. We set  $\alpha = 0.3$ . We performed an extensive set of experiments to determine the best value of  $\alpha$ , measuring the performance of DP-STD+S on the entire data

**Table 5**

Average number of columns and time percentage variation w.r.t. DP-STD obtained with DP-STD+S and LEX-W+S per class of instances.

class	Perc. Column Variation		Perc. Time Variation	
	DP-STD+S	LEX-W+S	DP-STD+S	LEX-W+S
Falkenauer T	-2.3	-44.7	-3.0	-51.8
Falkenauer U	-8.1	-14.6	-19.0	-25.7
Hard 28	-11.7	-16.4	-14.1	-19.0
School 1	-11.8	-15.5	-16.5	-20.2
School 2	-5.1	-16.4	-8.3	-21.1
School 3	-5.3	-26.9	-15.1	-38.3
Schwerin 1	-4.7	-11.9	-6.9	-15.3
Schwerin 2	-4.6	-10.7	-6.6	-14.7
Wäscher	-7.8	-11.2	-13.9	-16.1

set for  $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ . Setting  $\alpha = 0.3$  proved to be the best option, leading to the smallest number of columns and to the smallest computing time on 30% to 40% of the instances.

Table 5 reports the comparison of DP-STD+S and LEX-W+S to DP-STD w.r.t. number of columns and total computing time.

As the table shows, DP-STD+S yields an average reduction in the number of columns ranging from 2.3% (Falkenauer T instances) to 11.8% (School 1 instances). As to the computing time, the average reduction ranges from 3% (Falkenauer T instances) to 19% (Falkenauer U instances).

LEX-W+S yields substantially larger reductions. In terms of the number of columns, the average reduction ranges from 10.7% (Schwerin 2 instances) to 44.7% (Falkenauer T instances). In terms of computing time, it ranges from 14.7% (Schwerin 2 instances) to 51.8% (Falkenauer T instances). When considering all the instances, with LEX-W+S we obtain an average reduction by 18.7% in terms of the number of columns and by 24.7% in terms of computing time.

On the whole testbed, the total number of generated columns is 808,451 for DP-STD, 733,644 for LEX-W, 732,091 for DP-STD+S, and 663,577 LEX-W+S. W.r.t. DP-STD, LEX-W+S allows for a reduction in the number of columns by 17.91% on average. The total computing time is 3,575.47 seconds for DP-STD, 3,085.87 seconds for LEX-W, 3008.52 seconds for DP-STD+S, and 2572.59 seconds for LEX-W+S. W.r.t. DP-STD, the average reduction in computing time obtained for LEX-W+S is 28.04%.

Overall, the experiments suggest that LEX-W can be effectively combined with a preexisting stabilization technique such as the smoothing one proposed in [31], obtaining a combined method which is more effective than either of the two methods used separately.

## 6. Conclusions

We have proposed a lexicographic pricing problem for the FBPP which, among all the maximal columns of minimum reduced cost, generates one which maximizes one of three measures of maximality (density, weight, and diversity), and we have proposed a lexicographic dynamic pro-

gramming algorithm for its solution.

Computational results on a large testbed of instances from the literature suggest that solving a lexicographic pricer is indeed advantageous, and that the adoption of the weight measure allows for a substantial reduction in the number of columns and computing time needed to solve the FBPP to optimality, also when combined with a classical smoothing technique from the literature.

Future work includes the extension of our methods to other problems typically solved with column generation techniques, such as graph coloring and vehicle routing and, in particular, to problems whose pricing subproblem is solved via dynamic programming. It would also be of interest to investigate the adoption of our techniques, with focus on the LEX-W algorithm, to problems with a structure similar to that of the BPP, such as the BPP with conflicts or the BPP with precedence constraints, whose pricing problem enjoys a knapsack-like structure similar to that of the BPP/FBPP.

Another research direction is solving the pricing problem not as a lexicographic optimization problem, but rather as a proper bi-objective problem, exploring the behavior of the overall column generation method as a function of how the trade-off between the two objectives is established. More than two objectives could also be simultaneously considered, either in a multi-objective setting, or in the lexicographic one with three or more levels. Along the lines of [13, 14], techniques based on the notion of bound improvement could also be explored. In line with recent works on sparsity [18, 17] and on our results with dense columns, it could also be of interest to extend our methods to the generation of maximal columns of *minimum* density, whose pricing problem boils down to a lexicographic extension of the *minimum-cost maximal knapsack packing problem* studied in [22].

## References

- [1] Achterberg, T., 2009. SCIP: solving constraint integer programs. *Math. Program. Comput.* 1, 1–41.
- [2] Amaldi, E., Coniglio, S., Gualandi, S., 2010. Improving cutting plane generation with 0-1 inequalities by bi-criteria separation, in: *International Symposium on Experimental Algorithms*, Springer. pp. 266–275.
- [3] Amaldi, E., Coniglio, S., Gualandi, S., 2014. Coordinated cutting plane generation via multi-objective separation. *Mathematical Programming* 143, 87–110.
- [4] Amor, H.B., de Carvalho, J.V., 2005. Cutting stock problems, in: *Column generation*. Springer, pp. 131–161.
- [5] Amor, H.M.B., Desrosiers, J., Frangioni, A., 2009. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics* 157, 1167–1184.
- [6] Balas, E., Saxena, A., 2008. Optimizing over the split closure. *Mathematical Programming* 113, 219–240.
- [7] Belov, G., Scheithauer, G., 2006. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research* 171, 85–106.
- [8] Caprara, A., Dell’Amico, M., Díaz-Díaz, J.C., Iori, M., Rizzi, R., 2015. Friendly bin packing instances without integer round-up property. *Mathematical Programming* 150, 5–17.
- [9] Valério de Carvalho, J., 2002. LP models for bin packing and cutting stock problems. *European Journal of Operational Research* 141, 253–273.
- [10] Valério de Carvalho, J., 2005. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing* 17, 175–182.
- [11] Clautiaux, F., Alves, C., Valério de Carvalho, J., 2010. A survey of dual-feasible and superadditive functions. *Annals of Operations Research* 179, 317–342.
- [12] Coffman Jr, E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D., 2013. Bin packing approximation algorithms: survey and classification, in: *Handbook of combinatorial optimization*. Springer, pp. 455–531.
- [13] Coniglio, S., 2013. Bound-optimal cutting planes, in: *Proc. of 12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pp. 59–62.
- [14] Coniglio, S., Tieves, M., 2015. On the generation of cutting planes which maximize the bound improvement, in: *International Symposium on Experimental Algorithms*, Springer. pp. 97–109.
- [15] Delorme, M., Iori, M., Martello, S., 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* 255, 1–20.
- [16] Desrosiers, J., Lübbecke, M.E., 2005. A primer in column generation, in: *Desaulniers, G., Desrosiers, J., Solomon, M.M. (Eds.), Column Generation*. Springer US, Boston, MA, pp. 1–32.
- [17] Dey, S.S., Molinaro, M., 2018. Theoretical challenges towards cutting-plane selection. *Mathematical Programming* 170, 237–266.
- [18] Dey, S.S., Molinaro, M., Wang, Q., 2015. Approximating polyhedra with sparse inequalities. *Mathematical Programming* 154, 329–352.
- [19] Farley, A.A., 1990. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research* 38, 922–923.
- [20] Fischetti, M., Lodi, A., 2007. Optimizing over the first chvátal closure. *Mathematical Programming* 110, 3–20.
- [21] Frangioni, A., Gendron, B., 2013. A stabilized structured dantzig-wolfe decomposition method. *Mathematical Programming* 140, 45–76.
- [22] Furini, F., Ljubić, I., Sinnl, M., 2017. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research* 262, 438–448.
- [23] Gualandi, S., Malucelli, F., 2012. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing* 24, 81–100.
- [24] Lübbecke, M.E., Desrosiers, J., 2005. Selected topics in column generation. *Operations research* 53, 1007–1023.
- [25] Martello, S., Toth, P., 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, New York.
- [26] du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P., 1999. Stabilized column generation. *Discrete Mathematics* 194, 229–237.
- [27] Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2018. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing* 30, 339–360.
- [28] Rousseau, L.M., 2004. Stabilization issues for constraint programming based column generation, in: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, Springer. pp. 402–408.
- [29] Vanderbeck, F., 2005. Implementing mixed integer column generation, in: *Column generation*. Springer, pp. 331–358.
- [30] Wei, L., Luo, Z., Baldacci, R., Lim, A., 2018. A new branch-and-price-and-cut algorithm for one-dimensional bin packing problems. Accepted for publication on *INFORMS Journal on Computing*, 1–32.
- [31] Wentges, P., 1997. Weighted dantzig-wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research* 4, 151–162.
- [32] Zanette, A., Fischetti, M., Balas, E., 2011. Lexicography and degeneracy: can a pure cutting plane algorithm work? *Mathematical programming* 130, 153–176.