The sharpest column: stabilizing column generation for the bin packing problem via a lexicographic pricer

Stefano Coniglio

University of Southampton
University Road, SO17 1BJ, Southampton, UK

Fabio D'Andreagiovanni

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

Fabio Furini

LAMSADE, Université Paris-Dauphine, 75775 Paris Cedex 16, France

Abstract

In spite of being an extremely successful method to tackle mathematical programs involving a very large number of variables, Column Generation (CG) is known to suffer from stabilization issues which can slow down its convergence significantly. In this article, we propose a new parameter-free stabilization technique for CG based on solving a lexicographic pricing problem. Using the bin packing problem as case study, we show that, computationally, the new stabilization technique allows for a substantial reduction in the number of columns that are generated to solve the problem. While we present the approach for the bin packing problem, our methodology is general and can be applied to other classes of problems solved via CG.

Keywords: bin packing; column generation; stabilization techniques.

Email addresses: s.coniglio@soton.ac.uk (Stefano Coniglio), d.andreagiovanni@hds.utc.fr (Fabio D'Andreagiovanni), fabio.furini@dauphine.fr (Fabio Furini)

1. Introduction

Given a set $N = \{1, \ldots, n\}$ of n items with positive integer weights $w_1, \ldots, w_n$ and a positive integer bin capacity C, consider the *Bin Packing Problem* (BPP) of computing the smallest number of bins that are needed to pack all the items in N. A key problem in combinatorial optimization, $\mathcal{NP}$-hard and very challenging to solve from a computational viewpoint, BPP has been covered by a large body of literature, and it still is heavily studied [1, 2, 3, 4].

1.1. Column generation for BPP

Call *pattern* any set of items $S \subseteq N$ satisfying $\sum_{j \in S} w_j \leq C$ and let $\mathscr{S}$ be the collection of all feasible patterns, i.e., $\mathscr{S} = \left\{ S \subseteq N : \sum_{j \in S} w_j \leq C \right\}$. Let $A \in \{0,1\}^{n \cdot |\mathscr{S}|}$ be a binary matrix with $a_{jS} = 1$ if and only if item $j \in N$ belongs to pattern $S \in \mathscr{S}$, and let e be the all-one vector of n components. Upon introducing a binary variable ξ_S for each pattern $S \in \mathscr{S}$, equal to 1 if and only if pattern S is chosen, one of the most effective *set covering* formulations for BPP, see [4, 5], is the following one, which features exponentially many variables (or columns):

$$\min_{\xi \in \{0,1\}^{|\mathscr{S}|}} \left\{ \sum_{S \in \mathscr{S}} \xi_S : A\xi \geq e \right\}. \tag{1}$$

The formulation calls for the minimum number of patterns covering each item at least once.

Formulation (1) is typically solved by a *Branch-and-Price* (B&P) algorithm based on *Column Generation* (CG) (we refer the interested reader to [6] for further details). Key to this algorithm is solving with CG the Linear Programming (LP) relaxation of Formulation (1), which reads:

$$\min_{\xi \geq 0} \left\{ \sum_{S \in \mathscr{S}} \xi_S : A\xi \geq e \right\}. \tag{2}$$

This LP relaxation is known in the literature as *Fractional Bin Packing Problem* (FBPP). The dual bound provided by FBPP is very tight, as the difference between the optimal solution values of FBPP and BPP is, typically, smaller

or equal to 1—it is indeed conjectured that this may hold in general on every instance of the problem (so-called *modified integer round-up conjecture* [7]).

Let us describe the CG procedure required to solve Formulation (2). Starting from a restriction of Formulation (1) to a subset of columns $\tilde{\mathscr{S}}$ admitting a feasible solution, known as *Restricted Master Problem* (RMP), CG iteratively solves a *Pricing Problem* (PP) to generate a column with a strictly negative reduced cost to be added to $\tilde{\mathscr{S}}$. RMP is then reoptimized, and the procedure is iterated until no more columns with negative reduced cost are found.

Letting $\pi^* \in \mathbb{R}^n_+$ be a vector of optimal dual variables of RMP, PP can be formulated as the following 0-1 *knapsack problem*, where $x_j = 1$ if and only if item $j \in N$ is selected in the new pattern:

$$(PP) \quad \max_{x \in \{0,1\}^n} \left\{ \sum_{j \in N} \pi^*_j \, x_j : \sum_{j \in N} w_j \, x_j \leq C \right\}. \tag{3}$$

Let x^* be an optimal solution to (PP). If $\sum_{j \in N} \pi^*_j \, x^*_j > 1$, the pattern $S^* = \{j \in N : x^*_j = 1\}$ is added to $\tilde{\mathscr{S}}$ and the procedure is reiterated; otherwise, x^* itself constitutes a proof of optimality of RMP and the CG algorithm halts.

We remark that, as the dual of Formulation (2) contains exponentially many constraints $\sum_{j \in S} \pi_j \leq 1$, $S \in \mathscr{S}$, generating a column of minimum reduced cost $(1 - \sum_{j \in S^*} \pi^*_j)$ corresponds to separating, in the dual of RMP, an inequality of maximum violation (see Section 1.2 for further insights on this important equivalence).

Since PP is weakly $\mathcal{NP}$-hard, solving FBPP is also weakly $\mathcal{NP}$-hard due to the equivalence between optimization and separation [8]. This is consistent with the nature of PP which, being an instance of the 0-1 knapsack problem, is solvable in pseudopolynomial time $O(nC)$ by *Dynamic Programming* (DP).

1.2. The issue of stabilization

CG is known to suffer from a number of computational issues, including *dual oscillations*, *tailing-off effects*, and *primal degeneracy* of the master problem. In particular, the tailing off effect is often responsible for poor computational convergence, with the bound improvement becoming smaller and smaller the closer

we get to an optimal solution. The effect is even more pronounced for large problems subject to degeneracy. While this phenomenon can be partially explained in several ways, it is believed that the main reason lie in the unstable behavior of the dual variables [9, 10]. After the seminal work of [11], several stabilization techniques have been proposed in the literature [12, 13, 14]. Notably, though, they all suffer from the need for parameter tuning, see [10].

1.3. Contribution of the paper

Due to the connection between column generation in the primal and cutting plane generation in the dual, the large body of works on the separation of "good" cuts (sometimes called *sharpest cuts* [15]) can be extended to the column generation framework.

Inspired by the work in [16] on coordinated cutting plane generation (and in line with the very important stream of research on stabilization in CG), we propose a parameter-free stabilization technique based on a *Lexicographic Pricing Problem* (LPP), aiming at generating what one may call the *sharpest column*.

Among all columns with minimum reduced cost, our LPP is designed to find one which also maximizes another lower-priority objective among *density*, *diversity*, and *weight* (see Section 2). Crucially, we will show that, for BPP, LPP can be solved with an adaptation to the lexicographic setting of a classical DP method for the solution of the 0-1 knapsack problem.

We remark that the stabilization technique that we propose here is quite general, and it can be applied to other problems besides BPP.

2. A lexicographic pricing problem

It is not hard to realize that, very often, PP admits many optimal solutions, all of them achieving a minimum reduced cost. This is clear when π^* is sparse, which is often the case as, due to complementary slackness, $\pi^*_j = 0$ whenever $\sum_{S \in \mathscr{S}: j \in S} \xi_S > 1$. In such cases, the presence of a multitude of optimal solutions

to PP can be exploited, driving the CG method towards the generation of columns/patterns enjoying special properties.

2.1. Maximal columns

Let us consider the following definition:

Definition 1. *A pattern $S \in \mathscr{S}$ is called* maximal *if $S \cup \{j\} \notin \mathscr{S}$, $\forall j \in N \setminus S$.*

By inspecting the dual of Formulation (2), 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 can be ignored, greatly reducing the size of $\mathscr{S}$.

The arguably simplest way to achieve maximality is applying a greedy a *a posteriori* "maximalization" procedure which, after PP is solved, adds (in some order) items $j \in N$ with $\pi_j^* = 0$ to the generated pattern until either the capacity C is saturated or no other item can be added.

In practice, PP not only admits many columns with a minimum reduced costs, but also many maximal columns with a minimum reduced cost (see Example 1). The natural question is then: *should some of these maximal columns be preferred to the other ones?* We address it in the remainder of the paper.

2.2. The lexicographic pricer

In this paper, we consider the maximality aspect as a part of a lexicographic optimization procedure which, among all columns of minimum reduced cost, finds one which maximizes a lower-priority objective function by which maximality is achieved.

In more detail, we propose to solve the following *Lexicographic Pricing Problem* (LPP):

$$(LEX-PP) \qquad \max_{x\in\{0,1\}^n} \left\{ \big(f(x), g(x)\big) : \sum_{j\in N} w_j x_j \le C \right\},$$

where $f(x) = \sum_{j\in N} \pi_j^* x_j$ and $g(x)$ is a lower-priority objective function to be optimized hierarchically, among all solutions which minimize $f(x)$.

We consider the following three options for $g(x)$. In all three cases, it is easy to see that columns which maximize $g(x)$ are necessarily maximal.

1. *Density.* By maximizing the function $g(x) = \|x\|_1 = \sum_{j \in N} x_j = |S|$, one aims at generating columns maximizing the number of items in the pattern.
2. *Weight.* By maximizing the function $g(x) = \sum_{j \in N} w_j x_j$, one aims at generating columns maximizing the weight of the items in the pattern. Maximizing this function corresponds to minimizing the *waste* of the pattern, i.e., the empty space that is left in the bin that is about to be generated. The rationale behind this measure is that, arguably, columns with large weight are more likely to be contained in an optimal solution. Note that maximizing this function in the second level corresponds to solving a *subset sum problem* asking for a subset of items of maximum weight fitting in the bin.
3. *Diversity* or *coordination.* Let $\tilde{\mathscr{S}}$ be the set of columns contained in RMP at a given column generation iteration, and consider the average number of times item $j \in N$ is covered by the columns in $\tilde{\mathscr{S}}$, namely:

$$0 \leq \tilde{s}_j = \frac{|\tilde{\mathscr{S}}_j|}{|\tilde{\mathscr{S}}|} \leq 1, \quad \text{where} \quad \tilde{\mathscr{S}}_j = \left\{ S \subseteq \tilde{\mathscr{S}} : j \in S \right\}.$$

Maximizing, for some $\gamma \geq 0$, the function $\|x - \tilde{s}\|_1 + \gamma \|x\|_1$, one maximizes, with trade-off factor γ, a combination of the 1-norm distance $\|x - \tilde{s}\|_1$ between the column x being generated and the average $\tilde{s}$ of the previous columns and the density $\|x\|_1$ of x. With $\gamma > 1$, one can show that only maximal columns will be produced. The reason why diverse columns may be beneficial in CG is similar to why diverse cuts may be beneficial in cutting plane generation. We refer the interested reader to [16], where the function was originally proposed.

Note that, due to x being binary, $\|x - \tilde{s}\|_1 + \gamma \|x\|_1$ can be rewritten as:

$$\|x - \tilde{s}\|_1 + \gamma \|x\|_1 = \sum_{j \in N} x_j - 2 \sum_{j \in N} \tilde{s}_j x_j + \sum_{j \in N} \tilde{s}_j + \gamma \sum_{j \in N} x_j. \quad (4)$$

After dropping the constant term $\sum_{j \in N} \tilde{s}_j$ and letting $\gamma = 2$ as suggested in [16], we have: $g(x) = \sum_{j \in N} (3 - 2\tilde{s}_j)\, x_j$.

While, given a column x maximizing any of these three function, the value of any of them cannot be increased by adding items to x, maximizing the three quantities leads to different columns:

Example 1. *Consider an instance with $n = 6$ items with $w = (50, 8, 9, 49, 26, 25)$ and $C = 100$. Assume that RMP contains seven patterns: all 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 $\tilde{s} = \left(\frac{2}{7}, \frac{2}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{7}, \frac{2}{7}\right)$. We have $3e - 2\tilde{s} = \left(\frac{17}{7}, \frac{17}{7}, \frac{17}{7}, \frac{19}{7}, \frac{19}{7}, \frac{17}{7}\right)$. Consider, for instance, the following three maximal patterns of minimum reduced cost (equal to -1):*

pattern	*density*	*weight*	*diversity*
$S_8 = \{1, 2, 3, 5\}$	*4*	*93*	*10*
$S_9 = \{2, 3, 4, 5\}$	*4*	*92*	$\frac{72}{7}$
$S_{10} = \{4, 5, 6\}$	*3*	*100*	$\frac{55}{7}$

As it is clear, S_{10} is of maximum weight, but not of maximum density nor diversity. S_8 and S_9 are both of maximum density, but not of maximum weight. Note also how the diversity measure further differentiates between S_8 and S_9, as only S_9 achieves maximum diversity.

Note that one can always solve PP in lexicographic sense by optimizing the function $f(x) + \epsilon g(x)$ for a sufficiently small $\epsilon > 0$, see [16]. For large instances, though, a value of ϵ small enough to guarantee that the lexicographic priority be respected can be very small. If ϵ is smaller than machine precision, the presence of the lower-priority function $g(x)$ could be nullified, *de facto* reverting the lexicographic to a classical single-level one. To overcome this numerical limitation, we propose the following generalization of the DP algorithm typically used to solve the 0-1 knapsack problem corresponding to the standard PP.

2.3. Lexicographic Dynamic Programming

We propose a *Lexicographic Dynamic Programming* (LDP) algorithm for the solution of LPP which is entirely parameter independent.

For each item $j \in N$ and integer value $s \in \{0, \ldots, C\}$, let $\big(f_j(s), g_j(s)\big)$ be the optimal lexicographic solution value in terms of f and g. Let the binary operator $\succeq$ induce a partial order over pairs $\big(\big(f_j(s), g_j(s)\big), \big(f_{j'}(s'), g_{j'}(s')\big)\big)$ for all $j, j' \in N$ and $s, s' \in \{0, \ldots, C\}$, such that $\big(f_j(s), g_j(s)\big) \succeq \big(f_{j'}(s'), g_{j'}(s')\big)$ if an only if either $f_j(s) > f_{j'}(s')$ or $f_j(s) = f_{j'}(s')$ and $g_j(s) \geq g_{j'}(s')$.

Let $g(x) = \sum_{j \in N} c_j x_j$ for some $c \in \mathbb{R}^n$. For the three cases of density, weight, and diversity, we have, respectively, $c_j = 1$, $c_j = w_j$, and $c_j = 3 - 2\tilde{s}_j$, for all $j \in N$. One can see that $\big(f_j(s), g_j(s)\big)$ admits the following recursive structure:

$$\begin{pmatrix} f_j(s) \\ g_j(s) \end{pmatrix} = \begin{cases} \begin{pmatrix} f_{j-1}(s) \\ g_{j-1}(s) \end{pmatrix} & \text{if } \begin{pmatrix} f_{j-1}(s) \\ g_{j-1}(s) \end{pmatrix} \succeq \begin{pmatrix} f_{j-1}(s - w_j) + \pi_j^* \\ g_{j-1}(s - w_j) + c_j \end{pmatrix} \\ \begin{pmatrix} f_{j-1}(s - w_j) + \pi_j^* \\ g_{j-1}(s - w_j) + c_j \end{pmatrix} & \text{if } \begin{pmatrix} f_{j-1}(s) \\ g_{j-1}(s) \end{pmatrix} \prec \begin{pmatrix} f_{j-1}(s - w_j) + \pi_j^* \\ g_{j-1}(s - w_j) + c_j \end{pmatrix}. \end{cases} \tag{5}$$

An optimal solution to LEX-PP can thus be obtained by computing the values of $\big(f_j(s), g_j(s)\big)$ recursively with Equation (5), starting from $s = 1$ and $j = 1$, and initializing $f_0(s) = 0$ and $g_0(s) = 0$ for all $s \in \{0, \ldots, C\}$. The complexity of the algorithm is clearly $O(nC)$, the same as for the classical DP algorithm for the 0-1 knapsack problem.

3. Computational experience

After clarifying the aim of our experimental campaign and presenting the test bed of instances that we use, we illustrate the computational results that we obtain with the lexicographic pricing problem that we propose.

3.1. Aim of the experiments and setup

We assess the impact of our lexicographic stabilization technique by measuring the reduction in the number of columns that are generated when solving

FBPP to optimality. This is in line with the setup of other works on cutting plane generation [17, 16, 18], and better allows us to assess the impact of the different methods that we consider in a cleaner setting devoid of the many, often conflicting, aspects of B&P (such as primal heuristics and branching rules). In particular, we explicitly refrain from reporting the computing times here. We remark, though, that the time it takes to solve our lexicographic pricing problem LEX-PP is comparable to that taken by solving the standard pricing problem PP.

3.2. Experimental Settings

We use `CPLEX 12.7.0` (just called `CPLEX`, for brevity, in what follows) in single-threaded mode to solve the LP relaxation of RMP at each iteration of the CG procedure. All `CPLEX` parameters are set to their default value, except for `EPAGAP` and `EPOPT`, which are set to 1^{-9} (maximal tolerance) to obtain, from a numerical point of view, results as accurate as possible. RMP is reoptimized using `CPLEX`'s dual simplex algorithm. To have a clean set of experiments, we initialize RMP with singleton columns, one per item.

3.3. Instances

We use a test bed of 540 instances[1] proposed in [19, 20] (called in the literature: set "Scholl 1"). They are obtained with the following input settings: $|N| \in \{100, 200, 500\}$, $C \in \{100, 120, 150\}$, and w_j ranging in $[1, 100]$, $[20, 100]$, and $[30, 100]$ for $j = 1, \ldots, n$ (for brevity, we denote these three ranges by $W \in \{1, 2, 4\}$. See Table 1. The item weights are sampled as integer numbers from a uniform random distribution defined over those intervals. In this test bed, 20 instances are generated for each of the 27 instance classes using the settings above, for a total of 540 instances of different characteristics.

[1] Data set 1 from `https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.html`.

3.4. Results

We compare five different pricing algorithms for the solution of FBPP (see Formulation (2)):

1. `STD`: standard textbook implementation where PP is solved using `CPLEX` and Formulation (3).
2. `STD-MAX`: same setting as `STD` with an *a posteriori* maximalization phase in which items are added until the capacity is saturated. The greedy maximalization is performed using a *random* item order.
3. `LEX-DENS`: the LEX-PP pricing problem is solved as a two level lexicographic DP, using the `Density` $g(x)$ objective function.
4. `LEX-DIVER`: the LEX-PP pricing problem is solved as a two level lexicographic DP, using the `Diversity` $g(x)$ objective function.
5. `LEX-WEIGHT`: the LEX-PP pricing problem is solved as a two level lexicographic DP, using the `Weight` $g(x)$ objective function.

In all cases, further ties are broken at random.

A graphical representation of the relative performance of the five different CG pricing algorithms presented above is given by the performance profile in Figure 1.

For each instance, let τ be the ratio between the number of columns generated by the considered pricing algorithm and the minimum number of columns generated, on that instance, by all the pricing algorithms we tested.

For each pricing algorithm and for each value of τ in the horizontal axis, the vertical axis reports the percentage of the instances for which the algorithm at hand requires, for the solution of FBPP, at most τ times the number of columns that are generated by the algorithm which generates the smallest amount. To better clarify the meaning of the vertical axis, let us illustrate Figure 1 for $\tau = 1.2$, i.e., when each algorithm is allowed to generate at most 1.2 times more columns than the one generating the smallest quantity: for both `STD` and `STD-MAX` this happens in about 40% of the instances, whereas for `LEX-DENS`,

LEX-DIVER, and LEX-WEIGHT this happens in about 85, 90 and 95% of the instances, respectively. Each curve starts from the percentage of instances in which the corresponding algorithm generates the minimum number of columns. Overall, the best performances are graphically represented by the curves in the upper part of Figure 1.

The graph in Figure 1 clearly shows that LEX-WEIGHT achieves the best performance, as it is the pricing algorithm which generates the minimum number of columns in almost 60% of the instances. The second best algorithm is LEX-DIVER, outperforming the other ones in almost 30% of the instances, followed by LEX-DENS, which outperforms the other ones in 10% of the instances. The performance profile shows that STD-MAX achieves only very marginal improvements over STD, and it is dominated by the three LEX algorithms. It is worth noticing that by allowing 2 times the number of columns generated by the best lexicographic configurations, both STD and STD-MAX are only able to solve around 80% of the instances, managing to solve 100% of the them only when they are allowed 6 times the number of columns.

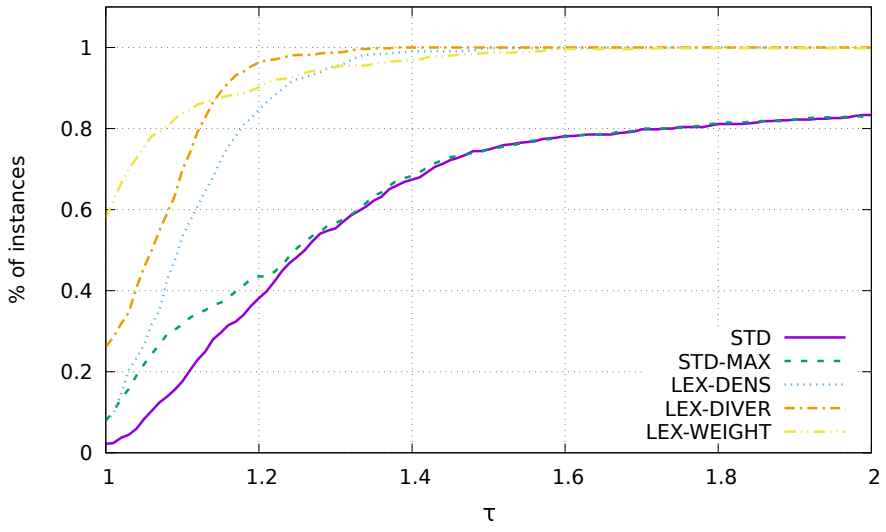

Figure 1: Performance profile in numbers of generated columns for the five CG pricing algorithms considered.

In Table 1, we report detailed computational results on the 540 benchmark instances, obtained solving FBPP via the five different pricing algorithms. The best pricing algorithm for each class of instances is reported in boldface. Each line of the table shows the average results of 20 instances with similar features. The first three columns of the table report the instance characteristics, i.e., the number of items $|N|$, the capacity C, and the distribution of the weights W. The column "`STD` # cols" reports the average number of columns generated by the pricing algorithm `STD`. These figures are then used to evaluate the performance of all the other pricing algorithms. The last four columns of the tables report the average percentage reduction in the number of columns of the other four pricing algorithms we tested. For each pricing algorithm, the reduction is measured as the percentage ratio between the difference in the number of columns generated by the algorithm at hand and `STD`, over the number of columns generated by the latter.

Table 1 clearly shows that all the pricing algorithms based on the lexicographic pricing problem (LEX-PP) lead to a considerable reduction in the number of columns. The average reduction is of -19.2%, -20.9%, and -23.8% for, respectively, `LEX-DENS`, `LEX-DIVER` and `LEX-WEIGHT`.

Table 1 also indicates that the *a posteriori* maximalization procedure carried out in `STD-MAX` is not effective, as it leads to an average column reduction of only -1.7%. This shows the effectiveness of the lexicographic approach of selecting, among all the columns with a minimum reduced cost, one which maximizes the measures of maximality we proposed.

As far at the comparison between the different lexicographic pricing problems and algorithms is concerned, Table 1 shows that the best one is `LEX-WEIGHT`. Indeed, this pricing algorithm always leads to an average reduction in the number of columns for all classes of instances in the test bed, with a minimum reduction of -3.1% and a maximum reduction of -73.6%. From the table, it also emerges that the most substantial reductions can be achieved for the instances with $W = 4$, i.e., instances with relatively large item weights and, in general, for the the larger instances with 500 items. We remark that, for the 20 instances with

$N = 500$, $C = 100$, and $W = 4$, `LEX-WEIGHT` achieves, on average, a reduction in the number of columns of -73.6%, which translates into an average saving of ≈ 1400 columns per instance.

4. Conclusions

We have proposed a lexicographic pricing problem for column generation which, among all columns of minimum reduced cost which are maximal, generates one which maximizes either the column density, its weight, or a measure of diversity w.r.t. the columns that have been previously generated.

Computational results on bin packing instances show a substantial reduction in the number of columns needed to solve FBPP to optimality, especially when column weight is maximized as the second-level lexicographic objective.

We remark that our approach can be easily generalized to any number of lexicographic objectives, so to allowing for breaking any further tie according to different metrics. Moreover, our new lexicographic pricing technique can be applied seamlessly to any B&P algorithm, also on top of other stabilization techniques, at no computational overhead, as our lexicographic pricing problem retains the same theoretical and practical complexity as the standard done.

Future work includes the extension to other problems typically solved with column generation techniques, such as graph coloring or vehicle routing and, in particular, to problems whose pricing subproblem is solved via dynamic programming. In line with recent work on sparsity [21], it can also be of interest to extended our results to the generation of a maximal column of *minimum* density—in the case of bin packing, this corresponds to solving, in the second lexicographic level, the *minimum-cost maximal knapsack packing problem* studied in [22].

References

[1] E. G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: survey and classification, in: Handbook of

			STD	Percentage Column Reduction			
$\|N\|$	C	W	# cols	STD-MAX	LEX-DENS	LEX-DIVER	LEX-WEIGHT
100	100	1	218.3	-2.7	0.3	1.3	**-7.0**
		2	159.4	-1.8	-14.9	**-18.8**	-16.3
		4	169.5	2.6	-48.5	**-58.8**	-49.2
	120	1	261.1	-4.2	-2.2	1.2	**-8.5**
		2	229.2	1.1	-11.7	-12.7	**-15.8**
		4	259.9	-0.1	-37.4	**-44.1**	-38.3
	150	1	297.9	-13.0	-9.3	-9.0	**-18.3**
		2	301.9	-1.6	-10.7	-11.2	**-19.3**
		4	243.0	-1.0	-14.2	-14.1	**-14.7**
200	100	1	499.9	-2.1	-0.6	1.0	**-4.5**
		2	365.3	1.4	-18.3	**-22.1**	-20.3
		4	499.0	-0.1	-60.1	**-67.5**	-61.7
	120	1	627.1	-3.5	0.6	1.8	**-7.2**
		2	501.8	-2.2	-16.4	**-17.2**	-17.1
		4	582.0	0.8	-41.1	**-44.9**	-38.3
	150	1	621.3	-10.5	-6.4	-6.8	**-15.3**
		2	600.2	-0.2	-9.8	-10.7	**-19.6**
		4	540.3	-0.1	-14.7	-17.4	**-25.0**
500	100	1	1392.7	-2.3	0.2	3.3	**-3.1**
		2	1101.1	-0.1	-26.9	**-28.6**	-28.2
		4	1890.3	1.0	-72.1	**-76.8**	-73.6
	120	1	1759.1	-4.3	-1.4	0.0	**-6.0**
		2	1451.5	0.5	-18.8	-19.5	**-21.8**
		4	2064.5	4.9	-54.5	**-56.8**	-52.1
	150	1	1599.9	-9.6	-3.8	-3.8	**-12.4**
		2	1427.5	0.1	-10.3	-13.0	**-21.3**
		4	1368.1	0.5	-15.3	-18.5	**-28.0**
Avg				-1.7	-19.2	-20.9	-23.8
Min				-13.0	-72.1	-76.8	-73.6
Max				4.9	0.6	3.3	-3.1

Table 1: Percentage reduction of the number of columns generated to solve FBPP. Each line reports the average over 20 instances with the same features.

combinatorial optimization, Springer, 2013, pp. 455–531.

[2] J. Valério de Carvalho, Lp models for bin packing and cutting stock problems, European Journal of Operational Research 141 (2) (2002) 253–273.

[3] F. Clautiaux, C. Alves, J. Valério de Carvalho, A survey of dual-feasible and superadditive functions, Annals of Operations Research 179 (1) (2010) 317–342.

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

[5] G. Belov, G. Scheithauer, A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, European Journal of Operational Research 171 (1) (2006) 85 – 106.

[6] J. Desrosiers, M. E. Lübbecke, A Primer in Column Generation, Springer US, Boston, MA, 2005, pp. 1–32.

[7] A. Caprara, M. Dell'Amico, J. C. Díaz-Díaz, M. Iori, R. Rizzi, Friendly bin packing instances without integer round-up property, Mathematical Programming 150 (1) (2015) 5–17.

[8] M. Grötschel, L. Lovász, A. Schrijver, Geometric algorithms and combinatorial optimization, Vol. 2, Springer Science & Business Media, 2012.

[9] M. Lübbecke, J. Desrosiers, Selected topics in column generation, Operations Research 53 (6) (2005) 1007–1023.

[10] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, Automation and combination of linear-programming based stabilization techniques in column generation, INFORMS Journal on Computing 30 (2) (2018) 339–360.

[11] O. du Merle, D. Villeneuve, J. Desrosiers, P. Hansen, Stabilized column generation, Discrete Mathematics 194 (1) (1999) 229 – 237.

[12] J. Valério de Carvalho, Using extra dual cuts to accelerate column generation, INFORMS Journal on Computing 17 (2) (2005) 175–182.

[13] A. Frangioni, B. Gendron, A stabilized structured dantzig–wolfe decomposition method, Mathematical Programming 140 (1) (2013) 45–76.

[14] H. M. B. Amor, J. Desrosiers, A. Frangioni, On the choice of explicit stabilizing terms in column generation, Discrete Applied Mathematics 157 (6) (2009) 1167 – 1184.

[15] M. Grötschel, The Sharpest Cut, Society for Industrial and Applied Mathematics, 2004.

[16] E. Amaldi, S. Coniglio, S. Gualandi, Coordinated cutting plane generation via multi-objective separation, Mathematical Programming 143 (1-2) (2014) 87–110.

[17] A. Zanette, M. Fischetti, E. Balas, Lexicography and degeneracy: can a pure cutting plane algorithm work?, Mathematical programming 130 (1) (2011) 153–176.

[18] S. Coniglio, M. Tieves, On the generation of cutting planes which maximize the bound improvement, in: International Symposium on Experimental Algorithms, Springer, 2015, pp. 97–109.

[19] A. Scholl, R. Klein, C. Jürgens, Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem, Computers & Operations Research 24 (7) (1997) 627 – 645.

[20] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Chichester, New York, 1990.

[21] S. S. Dey, M. Molinaro, Q. Wang, Approximating polyhedra with sparse inequalities, Mathematical Programming 154 (1-2) (2015) 329–352.

[22] F. Furini, I. Ljubić, M. Sinnl, An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem, European Journal of Operational Research 262 (2) (2017) 438–448.