New developments in the primal-dual column generation technique

Jacek Gondzio* Pablo González-Brevis† Pedro Munari‡

School of Mathematics, University of Edinburgh
The King's Buildings, Edinburgh, EH9 3JZ, UK
Technical Report ERGO 11-001,
January 24, 2011.

Abstract

The classical column generation is based on optimal solutions of the restricted master problems. This strategy frequently results in an unstable behaviour and may require an unnecessarily large number of iterations. To overcome this weakness, variations of the classical approach use interior points of the dual feasible set, instead of optimal solutions. In this paper, we address the primal-dual column generation technique, which relies on well-centred non-optimal solutions of the restricted master problems that are obtained by a primal-dual interior point method. Although good computational results are reported for this technique, it was only applied in a particular class of problems. Moreover, no theoretical analysis to guarantee its convergence is available. Here, we further investigate the primal-dual column generation technique and present extensive computational experiments in the context of integer programming, where column generation schemes are widely employed. The results show that the primal-dual technique usually leads to substantial reductions in the number of iterations as well as less running time when compared to the classical and also analytic centre approaches.

Keywords: interior point methods, column generation, linear programming.

1 Introduction

The difficulty in solving general integer programming problems is a well-known issue. Different approaches have been proposed in the last 50 years, but the existence of a general-purpose algorithm with polynomial complexity is still an open question. Currently, the efficient approaches usually consist of the combination of several techniques. For instance, many problems are

*School of Mathematics, University of Edinburgh, Scotland, United Kingdom. Email: J.Gondzio@ed.ac.uk.

†School of Mathematics, University of Edinburgh, Scotland, United Kingdom. Email: P.Gonzalez-Brevis@sms.ed.ac.uk. Partially supported by Beca Presidente de la República, Chile and on leave from Facultad de Ingeniería, Universidad del Desarrollo, Chile.

‡Instituto de Ciências Matemáticas e de Computação, University of São Paulo, São Carlos, Brazil. Email: munari@icmc.usp.br. Partially supported by CAPES and FAPESP, Brazil.

successfully solved by combining a branch-and-bound search with column and row generation procedures as well as specialized heuristics.

In this paper, we are concerned with an essential tool for integer programming: the column generation technique [29]. This technique is an iterative procedure applied to solve a linear programming problem with columns of its coefficient matrix generated by following a known rule. In the context of integer programming, the linear programming problems usually arise when a decomposition or relaxation technique is applied to an integer programming formulation, although the column generation method is independent of these techniques. Due to the duality relationships, column generation is equivalent to Kelley's cutting plane method [27] and, hence, we use the terms *column generation* and *cutting plane* in an interchangeable way.

Given a linear programming problem with a huge number of columns (variables), we call it *master problem* (MP). To solve this problem in an efficient way, the column generation technique considers a reduced version of it, called *restricted master problem* (RMP), in which only a few columns of the MP are considered at first. By solving the RMP, a dual solution is obtained and used in a set of one or more subproblems to generate new columns. These columns are added to the RMP so that a better approximation of the MP is achieved. The process is repeated until no more attractive columns can be generated and, hence, the optimal solution of the RMP is also optimal for the associated MP.

In a standard column generation procedure, every RMP is solved to optimality. Several drawbacks of this approach are reported in the literature (see [29] for a survey) and, hence, different strategies are used in practice. More efficient column generation procedures usually rely on interior points of the dual feasible set of the RMP, so that stable dual solutions are provided to the subproblems [22, 33, 17, 4]. In some of them, an interior point method is used to obtain non-optimal solutions that are well-centred in the dual feasible set. For instance, in the primal-dual column generation technique proposed in [22], the tolerance used to solve the RMP is dynamically adjusted in function of the relative gap. In the first iterations of the column generation procedure, there is no reason to be close to the optimality of the RMP, as it usually corresponds to merely a rough approximation of the MP. Hence, a loose tolerance is used to obtain a suboptimal solution of the RMP. As the gap in the column generation approaches zero, a better approximation of the MP is available and, therefore, a smaller tolerance (higher accuracy) should be used to solve the RMP. Substantial reductions in the number of iterations can be achieved by using this strategy (see [22, 32]).

Although good computational results are reported for the primal-dual column generation technique [22], it was only applied in a particular class of problems. Moreover, a theoretical analysis that guarantees the convergence of this method is not available in the literature. The purpose of this paper is to further investigate this technique and also to present computational experiments in the context of integer programming, where column generation schemes are widely employed. As a contribution, we present a new theoretical analysis of the method as well as an extensive computational study using instances from the literature. We have selected three classes of problems which are classical in the literature of column generation: the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). These problems are known to lead to degenerate

restricted master problems and, hence, causing instability and leading to the tailing-off effect in the classical column generation [3, 25, 9].

The structure of the remaining sections in this paper is the following. In Section 2, we present the main concepts in column generation and establish the notation for the remaining sections. The primal-dual column generation technique is described in Section 3, with new theoretical developments. In Section 4, we describe the extended Dantzig-Wolfe decomposition, since it is applied in the classes of problems considered here. In Section 5, a computational study comparing the primal-dual approach to other two column generation techniques is presented. The conclusions and future research are presented in Section 6.

2 Column generation technique

We are concerned with solving a linear programming problem, called the master problem (MP), which is represented as

$$z^\star := \min \quad \sum_{j \in N} c_j \lambda_j, \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{j \in N} a_j \lambda_j = b, \tag{2.1b}$$

$$\lambda_j \geq 0, \qquad \forall j \in N, \tag{2.1c}$$

where $N = \{1, \ldots, n\}$ is a set of indices, $\lambda = (\lambda_1, \ldots, \lambda_n)$ is the column vector of decision variables, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $a_j \in \mathbb{R}^m$, $\forall j \in N$. We assume that the MP has a huge number of variables which makes solving this problem a very difficult task. Furthermore, we assume the columns a_j are not given explicitly but are implicitly represented as elements of a set $\mathcal{A} \neq \varnothing$, and they can be generated by following a known rule. To solve the MP, we consider only a small subset of columns at first, which leads to the restricted master problem (RMP):

$$z_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{2.2a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{2.2b}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{2.2c}$$

for some $\overline{N} \subseteq N$. Any primal feasible solution $\bar{\lambda}$ of the RMP corresponds to a primal feasible solution $\hat{\lambda}$ of the MP, with $\hat{\lambda}_j = \bar{\lambda}_j$, $\forall j \in \overline{N}$, and $\hat{\lambda}_j = 0$, otherwise. Hence, the optimal value of any RMP gives an upper bound of the optimal value of the MP, *i.e.*, $z^\star \leq z_{RMP}$.

The column generation technique consists in an iterative process where we solve the RMP and use the obtained optimal solution to generate one or more new columns. Then, we modify the RMP by adding the generated column(s) and repeat the same steps until we can guarantee that no more columns are necessary. At the end of the process, we obtain an optimal solution to the associated MP.

Natural questions arise at this point: (a) how to check whether no more columns are necessary? and (b) how to generate new columns to be added to the RMP? The answers to both

questions are given by the *oracle*. The oracle is composed of one or more (pricing) subproblems, which are able to generate new columns by using a dual solution of the RMP. The idea behind the oracle is to check if a dual solution of the RMP is feasible for the MP.

Let $u = (u_1, \ldots, u_m)$ be the vector of dual variables associated to constraints (2.1b) of the MP. For any given pair $(\overline{\lambda}, \overline{u})$ of primal-dual solution, we assume that $\overline{\lambda}$ is a primal feasible solution. We can check the feasibility of the dual variables in the MP by using the reduced costs $s_j = c_j - \overline{u}^T a_j$, for each $j \in N$. If $s_j < 0$ for some $j \in N$, then the dual solution $\overline{u}_j$ is not feasible and, therefore, $\overline{\lambda}$ cannot be optimal. Otherwise, if $s_j \geq 0$ for all $j \in N$ and $b^T\overline{u} = c^T\overline{\lambda}$, then an optimal solution of the MP has been found.

Since we have assumed that columns a_j do not have to be explicitly available, we should avoid computing the values s_j for all $j \in N$. Instead, we use the minimum among them, obtained by solving the subproblem

$$z_{SP} := \min\{c_j - \overline{u}^T a_j | a_j \in \mathcal{A}\}. \tag{2.3}$$

For simplicity, we reset $z_{SP} := 0$ when $z_{SP} > 0$. In some applications, the subproblem (2.3) can be partitioned into several independent subproblems that provide different types of columns. In this case, z_{SP} corresponds to the sum of the smallest reduced costs of each subproblem.

The value z_{SP} is called the *value of the oracle*. If $z_{SP} = 0$, we can ensure that there is no negative reduced cost and, hence, an optimal solution of the MP has been obtained. Otherwise, the column a_j corresponding to the minimal reduced cost should be added to the RMP. At this point, more than one column may be found and we can add one or more of them to the RMP. Actually, any column with a negative reduced cost can be added to the RMP. However, by using (2.3) we can provide a lower bound of the optimal value of the MP, if we know a constant κ such that

$$\kappa \geq \sum_{i \in N} \lambda_i^\star, \tag{2.4}$$

where $\lambda^\star = (\lambda_1^\star, \ldots, \lambda_n^\star)$ is an optimal solution of the MP. Indeed, we cannot reduce z_{RMP} by more than κ times z_{SP} and, hence, we have

$$z_{RMP} + \kappa z_{SP} \leq z^\star \leq z_{RMP}. \tag{2.5}$$

The value of κ is promptly available when the extended Dantzig-Wolfe method is applied to obtain the column generation scheme. This will be clarified in Section 4.

The column generation terminates when both bounds in (2.5) are the same, *i.e.*, $z_{SP} = 0$. We refer to the number of RMPs solved as *outer* iterations. The number of iterations to solve a given RMP is called *inner* iterations.

2.1 Column generation strategies

In the classical column generation, the RMP is solved to optimality at each outer iteration. However, this approach is usually affected by the unstable behaviour of optimal dual solutions, which can lead to a large number of outer iterations. Different strategies have been proposed to overcome this weakness. For instance, the stabilization techniques choose a dual point called

stability centre and modify the RMP by adding penalization terms and/or artificial variables [31, 11, 5, 4]. The modified RMP is solved to optimality, but the dual solutions are kept relatively close to the stability centre and, hence, the variations in subsequent dual solutions are reduced. Good computational results are reported for these techniques, although they are dependent of appropriate choices of the stability centre and penalization terms, which can be difficult in practice. For performance comparisons involving stabilized approaches and the classical column generation, see [5, 4].

An important observation is that solving the RMP to optimality is not needed in a column generation procedure. Hence, strategies relying on dual solutions corresponding to interior points of the dual feasible set have been proposed [22, 33, 17, 38]. The purpose of these approaches is to avoid the oscillation observed in optimal dual solutions and, hence, to reduce the number of outer iterations when compared to the classical approach.

In [33, 34], the authors address the solution of two classes of combinatorial optimization problems by a cutting plane method which uses interior points of the dual set, obtained by a primal-dual interior point method. In those particular applications, the valid inequalities are explicitly known in advance, and for each dual solution of the RMP, the violated inequalities are found by full enumeration. If the violation is not large enough, then the tolerance is updated and the interior point method continues with the optimization of the RMP. Notice that this approach cannot be directly applied in the general context of column generation, as usually the columns cannot be fully enumerated, but are rather generated by solving a high time-consuming problem (NP-hard in many cases).

The primal-dual column generation proposed in [22] is a more general interior point approach. In this strategy, a primal-dual interior point method is used to find a non-optimal solution of the RMP, whose distance to optimality is defined in function of the relative gap. In the first outer iterations, each RMP is solved with a loose tolerance, and this tolerance is dynamically reduced throughout the iterations as the gap in the column generation approaches zero. The authors present promising computational results for a class of nonlinear programming problems, whose linearization is solved by column generation. A similar strategy is used in [32] to solve linear programming problems by combining Dantzig-Wolfe decomposition and a primal-dual interior point method. The authors also report a substantial reduction in the number of outer iterations when compared to other column generation procedures. To the best of our knowledge, these strategies have never been applied in the context of integer programming, where column generation schemes are widely employed.

An interior point column generation based on the simplex method is proposed in [38]. At each outer iteration, in order to obtain a dual solution in the interior of the dual space, the dual problem associated to the RMP is solved several times using randomly generated objective functions. Then, a set of vertices of the dual space is generated and an interior dual point is given by the convex combination of the points in the set. The authors present computational results considering instances of the VRPTW, for which the number of outer iterations and CPU time were reduced in relation to the classical as well as stabilized column generation. However, for applications with large-scale RMPs, the need for solving these problems several times for different objective values adversely affects the efficiency of the approach.

A very efficient interior point approach is given by the analytic centre cutting plane method (ACCPM) [16, 1, 17]. The strategy consists in computing a dual point which is an approximate analytic centre of the localization set associated to the current RMP. The localization set is given by the intersection of the dual space of the RMP with a half-space given by the best lower bound found for the optimal dual value of the MP. Relying on points in the centre of the localization set, usually leads to a small difference between dual solutions of two subsequent outer iterations and also contributes to the generation of deeper cuts. A very important property of this approach is given by its theoretical fully polynomial complexity. Although other polynomial cutting plane methods are proposed in the literature, no efficient computational implementations are publicly available for them (see [35]).

3 Primal-dual column generation

Proposed in [22], the primal-dual column generation method (PDCGM) is based on non-optimal solutions of the RMPs. A primal-dual interior point method is employed to solve the RMPs, which makes possible obtaining primal-dual feasible solutions which are well-centred in the feasible set, but have a nonzero distance to optimality.

The PDCGM approach differs essentially from the classical and the analytic centre approaches. Those techniques can be seen as extremal, as they are based on optimal solutions. In the classical column generation we solve each RMP to optimality, while in the analytic centre technique we solve a modified version of the RMP, which gives an approximation of the analytic centre of the corresponding localization set. The idea of the primal-dual column generation technique places it somewhere in the middle of these two approaches. It relies on solutions that are close-to-optimality, but at the same time not far from the analytic centre of the dual feasible set. The contribution of using non-optimal solutions is twofold. First, a smaller number of inner iterations is needed to solve each RMP and, hence, the CPU time per outer iteration is reduced. Second, a more stable column generation strategy is obtained and, as a result, smaller number of outer iterations as well as less total CPU time are usually required.

3.1 Theoretical background

Following the notation of Section 2, we consider that a given RMP is represented by (2.2), with optimal primal-dual solution $(\overline{\lambda}, \overline{u})$. Similarly to the classical approach, the primal-dual column generation starts with an initial RMP with enough columns to avoid an unbounded solution. However, at a given outer iteration, a suboptimal feasible solution $(\tilde{\lambda}, \tilde{u})$ of the current RMP is obtained, which is defined as follows.

Definition 3.1 *A primal-dual feasible solution $(\tilde{\lambda}, \tilde{u})$ of the RMP is called suboptimal solution, or ε-optimal solution, if it satisfies $(c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|)$, for some tolerance $\varepsilon > 0$.*

We denote by $\tilde{z}_{RMP} = c^T\tilde{\lambda}$ the objective value corresponding to the suboptimal solution $(\tilde{\lambda}, \tilde{u})$. Since $c^T\tilde{\lambda} \geq c^T\overline{\lambda} = z_{RMP}$, $\tilde{z}_{RMP}$ is a valid upper bound of the optimal value of the MP.

The solution $(\tilde{\lambda}, \tilde{u})$ should also be well-centred in the primal-dual feasible set, in order to provide a more stable dual information to the oracle. We say a point (λ, u) is well-centred if it

satisfies

$$\gamma\mu \leq (c_j - u^T a_j)\lambda_j \leq (1/\gamma)\mu, \quad \forall j \in \overline{N}, \tag{3.1}$$

for some $\gamma \in (0.1, 1]$, where $\mu = (1/|\overline{N}|)(c^T - u^T A)\lambda$. By imposing (3.1), we guarantee that the point is not too close to the boundary of the primal-dual feasible set and, hence, the oscillation of the dual solutions will be relatively small. Notice that (3.1) is a natural way of stabilizing the dual solutions, if a primal-dual interior point method is used to solve the RMP [44].

Once the suboptimal solution of the RMP is obtained, the oracle is called with the dual solution $\tilde{u}$ as a query point. Then, it should return either a value $\tilde{z}_{SP} = 0$, if no columns could be generated from the proposed query point, or a value $\tilde{z}_{SP} < 0$, together with one or more columns to be added to the RMP. Consider the value $\kappa > 0$ defined as (2.4) in Section 2. As already mentioned before, a suitable value for κ is usually promptly available in a column generation scheme. According to Lemma 3.2, a lower bound of the optimal value of the MP can still be obtained.

Lemma 3.2 *Let $\tilde{z}_{SP}$ be the value of the oracle corresponding to the suboptimal solution $(\tilde{\lambda}, \tilde{u})$. Then, $\kappa\tilde{z}_{SP} + b^T\tilde{u} \leq z^\star$.*

Proof. Let $\lambda^\star$ be an optimal primal solution of the MP. By using (2.1b), (2.3) and $\tilde{z}_{SP} \leq 0$, we have that

$$\begin{aligned} c^T\lambda^\star - b^T\tilde{u} &= \sum_{j\in N} c_j\lambda_j^\star - \sum_{j\in N} \lambda_j^\star a_j^T\tilde{u} \\ &= \sum_{j\in N} \lambda_j^\star(c_j - a_j^T\tilde{u}) \\ &\geq \sum_{j\in N} \lambda_j^\star\tilde{z}_{SP} \\ &\geq \kappa\tilde{z}_{SP}. \end{aligned}$$

Therefore, $z^\star = c^T\lambda^\star \geq \kappa\tilde{z}_{SP} + b^T\tilde{u}$. □

The tolerance ε which controls the distance of $(\tilde{\lambda}, \tilde{u})$ to optimality can be loose at the beginning of the column generation process, as a very rough approximation of the MP is known at this time. This tolerance should be reduced throughout the outer iterations, and be tight when the gap is small. Hence, we can dynamically adjust it by using the relative gap in the outer iterations, given by

$$gap = \frac{c^T\tilde{\lambda} - (\kappa\tilde{z}_{SP} + b^T\tilde{u})}{1 + |c^T\tilde{\lambda}|}.$$

At the end of every outer iteration, we recompute the relative gap, and the tolerance ε is updated as

$$\varepsilon = \min\{\varepsilon_{\max},\ gap/D\}, \tag{3.2}$$

where $D > 1$ is the *degree of optimality* that relates the tolerance ε to the relative gap. Here, we consider it is a fixed parameter. Also, an upper bound $\varepsilon_{\max}$ is used so that the suboptimal solution is not far away from the optimum.

It is important to emphasize that unlike in the classical approach $\tilde{z}_{SP} = 0$ does not suffice to terminate the column generation process. Indeed $(\tilde{\lambda}, \tilde{u})$ is a feasible but suboptimal solution and therefore there may still be a difference between $c^T\tilde{\lambda}$ and $b^T\tilde{u}$. Lemma 3.3 shows that the gap is still reduced in this case, and the progress of the algorithm is guaranteed.

Lemma 3.3 *Let $(\tilde{\lambda}, \tilde{u})$ be the suboptimal solution of the RMP, found at iteration k with tolerance $\varepsilon^k > 0$. If $\tilde{z}_{SP} = 0$, then the new relative gap is strictly smaller than the previous one, i.e., $gap^k < gap^{k-1}$.*

Proof. We have that $\tilde{z}_{RMP} = c^T\tilde{\lambda}$ is an upper bound of the optimal solution of the MP. Also, from Lemma 3.2 we obtain the lower bound $b^T\tilde{u}$, since $\tilde{z}_{SP} = 0$. Hence, the gap in the current iteration is given by

$$gap^k = \frac{c^T\tilde{\lambda} - b^T\tilde{u}}{1 + |c^T\tilde{\lambda}|}.$$

Notice that the right-hand side of this equality is less than or equal to ε^k, the tolerance used to obtain $(\tilde{\lambda}, \tilde{u})$. Hence, $gap^k \leq \varepsilon^k$. We have two possible values for ε^k. If $\varepsilon^k = \varepsilon_{max}$, then by (3.2) $gap^{k-1} \geq D\varepsilon^k > \varepsilon^k$. Otherwise, $\varepsilon^k = gap^{k-1}/D$ and, again, $gap^{k-1} > \varepsilon^k$. Therefore, we conclude $gap^k < gap^{k-1}$. □

Algorithm 1 summarizes the above discussion. Notice that the primal-dual column generation method has a simple algorithmic description, similar to the classical approach. Thus, it can be implemented in the same level of difficulty if a primal-dual interior point solver is readily available. Notice that κ is known in advance and problem dependent.

Algorithm 1: Primal-Dual Column Generation Method

1. **Input:** Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
2. **set** LB $= -\infty$, UB $= \infty$, $gap = \infty$, $\varepsilon = 0.5$;
3. **while** $(gap \geq \delta)$ **do**
4. find a well-centred ε-optimal solution $(\tilde{\lambda}, \tilde{u})$ of the RMP;
5. UB $= \min(\text{UB}, \tilde{z}_{RMP})$;
6. call the oracle with the query point $\tilde{u}$;
7. LB $= \max(\text{LB}, \kappa\tilde{z}_{SP} + b^T\tilde{u})$;
8. $gap = (\text{UB} - \text{LB})/(1 + |\text{UB}|)$;
9. $\varepsilon = \min\{\varepsilon_{\max},\ gap/D\}$;
10. if $(\tilde{z}_{SP} < 0)$ then add the new columns to the RMP;
11. **end**(while)

Since the PDCGM relies on suboptimal solutions of each RMP, it is important to ensure that it is a valid column generation procedure, *i.e.*, a finite iterative process that delivers an optimal solution of the MP. This result is given in Theorem 3.4.

Theorem 3.4 *Let $z^\star$ be the optimal value of the MP. Given $\delta > 0$, the primal-dual column generation method converges in a finite number of steps to a primal feasible solution $\hat{\lambda}$ of the MP with objective value $\tilde{z}$ that satisfies*

$$(\tilde{z} - z^\star) < \delta(1 + |\tilde{z}|). \tag{3.3}$$

Proof. Consider an arbitrary iteration k of the primal-dual column generation method, with corresponding suboptimal solution $(\tilde{\lambda}, \tilde{u})$. After calling the oracle, two situations may occur:

1. $\tilde{z}_{SP} < 0$ and new columns have been generated. These columns correspond to dual constraints of the MP that are violated by the dual point $\tilde{u}$. Since the columns are added to the RMP, the corresponding dual constraints will not be violated in the next iterations. Therefore, it guarantees the progress of the algorithm. Also, this case can only happen a finite number of times, as there are a finite number of columns in the MP.

2. $\tilde{z}_{SP} = 0$ and no columns have been generated. If additionally we have $\varepsilon^k < \delta$, then from Lemma 3.3 the gap in the current iteration satisfies $gap^k < \delta$, and the algorithm terminates with the suboptimal solution $(\tilde{\lambda}, \tilde{u})$. Otherwise, we also know from Lemma 3.3 that the gap is still reduced, and although the RMP in the next iteration will be the same, it will be solved to a tolerance $\varepsilon^{k+1} < \varepsilon^k$. Moreover, the gap is reduced by a factor of $1/D$ and, hence, after a finite number of iterations we obtain a gap less than δ.

At the end of the iteration, if the current gap satisfies $gap^k < \delta$, then the algorithm terminates and we have

$$\frac{\tilde{z}_{RMP} - (\tilde{z}_{SP} + b^T\tilde{u})}{1 + |\tilde{z}_{RMP}|} < \delta.$$

Since $\tilde{z}_{SP} + b^T\tilde{u} \leq z^\star$, the inequality (3.3) is satisfied with $\tilde{z} = \tilde{z}_{RMP}$. The primal solution $\tilde{\lambda}$ leads to a primal feasible solution of the MP, given by $\hat{\lambda}_j = \tilde{\lambda}_j$, $\forall j \in \overline{N}$, and $\hat{\lambda}_j = 0$, otherwise. If $gap^k \geq \delta$, a new iteration is carried out and we have one of the above situations again. □

3.2 Remarks about implementation

Having presented the theoretical analysis of the PDCGM, it is important to give some remarks about its implementation. As requested by (3.1), the suboptimal solutions are well-centred points in the primal-dual feasible set. This contributes to the stabilization of the dual points and, hence, reduces the number of outer iterations in general. In our implementation, each RMP is solved by the interior point solver HOPDM [18]. It keeps the iterates inside a neighbourhood of the central path, which has the form (3.1). To achieve this, the solver makes use of multiple centrality correctors [19, 7].

An efficient warmstarting technique is essential for a good performance of a column generation technique based on a interior point method, as the PDCGM. Throughout the column generation process, closely-related problems are solved, as the RMP in a given iteration differs from the RMP of the previous iteration by merely a few columns. Hence, this similarity should be exploited in order to reduce the computational effort of solving a sequence of problems. In our implementation of PDCGM, we rely on the warmstarting techniques available in the

solver HOPDM (see [19, 20, 21]). The main idea of these methods consists of storing a close-to-optimality and well-centred iterate when solving a given RMP. After a modification is carried out on the RMP, the stored point is used as a good initial point to start from.

Notice that a primal-dual interior point method is well-suited for the implementation of the PDCGM. In fact, (standard) simplex type methods cannot straightforwardly provide suboptimal solutions which are well-centred in the dual space. Instead, the primal and dual solutions will always be on the boundaries of their corresponding feasible sets. Besides, there is no control on the infeasibilities of the solutions before optimality is reached in a simplex method.

4 Dantzig-Wolfe decomposition

In the classes of problems addressed in this paper, the column generation schemes are obtained by applying extended Dantzig-Wolfe decomposition to the corresponding integer programming formulations. In this section, we briefly describe the fundamental concepts of this approach.

The Dantzig-Wolfe decomposition (DWD) is a technique proposed for linear programming problems with a special structure in the coefficient matrix. The original aim of this technique was to make large linear problems tractable as well as to speed up the solution by the simplex method [8]. Except for some classes of problems, the DWD was not advantageous for general linear programming problems. However, it showed to be very successful when extended to integer programming problems (see [2, 42]). In this context, the focus was to provide stronger bounds when solving linear relaxations in order to speed up a branch-and-bound search.

4.1 DWD for integer programming

Similar to the continuous case, the extended DWD is applied to integer programming formulations that have a special structure in the coefficient matrix. Usually, the matrix is very sparse and composed by several blocks which would be independent except for the existence of a set of linking constraints. Consider the following (original) integer programming problem:

$$\min \quad c^T x, \tag{4.1a}$$

$$\text{s.t.} \quad Ax = b, \tag{4.1b}$$

$$x \in \mathcal{X}, \tag{4.1c}$$

where $\mathcal{X} = \{x \in \mathbb{Z}^n_+ : Dx = d\}$ is a discrete set, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $D \in \mathbb{R}^{h \times n}$, $d \in \mathbb{R}^h$. Let us assume that without the presence of constraint (4.1b), the problem would be easily solved by taking advantage of the structure of $\mathcal{X}$, particularly of matrix D.

To apply extended DWD, we consider the convexification approach, although alternative approaches can be used as well (see [29]). To this end, we consider the convex hull of the set $\mathcal{X}$, denoted by $\mathcal{C} = conv(\mathcal{X})$. Assume that we know the sets of all extreme points p_q and extreme rays p_r that fully represent $\mathcal{C}$. Hence, we can write any $x \in \mathcal{C}$ as

$$x = \sum_{q \in \mathcal{Q}} \lambda_q p_q + \sum_{r \in \mathcal{R}} \mu_r p_r,$$

where the sets $\mathcal{Q}$ and $\mathcal{R}$ consist of indices of all extreme points and extreme rays of $\mathcal{C}$, respectively. By using this equality in problem (4.1), we obtain the equivalent formulation:

$$\min \quad \sum_{q\in\mathcal{Q}} \lambda_q(c^T p_q) + \sum_{r\in\mathcal{R}} \mu_r(c^T p_r), \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} \lambda_q(Ap_q) + \sum_{r\in\mathcal{R}} \mu_r(Ap_r) = b, \tag{4.2b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{4.2c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{4.2d}$$

$$x = \sum_{q\in\mathcal{Q}} \lambda_q p_q + \sum_{r\in\mathcal{R}} \mu_r p_r, \tag{4.2e}$$

$$x \in \mathbb{Z}^n_+. \tag{4.2f}$$

Notice that we still need to keep $x \in \mathbb{Z}^n_+$ in order to guarantee the equivalence between (4.2) and (4.1). However, relaxing the integrality of x in (4.2) usually leads to a lower bound that is the same as or stronger than the one obtained by the linear programming relaxation of (4.1). For this reason, the relaxation of (4.2) is very important when solving (4.1) by a branch-and-bound approach.

Assume we have relaxed the integrality on x. Thus, there is no need to keep the constraints (4.2e) in the formulation of (4.2). By denoting $c_j = c^T p_j$ and $a_j = Ap_j$, $\forall j \in \mathcal{Q}$ and $\forall j \in \mathcal{R}$, a *relaxation* of the problem (4.2) is given by:

$$\min \quad \sum_{q\in\mathcal{Q}} c_q\lambda_q + \sum_{r\in\mathcal{R}} c_r\mu_r \tag{4.3a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} a_q\lambda_q + \sum_{r\in\mathcal{R}} a_r\mu_r = b, \tag{4.3b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{4.3c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{4.3d}$$

which is called Dantzig-Wolfe master problem (DW-MP). Under the column generation perspective, instead of representing the DW-MP by every extreme point p_q, $q \in \mathcal{Q}$, and every extreme ray p_r, $r \in \mathcal{R}$, we consider only a subset of them, for some $\overline{\mathcal{Q}} \subseteq \mathcal{Q}$ and $\overline{\mathcal{R}} \subseteq \mathcal{R}$.

Usually, the set $\mathcal{X}$ can be represented as the Cartesian product of K independent sets, due to a special structure in the matrix D that allows it to be partitioned in several independent submatrices D^k, $k = 1, \ldots, K$. Let us define $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_K$, where

$$\mathcal{X}_k = \{x^k \in \mathbb{Z}^{|L_k|}_+ : D^k x^k = d^k\}, \quad \forall k = 1, \ldots, K,$$

where $|L_k|$ is the number of variables associated to $\mathcal{X}_k$, and x^k is the vector containing the components of x associated to $\mathcal{X}_k$. For simplicity, we assume the set $\mathcal{X}$ is bounded and, hence, $\mathcal{R} = \varnothing$, although the following discussion can be extended to deal with unbounded cases (see

[36]). Following the same ideas as described so far, the DW-MP can be rewritten as:

$$\min \quad \sum_{k=1}^{K} \sum_{q \in \mathcal{Q}_k} c_q^k \lambda_q^k \tag{4.4a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} \sum_{q \in \mathcal{Q}_k} a_q^k \lambda_q^k = b, \tag{4.4b}$$

$$\sum_{q \in \mathcal{Q}_k} \lambda_q^k = 1, \qquad \forall k = 1, \ldots, K, \tag{4.4c}$$

$$\lambda_q^k \geq 0, \qquad \forall q \in \mathcal{Q}_k,\ \forall k = 1, \ldots, K, \tag{4.4d}$$

where the extreme points of the subset $\mathcal{X}_k$ are represented by each p_q with $q \in \mathcal{Q}_k$. Now, K independent subproblems are obtained and rather than adding only one column to the RMP at each outer iteration, K columns can be added. In fact, if we denote by u and v the dual variables associated to constraints (4.4b) and (4.4c), respectively, we have the following subproblem for each $k = 1, \ldots, K$:

$$\begin{aligned} z_{SP}^k &:= \min \left\{ c_q^k - u^T a_q^k - v_k \mid q \in \mathcal{Q}_k \right\}, \\ &= \min \left\{ (c^k - (A^k)^T u)^T x^k - v_k \mid x^k \in \mathcal{X}_k \right\}, \end{aligned}$$

where A^k are the columns in A associated to the variables x^k, $k = 1, \ldots, K$.

There are some applications in which the K subproblems are identical hence they will generate the same columns for a given dual point $\overline{u}$. We can avoid this undesirable situation by using an aggregation of variables

$$\lambda_q := \sum_{k=1}^{K} \lambda_q^k.$$

As a consequence, we can drop the index k from Q_k and denote it simply by Q, since all Q_k represent the same set. The same simplification may be applied to the parameters c_q^k and a_q^k. Considering all these changes together, we can rewrite problem (4.4) as the following *aggregated* master problem:

$$\min \quad \sum_{q \in \mathcal{Q}} c_q \lambda_q \tag{4.5a}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{Q}} a_q \lambda_q = b, \tag{4.5b}$$

$$\sum_{q \in \mathcal{Q}} \lambda_q = K, \tag{4.5c}$$

$$\lambda_q \geq 0, \qquad \forall q \in \mathcal{Q}. \tag{4.5d}$$

Although similar to the DW-MP, there is now only one subproblem associated to the aggregated master problem, which is given by any z_{SP}^k, since they are identical. If $0 \in \mathcal{X}_k$ and its associated

cost is also zero, then the equality in constraint (4.5c) can be relaxed to

$$\sum_{q \in \mathcal{Q}} \lambda_q \leq K.$$

Besides, if K is sufficiently large, then this inequality holds strictly in the optimal solution and, hence, (4.5c) can be dropped from the problem.

It is noteworthy that any solution of the subproblem that has a negative reduced cost can lead to an attractive column of the MP. Hence, if the method applied to solve the subproblem is able to find the best t-solutions, for a given $t > 0$, then we can generate up to t columns, instead of only one. It is usually a good strategy, as it reduces the number of outer iterations. In practice, there must be a compromise between the number of columns added to the RMP at each iteration and the CPU time to solve it.

4.2 Equivalence to Lagrangian relaxation

The dual problem associated to the DW-MP (4.3) has the same form as the problem we obtain by applying Lagrangian relaxation for integer programming in the original problem (4.1) (see [13]). To see this, we associate a vector of Lagrange multipliers u to constraints (4.1b), and use them to penalize the violation of these constraints. Recall that $\mathcal{C}$ denotes the convex hull of $\mathcal{X}$. We define the Lagrangian subproblem as

$$\begin{aligned} L_D(u) &= \min_{x \in \mathcal{R}^n} \left\{ c^T x - u^T (Ax - b),\ x \in \mathcal{X} \right\} \\ &= \min_{x \in \mathcal{R}^n} \left\{ c^T x - u^T (Ax - b),\ x \in \mathcal{C} \right\} \\ &= u^T b + \min_{x \in \mathcal{R}^n} \left\{ (c^T - u^T A) x,\ x \in \mathcal{C} \right\}. \end{aligned}$$

For an arbitrary vector $\overline{u}$, we obtain a lower bound for the optimal value of the problem (4.1) by solving $L_D(\overline{u})$. The best lower bound we can obtain is given by the Lagrangian dual problem

$$\mathcal{L} := \max_{u \in \mathcal{R}^m} L_D(u).$$

By representing the elements of $\mathcal{C}$ by its extreme points p_q and extreme rays p_r, with $q \in \mathcal{Q}$ and $r \in \mathcal{R}$, we can rewrite $\mathcal{L}$ as the following linear programming problem

$$\begin{aligned} \mathcal{L} = \max \quad & u^T b + v \\ \text{s.t.} \quad & u^T A p_q + v \leq c^T p_q, \quad \forall q \in \mathcal{Q}, \\ & u^T A p_r \leq c^T p_r, \quad \forall r \in \mathcal{R}, \end{aligned}$$

which is the dual problem of the DW-MP (4.3). It shows the relationship between DWD and Lagrangian relaxation. Furthermore, if we consider solving the above problem by using the Kelley's cutting plane method, we start with subsets $\mathcal{Q}' \in \mathcal{Q}$ and $\mathcal{R}' \in \mathcal{R}$, and generate the remaining constraints iteratively, by recurring to the Lagrangian subproblem $L_D(u)$. This row generation in the dual space is equivalent to the column generation in the primal space.

5 Computational experiments

In this section, we show the results of computational experiments for three classes of problems from the literature of column generation. They are the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). These problems are known to be very degenerate, hence, causing instability and leading to the tailing-off effect in the classical approach.

One interesting fact about the selected classes is that column generation techniques were originally proposed in a intuitive way for these problems [30, 14, 15, 10]. A few years later, with a better understanding of this subject, the approaches were proved equivalent (or similar) to applying a Dantzig-Wolfe decomposition to an integer programming formulation [41, 3, 25, 9].

5.1 Implemented strategies

In order to carry out the computational experiments presented here, we have implemented three different column generation strategies for each application. They are:

- Classical column generation (CCG): each RMP is solved to optimality by the simplex method available on the commercial solver CPLEX [23]. At each iteration, the current RMP uses the optimal basis of the previous RMP as a warmstarting, which is provided by the solver.

- Primal-dual column generation (PDCGM): the suboptimal solutions of each RMP are obtained by using the interior point solver HOPDM [18], which is able to efficiently provide well-centred dual points.

- Analytic centre cutting plane (ACCPM): the dual point at each iteration is an approximate analytic centre of the localization set associated to the current RMP. The applications were implemented on top of the open-source solver OBOE/COIN [6], a state-of-the-art implementation of the analytic centre strategy.

For each application, the subproblems were solved using the same source-code for all the strategies. Also, CCG and PDCGM are initialized with the same columns and, hence, have the same initial RMP. ACCPM requires an initial dual point to be initialized, instead of a set of initial cuts. After preliminary tests we chose initial points that led to a better performance of the method on average. We have used different initial points for each application, as will be specified later. All codes were run on the same computer with processor Intel Core 2 Duo 1.66 Ghz, 1GB RAM, and Linux operating system. The default accuracy, δ, has been set to 10^{-6}.

Remark In the remainder of this section, we have adopted a compact representation of vectors, for clarity purposes. For a given set $I = \{1, \ldots, n\}$ of indices, we denote by $[x_{ij}]_{i,j \in I}$ the vector $(x_{11}, x_{12}, \ldots, x_{ij}, \ldots, x_{n(n-1)}, x_{nn})$.

5.2 Cutting stock problem

The one-dimensional CSP consists in determining the smallest number of rolls of width W that have to be cut in order to satisfy the demands d_j of pieces of width w_j, for $j \in M = \{1, 2, \ldots, m\}$

[3]. We assume there is an upper bound n on the number of rolls needed to satisfy the demands and, hence, we associate an index in the set $N = \{1, 2, \ldots, n\}$ to each roll. A mathematical formulation originally proposed in [26] is given by:

$$\min \quad \sum_{i \in N} y_i, \tag{5.1a}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} \geq d_j \qquad \forall j \in M, \tag{5.1b}$$

$$\sum_{j \in M} w_j x_{ij} \leq W y_i \qquad \forall i \in N, \tag{5.1c}$$

$$y_i \in \{0, 1\}, \qquad \forall i \in N, \tag{5.1d}$$

$$x_{ij} \geq 0 \text{ and integer}, \qquad \forall i \in N, \forall j \in M, \tag{5.1e}$$

where $y_i = 1$ if the roll i is used, and 0 otherwise. The number of times a piece of width w_j is cut from roll i is denoted with x_{ij}. Constraints (5.1b) guarantee that all demands must be satisfied, and constraints (5.1c) enforce that the sum of the widths of all pieces cut from a roll does not exceed its width W.

5.2.1 Dantzig-Wolfe decomposition

The coefficient matrix of problem (5.1) has a special structure with coupling constraints given by (5.1b), which is well-suited to the application of the extended DWD. Consider the set $\mathcal{X}$ of all points that satisfy the constraints (5.1c), (5.1d) and (5.1e). Following the discussion presented in Section 4.1, we define the subsets $\mathcal{X}_i$, for each $i \in N$, which are independent to each other and satisfy $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$. Furthermore, we replace each $\mathcal{X}_i$ by its convex hull $conv(\mathcal{X}_i)$, which is a bounded set and hence can be fully represented by the set of its extreme points. For each $i \in N$, let P_i be the set of indices of all extreme points of $conv(\mathcal{X}_i)$. These extreme points are then denoted by $(y_p^i, x_{p1}^i, \ldots, x_{pm}^i)$, for each $p \in P_i$. Following this notation, we have the master problem:

$$\min \quad \sum_{i \in N} \sum_{p \in P_i} y_p^i \lambda_p^i, \tag{5.2a}$$

$$\text{s.t.} \quad \sum_{i \in N} \sum_{p \in P_i} x_{pj}^i \lambda_p^i \geq d_j, \qquad \forall j \in M, \tag{5.2b}$$

$$\sum_{p \in P_i} \lambda_p^i = 1, \qquad \forall i \in N, \tag{5.2c}$$

$$\lambda_p^i \geq 0, \qquad \forall i \in N, \forall p \in P_i. \tag{5.2d}$$

Let $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_n)$ be the dual variables associated to constraints (5.2b) and (5.2c), respectively. The oracle corresponding to the master problem (5.2) is given by a set of n subproblems ($i \in N$) of the form

$$\min \quad y_i - \sum_{j \in M} \overline{u}_j x_{ij} - \overline{v}_i, \tag{5.3a}$$

$$\text{s.t.} \quad (y_i, x_{i1}, \ldots, x_{im}) \in conv(\mathcal{X}_i), \tag{5.3b}$$

where $\overline{u}$ and $\overline{v}$ represent an arbitrary dual solution. Since the stock pieces are identical, the subproblems are the same for every $i \in N$, and hence the oracle will generate n equal columns. To avoid this, we apply the aggregation of variables (see Section 4.1 for details). The resulting master problem is given by:

$$\min \quad \sum_{p \in P} y_p \lambda_p, \tag{5.4a}$$

$$\text{s.t.} \quad \sum_{p \in P} x_{pj} \lambda_p \geq d_j, \qquad \forall j \in M, \tag{5.4b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{5.4c}$$

where P represents the set of indices of all extreme points of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 = \ldots = \mathcal{X}_n$. Also, we dropped the convexity constraint, as $\mathbf{0} \in conv(\bar{\mathcal{X}})$ and n is an inactive upper bound in the constraint $\sum_{p \in P} \lambda_p \leq n$. The oracle is now given by only one subproblem, which is the same as (5.3), except for dropping the index i and the variable v_i. To solve the subproblem we first solve a knapsack problem given by

$$\max \quad \sum_{j \in M} \overline{u}_j x_j, \tag{5.5a}$$

$$\text{s.t.} \quad \sum_{j \in M} w_j x_j \leq W, \tag{5.5b}$$

$$x_j \geq 0 \text{ and integer}, \qquad \forall j \in M. \tag{5.5c}$$

An optimal solution $(x_1^\star, \ldots, x_m^\star)$ of this subproblem is used to generate a column of (5.4). If $1 - \sum_{j \in M} \overline{u}_j x_j^\star < 0$, then the column is generated by setting $y_p := 1$ and $x_{pj} := x_j^\star$ for all $j \in M$. Otherwise, we assume the solution is given by an empty pattern and, hence, the column is generated by setting $y_p := 0$ and $x_{pj} := 0$ for all $j \in M$. If the k-best solutions of the knapsack problem are available, for a given $k > 0$, then up to k columns can be generated at each call to the oracle.

5.2.2 Computational results

To analyse the performance of different column generation strategies applied to solving problem (5.4), we have selected 261 instances from the literature in one-dimensional CSP (`http://www.math.tu-dresden.de/~capad/`). The initial RMP consists of columns generated by m homogeneous cutting patterns, which corresponds to selecting only one piece per pattern, as many times as possible without violating the width W. In the ACCPM approach, we have used the initial guess $u^0 = 0.5e$. The knapsack problem is solved using a branch-and-bound method described in [28], the implementation of which was provided by the author.

Adding one column to the RMP. In the first set of experiments we consider that only one column is generated by the oracle at each iteration. We have grouped the instances according to m, the number of pieces. Table 1 presents for each class the number of instances (inst), the average number of outer iterations (ite) and the average total CPU time in seconds (time) to solve the instances by each column generation method. The last row (All) presents the average

			CCG		PDCGM		ACCPM		(1)		(2)	
class	**size (m)**	**inst**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**
1	< 100	26	174.0	0.2	127.6	0.6	165.5	0.5	1.36	0.25	1.30	0.78
2	$[100, 150)$	42	408.4	0.9	270.5	2.2	330.5	2.7	1.51	0.38	1.22	1.20
3	$[150, 200)$	110	728.2	5.7	462.7	10.6	589.6	17.3	1.57	0.53	1.27	1.63
4	$[200, 300)$	74	799.8	11.25	509.11	14.74	651.8	25.8	1.57	0.76	1.28	1.75
5	≥ 300	9	1427.3	1473.9	1154.0	358.1	1260.3	1512.8	1.24	4.12	1.09	4.22
All		**261**	**665.9**	**56.6**	**435.4**	**21.4**	**546.4**	**67.3**	**1.53**	**2.64**	**1.26**	**3.14**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.

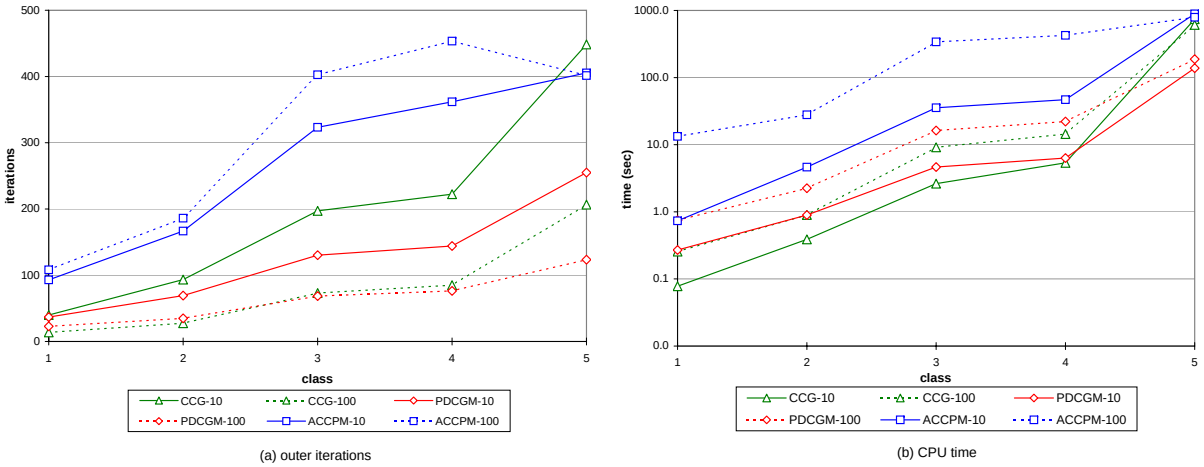
Table 1: CSP - Results adding one column at a time.

results considering the instances in all the classes. Columns CCG, PDCGM and ACCPM show the results for each strategy. Columns (1) and (2) present the relative results of CCG and ACCPM with respect to PDCGM, respectively. Values greater than one mean that PDCGM is more efficient than CCG or ACCPM. For instance, in class 5 (the larger instances), PDCGM is on average 4.12 times faster than CCG and 4.22 times faster than ACCPM. Moreover, the average number of outer iterations is 24% higher than PDCGM for CCG, and 9% higher for ACCPM. Notice that PDCGM requires fewer outer iterations for all classes when compared to both CCG and ACCPM. With respect to CPU time, the performance of PDCGM is enhanced when the instances become larger. Considering all the instances together, PDCGM has the best overall performance on average, being 2.64 and 3.14 times faster than the classical and the analytic centre approaches, respectively.

Adding k-best columns to the RMP. The knapsack solver is able to obtain not only the optimal solution, but also the k-best solutions for a given $k > 0$. Hence, we can generate up to k columns in one call to the oracle to be added to the RMP. It usually improves the performance of a column generation procedure, since more information is gathered at each iteration. With this in mind, we carried out a second experiment in which we tested this strategy for three different values of k: 10, 50 and 100. Figure 1 presents the results in terms of outer iterations and CPU time. The results are shown for each class of instances. For clarity purposes we have chosen to plot the results using two values of k. We have used the name of the method and the choice of k to label each strategy. For instance, CCG-10 denotes the results for the classical column generation technique in which up to 10 columns are generated by the oracle at each outer iteration. Table 3 summarizes the results of CCG and ACCPM in relation to PDCGM.

In Figure 1(a) we observe that when the number of columns added at a time increases, two approaches, CCG and PDCGM, need fewer outer iterations. ACCPM requires more outer iteration to cope with multiple cuts. In classes 3, 4 and 5, PDCGM requires fewer outer iterations than CCG and ACCPM on average, for a fixed k. In Figure 1(b) it can be seen that the CPU time required to solve the instances is inversely proportional to the number of columns added in the majority of the cases. PDCGM has better average CPU times than ACCPM in all classes, and it is the best of the three strategies in the larger instances.

In the relative results given in Table 2, we notice that the differences between PDCGM and ACCPM become larger as more columns are added at each iteration. On the other hand,



(a) outer iterations (b) CPU time

Figure 1: CSP - Results when up to k columns are added at a time.

k	10				50				100			
	(1)		(2)		(1)		(2)		(1)		(2)	
class	ite	time	ite	time	ite	time	ite	time	ite	time	ite	time
1	1.09	0.29	2.53	2.72	0.71	0.32	3.85	6.05	0.60	0.34	4.75	17.80
2	1.35	0.43	2.41	5.18	0.93	0.38	4.19	10.70	0.79	0.40	5.34	12.42
3	1.51	0.57	2.48	7.64	1.19	0.53	4.48	17.86	1.07	0.56	5.88	20.96
4	1.54	0.85	2.51	7.41	1.24	0.69	4.48	16.18	1.11	0.65	5.94	19.26
5	1.76	5.36	1.59	6.43	1.97	3.50	2.55	3.42	1.67	3.23	3.25	4.20
All	**1.51**	**3.25**	**2.42**	**6.87**	**1.22**	**1.81**	**4.30**	**11.21**	**1.08**	**1.45**	**5.63**	**14.83**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.

Table 2: CSP - Relative results when up to k columns are added at a time.

the performances of CCG and PDCGM become closer as the number of generated columns increases, but PDCGM is still 1.45 times faster than CCG for $k = 100$. For each choice of k, PDCGM is on average more efficient than CCG and ACCPM in terms of both outer iterations and CPU time, when the instances in all classes are considered together. A similar result was observed for PDCGM in the first experiment, in which only one column is generated at each call to the oracle.

5.3 Vehicle routing problem with time windows

Consider a set of vehicles $V = \{1, 2, \ldots, v\}$ available to service a set of customers $C = \{1, 2, \ldots, n\}$ with demands d_i, $i \in C$. We assume all the vehicles are identical and are initially at a same depot, and every route must start and finish at this depot. A vehicle can serve more than one customer in a route, as long as its maximum capacity q is not exceeded. Each customer $i \in C$ must be served once within a time window $[a_i, b_i]$. Besides, a service time is assigned for each customer. Late arrivals (after time b_i) are not allowed and if a vehicle arrives earlier to a customer it needs to wait until the window is open (a_i). The objective is to design a set of minimum cost routes in order to serve all the customers.

Let $N = \{0, 1, \ldots, n, n+1\}$ be a set of vertices such that vertices 0 and $n+1$ represent the depot, and the remaining vertices correspond to the customers in C. The time of travelling

from vertex i to vertex j, denoted by t_{ij}, satisfies the triangle inequality and includes the service time at the vertex i. By using this notation, we can formulate the VRPTW as follows:

$$\min \quad \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk} \tag{5.6a}$$

$$\text{s.t.} \quad \sum_{k\in V}\sum_{j\in N} x_{ijk} = 1, \qquad \forall i \in C, \tag{5.6b}$$

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ijk} \leq q, \qquad \forall k \in V, \tag{5.6c}$$

$$\sum_{j\in N} x_{0jk} = 1, \qquad \forall k \in V, \tag{5.6d}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0, \qquad \forall h \in C,\ \forall k \in V, \tag{5.6e}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1, \qquad \forall k \in V, \tag{5.6f}$$

$$s_{ik} + t_{ij} - M(1 - x_{ijk}) \leq s_{jk}, \qquad \forall\ i,j \in N,\ \forall k \in V, \tag{5.6g}$$

$$a_i \leq s_{ik} \leq b_i, \qquad \forall i \in N,\ \forall k \in V, \tag{5.6h}$$

$$x_{ijk} \in \{0,1\}, \qquad \forall\ i,j \in N,\ \forall k \in V. \tag{5.6i}$$

The binary variable x_{ijk} determines whether vehicle $k \in V$ visits vertex $i \in N$ and then goes immediately to vertex $j \in N$. The time vehicle $k \in V$ starts to service the customer $i \in C$ is represented by the variable s_{ik}. We assume that all the parameters are non-negative integers, c_{ij} is given by the Euclidean distance between vertices i and j, and M is a sufficiently large number. Constraints (5.6b) guarantee that each customer must be visited by only one vehicle. Constraints (5.6c) enforce that a vehicle cannot exceed its capacity. Both constraints together ensure that the demand of each client has to be satisfied by only one vehicle. Moreover, constraints (5.6d) and (5.6f) enforce that each vehicle must start and finish its route at the depot, respectively. Constraints (5.6e) guarantee that once a vehicle visits a customer and serves it, it must then move to another customer or end its route at the depot. Constraints (5.6g) establish the relationship between the vehicle departure time from a customer and its immediate successor. Indeed, if $x_{ijk} = 1$ then the constraint becomes $s_{ik}+t_{ij} \leq s_{jk}$. Constraints (5.6h) enforce that the vehicle k serves customer i between time a_i and b_i. Note that if a vehicle is not used, its route is defined as $(0, n+1)$.

5.3.1 Dantzig-Wolfe decomposition

The coefficient matrix of the above formulation has a special structure that can be exploited by DWD, with coupling constraints given by (5.6b). Similar to what was done for the CSP, let $\mathcal{X}$ be the set of all points satisfying constraints (5.6c) to (5.6i). We can then define v independent subsets $\mathcal{X}_k$ from $\mathcal{X}$, for each $k \in V$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_v$. We replace each $\mathcal{X}_k$ by its convex hull $conv(\mathcal{X}_k)$, which can be fully represented by its set of extreme points. For each $k \in V$, P_k represents the set of indices of all extreme points of $conv(\mathcal{X}_k)$. Following

developments in Section 4.1, we obtain the master problem

$$\min \quad \sum_{k\in V}\sum_{i\in N}\sum_{j\in N}\sum_{p\in P_k} c_{ij}x^k_{ijp}\lambda^k_p \tag{5.7a}$$

$$\text{s.t.} \quad \sum_{k\in V}\sum_{j\in N}\sum_{p\in P_k} x^k_{ijp}\lambda^k_p = 1, \qquad \forall i \in C, \tag{5.7b}$$

$$\sum_{p\in P_k} \lambda^k_p = 1, \qquad \forall k \in V, \tag{5.7c}$$

$$\lambda^k_p \geq 0, \qquad \forall k \in V, \forall p \in P_k, \tag{5.7d}$$

where for a given $p \in P_k$, x^k_{ijp} are the components of the corresponding extreme point of $conv(\mathcal{X}_k)$, for all $i, j \in N$. Since the vehicles are identical, the subsets $\mathcal{X}_k$ will be the same for every $k \in V$ and hence the oracle will generate v identical columns. Similar to Section 5.2.1, we can avoid this by aggregating variables and using the following master problem:

$$\min \quad \sum_{i\in N}\sum_{j\in N}\sum_{p\in P} c_{ij}x_{ijp}\lambda_p \tag{5.8a}$$

$$\text{s.t.} \quad \sum_{j\in N}\sum_{p\in P} x_{ijp}\lambda_p = 1, \qquad \forall i \in C, \tag{5.8b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{5.8c}$$

where P is the set of indices of all extreme points of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 = \ldots = \mathcal{X}_v$. The convexity constraint has been dropped since $\mathbf{0} \in conv(\bar{\mathcal{X}})$ and v is a loose upper bound in the constraint $\sum_{p\in P} \lambda_p \leq v$ (see Section 4.1). Let $u = (u_1, \ldots, u_n)$ denote the dual variables associated to constraints (5.8b). Furthermore, let $\overline{u} = (\overline{u}_1, \ldots, \overline{u}_n)$ be an arbitrary dual solution, and assume $\overline{u}_0 = \overline{u}_{n+1} = 0$. The oracle associated with problem (5.8) is given by the subproblem

$$\begin{aligned} \min \quad & \sum_{i\in N}\sum_{j\in N}(c_{ij} - \overline{u}_j)x_{ij} \\ \text{s.t.} \quad & [x_{ij}, s_i]_{i,j\in N} \in conv(\bar{\mathcal{X}}). \end{aligned}$$

This subproblem is an elementary shortest path problem with resource constraints. An optimal solution $[x^\star_{ij}, s^\star_i]_{i,j\in N}$ of this problem is an extreme point of $conv(\bar{\mathcal{X}})$. To generate a column of (5.8), we set $x_{ijp} = x^\star_{ij}$, for all $i, j \in N$.

Although several algorithms are available in the literature (see [24] for a survey), solving the above subproblem to optimality may require a relatively large CPU time, especially when the time windows are wide. As a consequence, a relaxed version is solved in practice, in which non-elementary paths are allowed, *i.e.*, paths that visit the same customer more than one time. Although the lower bound provided by the column generation scheme may be slightly worse in this case, the CPU time to solve the subproblem is considerably reduced. We have adopted this approach in our implementation.

			CCG		PDCGM		ACCPM		(1)		(2)	
class	**size (n)**	**inst**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**
1	25	29	99.9	1.1	48.7	0.7	106.6	0.8	2.05	1.65	2.19	1.26
2	50	29	279.6	28.8	101.3	8.9	162.5	11.2	2.76	3.21	1.60	1.25
3	100	29	797.8	686.6	213.9	184.7	292.2	237.1	3.73	3.72	1.37	1.28
All		**87**	**392.4**	**238.8**	**121.3**	**64.8**	**187.1**	**83.0**	**3.23**	**3.69**	**1.54**	**1.28**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.

Table 3: VRPTW - Results adding one column at a time.

5.3.2 Computational results

To test the three column generation strategies on the VRPTW, we have selected 87 instances from the literature (`http://www2.imm.dtu.dk/~jla/solomon.html`), which were originally proposed in [39]. The initial columns of the RMP have been generated by n-single customer routes which correspond to assigning one vehicle per customer. In the ACCPM approach, we have considered the initial guess $u^0 = 100.0e$. The subproblem is solved by our own implementation of the bounded bidirectional dynamic programming algorithm proposed in [37], with state-space relaxation and identification of unreachable nodes [12]. We have divided the instances in three different classes using n, the number of customers.

Adding one column to the RMP. In Table 3 we compare the performance of the three strategies when only one column is generated by the oracle at each iteration. For each class we present the number of instances (inst), the average number of outer iterations (ite) and the average total CPU time in seconds (time) of each column generation method. The last row (All) shows the average results considering the instances in all classes. Columns CCG, PDCGM and ACCPM show the results for each strategy. Columns (1) and (2) present the relative results of CCG and ACCPM with respect to PDCGM, respectively. The performance of PDCGM is clearly superior in terms of outer iterations and CPU time when it is compared with the other two strategies. On average, the number of outer iterations of CCG and ACCPM are 3.23 and 1.54 times larger than that of PDCGM, respectively. Moreover, when CPU time is considered, PDCGM is 3.69 and 1.28 times faster than CCG and ACCPM, respectively. Note that as we increase the size of the instances, the differences between PDCGM and CCG increase substantially when both, number of iterations and CPU time are considered. On the other hand, the differences in outer iterations with ACCPM become smaller when problems with larger number of customers are solved while there is not a substantial variation with respect to the CPU time.

Adding k-best columns to the RMP. Since the subproblem solver is able to provide the k-best solutions at each call to the oracle, we carried out a second experiment, similar to the one done for the CSP. For each column generation method, we solved each instance using k equal to 10, 50, 100 and 300. In Figure 2 we show the average results of these experiments in terms of outer iterations and CPU time. For clarity purposes, we have only plotted results for k equal to 10 and 300. We have used the same labeling system as for the CSP. For instance,

PDCGM-300 indicates the results when the primal-dual column generation method is used and up to 300 columns are added to the RMP at each outer iteration.

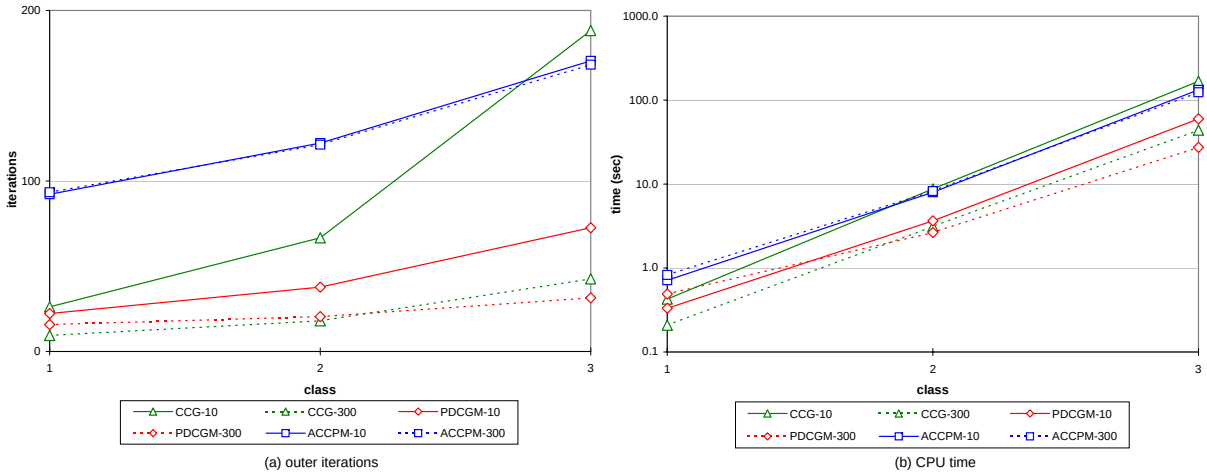


Figure 2: VRPTW - Results when up to k columns are added at a time.

In Figure 2(a) we observe that ACCPM performs similarly no matter the choice of k. On the other hand and similar to the CSP case, PDCGM and CCG require fewer outer iterations when the number of columns added to the RMP increases. For the class with largest instances (*i.e.*, class 3), PDCGM requires fewer outer iterations than ACCPM and CCG, for a fixed k. In Figure 2(b) the behaviour for the three approaches with respect to CPU time is presented. For the VRPTW instances, the CPU time required to solve them is inversely proportional to the number of columns added for all classes. Again, PDCGM has better average CPU times than ACCPM in all classes, and it is the best of the three strategies when the larger instances are considered.

Table 4 shows the relative results when the k-best columns are added at each iteration. In relative terms and by increasing the number of columns added to the RMP, CCG becomes faster and requires less outer iterations than PDCGM when solving class 1 (*i.e.*, set of smaller instances). Nevertheless, PDCGM is still a better strategy for solving larger instances independent of the number of columns added at each iteration. On average, when all instances are considered and for every choice of k, PDCGM performs better than ACCPM and CCG in both, number of outer iterations and CPU time. For instance, considering $k = 300$ and all the instances, PDCGM is 1.55 and 4.36 times faster than CCG and ACCPM, respectively. In terms of outer iterations, there are not significant differences between PDCGM and CCG for this choice of k, while ACCPM requires 5.66 times the number of iterations of PDCGM.

5.4 Capacitated Lot-Sizing Problem with Setup Times

Consider a set of time periods $N = \{1, \ldots, n\}$ and a set of items $M = \{1, \ldots, m\}$ that are processed by a single machine. The objective is to minimize the total cost of producing, holding and setting up the machine in order to satisfy the demands d_{jt} of item $j \in M$ at each time period $t \in N$. The production, holding and setup costs of item j in period t are denoted by c_{jt}, h_{jt} and f_{jt}, respectively. The processing and setup times required to manufacture item j in

k	10				50				100				300			
	(1)		(2)		(1)		(2)		(1)		(2)		(1)		(2)	
class	ite	time	ite	time	ite	time	ite	time	ite	time	ite	time	ite	time	ite	time
1	1.17	1.29	4.14	2.15	0.80	0.90	5.10	2.34	0.73	0.77	5.52	2.37	0.59	0.43	5.91	1.69
2	1.77	2.41	3.24	2.19	1.25	1.91	4.55	3.02	1.12	1.75	5.16	3.29	0.88	1.18	5.93	3.12
3	2.59	2.80	2.35	2.21	1.81	2.09	3.39	3.22	1.73	2.00	4.37	3.98	1.36	1.61	5.35	4.53
All	**2.12**	**2.77**	**2.90**	**2.21**	**1.46**	**2.07**	**4.05**	**3.20**	**1.33**	**1.97**	**4.85**	**3.91**	**1.03**	**1.55**	**5.66**	**4.36**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.

Table 4: VRPTW - Relative results for different choices of k.

time period t are represented by a_{jt} and b_{jt}, respectively. The capacity of the machine in time period t is denoted by C_t. This problem is known as the capacitated lot sizing problem with setup times (CLSPST). We consider the following formulation proposed in [40]

$$\min \quad \sum_{t\in N}\sum_{j\in M}(c_{jt}x_{jt} + h_{jt}s_{jt} + f_{jt}y_{jt}) \tag{5.9a}$$

$$\text{s.t.} \quad \sum_{j\in M}(a_{jt}x_{jt} + b_{jt}y_{jt}) \leq C_t, \qquad \forall t \in N \tag{5.9b}$$

$$s_{j(t-1)} + x_{jt} = d_{jt} + s_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{5.9c}$$

$$x_{jt} \leq Dy_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{5.9d}$$

$$x_{jt} \geq 0, \qquad \forall j \in M,\ \forall t \in N, \tag{5.9e}$$

$$s_{jt} \geq 0, \qquad \forall j \in M,\ \forall t \in N, \tag{5.9f}$$

$$y_{jt} \in \{0,1\}, \qquad \forall j \in M,\ \forall t \in N, \tag{5.9g}$$

where x_{jt} represents the production level of item j in time period t and s_{jt} is the number of units in stock of item j at the end of time period t. Also, the final inventory for every product j is set to zero ($i.e., s_{jn} = 0$). The binary variable y_{jt} determines whether item j is produced in time period t ($y_{jt} = 1$) or not ($y_{jt} = 0$). Constraints (5.9b) enforce that the elapsed time in period t for a given plan of production should not exceed the capacity of the machine in that period. Constraints (5.9c) are the inventory equations which ensure that the production and units of each item in stock at the beginning of a given period must satisfy the demand while the remaining units are stored for next time period. Constraints (5.9d) guarantee that if item j is produced in period t, then the machine must be set up, where D is a sufficiently large number. Constraints (5.9e) and (5.9f) ensure that the level of production and stock at each period t for each item j are non-negative.

5.4.1 Dantzig-Wolfe decomposition

As with the CSP and VRPTW in previous sections, the coefficient matrix of the above formulation has a special structure. We have chosen (5.9b) as the coupling constraint, and the set $\mathcal{X}$ is given by all the points satisfying constraints (5.9c) to (5.9g). For each $j \in M$, we define a subset $\mathcal{X}_j$ by fixing j in $\mathcal{X}$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$. Following Section 4.1, we replace each $\mathcal{X}_j$ by its convex hull, which is fully represented by its extreme points. The resulting master

problem is

$$\min \quad \sum_{j\in M}\sum_{t\in N}\sum_{p\in P_j}\left(c_{jt}x^j_{pt}+h_{jt}s^j_{pt}+f_{jt}y^j_{pt}\right)\lambda^j_p \tag{5.10a}$$

$$\text{s.t.} \quad \sum_{j\in M}\sum_{p\in P_j}\left(a_{jt}x^j_{pt}+b_{jt}y^j_{pt}\right)\lambda^j_p \le C_t, \qquad \forall t\in N, \tag{5.10b}$$

$$\sum_{p\in P_j}\lambda^j_p = 1, \qquad \forall j\in M, \tag{5.10c}$$

$$\lambda^j_p \ge 0, \qquad \forall j\in M, \forall p\in P_j, \tag{5.10d}$$

where P_j is the set of indexes of all extreme points of $conv(\mathcal{X}_j)$, for each $j \in M$. To obtain the oracle associated to this master problem, let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_m)$ denote the dual variables associated to constraints (5.10b) and (5.10c), respectively. Note that u is restricted to be non-positive. Let $\overline{u} = (\overline{u}_1, \ldots, \overline{u}_n)$ and $\overline{v} = (\overline{v}_1, \ldots, \overline{v}_m)$ be an arbitrary dual solution. The oracle is then defined by m subproblems of the form

$$\min \quad \sum_{t\in N}[(c_{jt}-a_{jt}\overline{u}_t)x_{jt}+h_{jt}s_{jt}+(f_{jt}-b_{jt}\overline{u}_t)y_{jt}] - \overline{v}_j \tag{5.11a}$$

$$\text{s.t.} \quad [x_{jt}, s_{jt}, y_{jt}]_{t\in N} \in conv(\mathcal{X}_j), \tag{5.11b}$$

for each $j \in M$. Each subproblem is a single-item lot sizing problem with modified production and set up costs, and without capacity constraint. Hence, it can be solved by the Wagner-Whitin algorithm [43]. For a given $j \in M$, if the optimal value of the subproblem is negative, the corresponding optimal solution $[x^\star_{jt}, s^\star_{jt}, y^\star_{jt}]_{t\in N}$ is used to generate a column of (5.10) by setting $x^j_{pt} = x^\star_{jt}$, $s^j_{pt} = s^\star_{jt}$ and $y^j_{pt} = y^\star_{jt}$. Otherwise, the solution is discarded and no column is generated from that subproblem. Since m subproblems are solved in each call to the oracle, we add up to m columns to the RMP at each outer iteration.

5.4.2 Computational results

We have selected 751 instances proposed in [40] to test the aforementioned column generation strategies. The CCG and PDCGM approaches are initialized using a single-column Big-M technique. The coefficients of this column are set to 0 in the capacity constraints and set to 1 in the convexity constraints. In the ACCPM approach, we have chosen $u^0 = 10.0e$ as the initial guess. The subproblems are solved using our own implementation of the Wagner-Whitin algorithm [43].

Using Trigeiro et al.'s instances we found that all of them were solved, by each of the strategies, in less than 100 seconds showing similar behaviours. The majority of test examples were solved in less than 0.1 seconds. From these results, no meaningful comparisons and conclusions can be derived, and therefore, we have modified the instances in order to challenge the column generation approaches. For each instance and for each product j we have replicated their demands 5 times and divided the capacity, processing time, set up time and costs by the same factor. Also, we have increased the capacity by 10%. Note that we have increased the size of the problems in time periods but not in items and all instances remain feasible. In Table 5, we

		CCG		PDCGM		ACCPM (⋆)		(1)		(2)	
class	**inst**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**
E	58	38.1	1.1	29.7	1.2	38.3	1.1	1.28	0.85	1.29	0.93
F	70	33.4	0.9	27.9	1.1	40.4	1.2	1.19	0.81	1.44	1.07
G	71	44.9	9.4	32.4	6.7	43.2	7.9	1.39	1.41	1.33	1.18
W	12	66.4	1.7	55.3	2.5	48.6	1.5	1.20	0.70	0.88	0.58
X1	180	47.5	6.0	28.8	4.3	35.2	4.5	1.65	1.39	1.22	1.05
X2	180	42.6	10.6	20.5	5.6	27.3	7.0	2.07	1.89	1.33	1.24
X3	180	48.9	18.2	18.7	7.5	24.2	9.4	2.61	2.45	1.30	1.26
All	**751**	**44.66**	**9.44**	**25.13**	**5.04**	**32.37**	**5.98**	**1.78**	**1.87**	**1.29**	**1.19**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.
(⋆) A subset of 7 instances could not be solved by ACCPM using the default accuracy level, $\delta = 10^{-6}$ (4 from class X2 and 3 from class X3). To overcome this we have used $\delta = 10^{-5}$.

Table 5: CLSPST - Results of modified instances.

show a summary of our findings.

We have divided the instances in 7 classes. For each class we present the number of instances (inst), the average number of outer iterations (ite) and the average CPU time in seconds (time) required to solve all the instances in the class. The last row (All) shows the average results considering the 751 instances. As for the CSP and VRPTW, columns CCG, PDCGM and ACCPM present the results for each strategy. Columns (1) and (2) show the relative results of CCG and ACCPM with respect to PDCGM, respectively. Considering the modified instances and for all classes, PDCGM shows on average a reduction in number of outer iterations when compared with ACCPM and CCG. With respect to CPU time, on average PDCGM is faster than ACCPM and CCG. This does not hold for all classes since for some of them CCG and ACCPM are more efficient than PDCGM. However, the overall performance using PDCGM is better since it is always faster than CCG and ACCPM when larger instances are solved (*i.e.*, classes G, X1, X2 and X3).

In addition to the previous analysis, we have considered a set of more challenging instances. We have taken 3 instances from [40], which were used in [9] as a comparison set, to test the three column generation strategies. Additionally, we have selected 8 additional instances from the sets of larger classes, X2 and X3. This small set of 11 instances[1] has been replicated 5, 10, 15 and 20 times following the same procedure described above. The summary of our findings are presented in Table 6, where column r denotes the factor used to replicate the selected instances.

We observe that the number of outer iterations increases as the factor r increases. For every choice of r, PDCGM requires fewer outer iterations when compared to ACCPM and CCG. In addition, the CPU time increases as the size of the instances (and difficulty) increases. Again, PDCGM takes advantage of the reduction of outer iterations for large instances to achieve better times when compared to CCG and ACCPM for every choice of r.

Considering the 44 instances (11 instances and 4 values for r) PDCGM is, on average, 2.83 and 2.61 times faster than CCG and ACCPM, respectively. Additionally, CCG requires 96% more outer iterations than PDCGM while ACCPM needs on average 79% more.

[1]Instances: G30, G53, G57, X21117A, X21117B, X21118A, X21118B, X31117A, X31117B, X31118A, X31118B.

	CCG		PDCGM		ACCPM		(1)		(2)	
r	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**	**ite**	**time**
5	27.5	6.6	11.5	2.2	22.5	4.6	2.39	2.96	1.95	2.06
10	32.0	123.2	15.6	29.9	29.5	70.8	2.05	4.11	1.88	2.36
15	38.4	441.5	20.0	152.9	36.4	393.0	1.92	2.89	1.82	2.57
20	45.5	1395.8	25.9	509.9	42.4	1344.6	1.75	2.74	1.64	2.64
All	**35.8**	**491.8**	**18.3**	**173.7**	**32.7**	**453.3**	**1.96**	**2.83**	**1.79**	**2.61**

(1) CCG/PDCGM; (2) ACCPM/PDCGM.

Table 6: CLSPST - Results using a small set of replicated instances.

6 Conclusions

In this paper we have presented new developments in theory and applications of the primal-dual column generation method (PDCGM). The method is applicable in a wide context when a sequence of restricted master problems (RMPs) have to be solved. PDCGM does it by finding non-optimal and well-centered solutions of RMPs. We have provided computational evidence that the method is powerful when column generation procedure is applied in the context of integer programming. We have developed a theoretical analysis of the method showing that PDCGM converges to an optimum if such exists. We have tested the approach on three well-known classes of problems, namely the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW) and the capacitated lot sizing problem with setup times (CLSPST). The average performance of PDCGM applied to Dantzig-Wolfe reformulations of these problems is significantly better than those of the classical column generation method based on simplex algorithm (CCG) and the analytic centre cutting plane approach (ACCPM).

The success of PDCGM is due to the use of suboptimal solutions that are controlled by a dynamic adjustment of the accuracy required to solve each RMP. Moreover, PDCGM takes advantage of the efficiency of a primal-dual interior point method to solve large scale problems. We have provided computational evidence which demonstrates the effectiveness of PDCGM working in different conditions of column generation, in which single as well as many columns are added to the RMP at each outer iteration, on aggregated and disaggregated schemes. One important feature is that the relative performance of PDCGM improves when larger instances are considered.

Several avenues are available for further studies involving the primal-dual column generation technique. One of them is to combine PDCGM with a branch-and-bound search, in order to generate a branch-and-price framework that is able to solve the original integer programming problems. Furthermore, since PDCGM relies on an interior point method, the investigation of new effective warmstarting strategies applicable in this context is essential for the success of the framework.

Acknowledgements

The authors are grateful to Dr. Franklina M. B. Toledo for preparing the instances of the CLSPST, and to Aline A. S. Leão for providing us with the knapsack solver used in our tests.

Also, the authors would like to thank Dr. Raf Jans for facilitating his results of the CLSPST instances for validation purposes.

References

[1] D. S. Atkinson and P. M. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69:1–43, 1995.

[2] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[3] H. Ben Amor and J. Valério de Carvalho. Cutting stock problems. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 131–161. Springer US, 2005.

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

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

[6] COIN-OR. *OBOE: the Oracle Based Optimization Engine*, 2010. Available at http://projects.coin-or.org/OBOE.

[7] M. Colombo and J. Gondzio. Further development of multiple centrality correctors for interior point methods. *Comput. Optim. Appl.*, 41(3):277–305, 2008.

[8] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[9] Z. Degraeve and R. Jans. A New Dantzig-Wolfe Reformulation and Branch-and-Price Algorithm for the Capacitated Lot-Sizing Problem with Setup Times. *Operations Research*, 55(5):909–920, 2007.

[10] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[11] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.

[12] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle-routing problems. *Networks*, 44:216–229, 2004.

[13] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Studies*, pages 82–114, 1974.

[14] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[15] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem - part II. *Operations Research*, 11(6):863–888, 1963.

[16] J. L. Goffin, A. Haurie, and J. P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38(2):284–302, 1992.

[17] J.-L. Goffin and J.-P. Vial. Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17:805–868, 2002.

[18] J. Gondzio. HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225, 1995.

[19] J. Gondzio. Warm start of the primal-dual method applied in the cutting-plane scheme. *Mathematical Programming*, 83:125–143, 1998.

[20] J. Gondzio and A. Grothey. Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization*, 13(3):842–864, 2003.

[21] J. Gondzio and A. Grothey. A new unblocking technique to warmstart interior point methods based on sensitivity analysis. *SIAM Journal on Optimization*, 19(3):1184–1210, 2008.

[22] J. Gondzio and R. Sarkissian. Column generation with a primal-dual method. Technical Report 96.6, Logilab, 1996.

[23] IBM ILOG CPLEX v.12.1. *Using the CPLEX Callable Library*, 2010.

[24] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 33–65. Springer US, 2005.

[25] B. Kallehauge, J. Larsen, O. B. Madsen, and M. M. Solomon. Vehicle routing problem with time windows. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 67–98. Springer US, 2005.

[26] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

[27] L. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[28] A. A. S. Leão. Geração de colunas para problemas de corte em duas fases. Master's thesis, ICMC - University of Sao Paulo, Brazil, 2009.

[29] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[30] A. S. Manne. Programming of economic lot sizes. *Management Science*, 4(2):115–135, 1958.

[31] R. E. Marsten, W. W. Hogan, and J. W. Blankenship. The boxstep method for large-scale optimization. *Operations Research*, 23(3):389–405, 1975.

[32] R. K. Martinson and J. Tind. An interior point method in Dantzig-Wolfe decomposition. *Computers and Operation Research*, 26:1195–1216, 1999.

[33] J. Mitchell and B. Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, 62:253–276, 1996.

[34] J. E. Mitchell. Computational experience with an interior point cutting plane algorithm. *SIAM Journal of Optimization*, 10(4):1212–1227, 2000.

[35] J. E. Mitchell. Polynomial interior point cutting plane methods. *Optimization Methods and Software*, 18(5):507–534, 2003.

[36] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[37] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170, 2008.

[38] L.-M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[39] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):pp. 254–265, 1987.

[40] W. W. Trigeiro, L. J. Thomas, and J. O. McClain. Capacitated lot sizing with setup times. *Management Science*, 35(3):353–366, 1989.

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

[42] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.

[43] H. M. Wagner and T. M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958.

[44] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.