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,
March 19, 2012.

Abstract

The optimal solutions of the restricted master problems typically leads to an unstable behaviour of the standard column generation technique and, consequently, originates an unnecessarily large number of iterations of the method. To overcome this drawback, variations of the standard approach use interior points of the dual feasible set instead of optimal solutions. In this paper, we focus on a variation known as the primal-dual column generation technique which uses a primal-dual interior point method to obtain well-centred non-optimal solutions of the restricted master problems. We show that the method converges to an optimal solution of the master problem even though non-optimal solutions are used in the course of the procedure. Also, computational experiments are presented using linear-relaxed reformulations of three classical integer programming problems: the cutting stock problem, the vehicle routing problem with time windows, and the capacitated lot sizing problem with setup times. The numerical results indicate that the appropriate use of a primal-dual interior point method within the column generation technique contributes to a reduction of the number of iterations as well as the running times, on average. Furthermore, the results show that the larger the instance, the better the relative performance of the primal-dual column generation technique.

Keywords: column generation; interior point methods; linear programming

1 Introduction

The column generation technique has become a very important tool in the solution of optimization problems [32, 44]. This technique is an iterative procedure applied to solve a linear programming problem with a huge number of variables, called the *master problem* (MP), such

[*]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 and 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.

that the columns in the coefficient matrix of this problem can be generated by following a known rule. By exploiting this characteristic, the column generation technique starts with a reduced version of the problem, called the *restricted master problem* (RMP), in which only a few columns of the MP are considered at first. Iteratively, new columns are generated and added to the RMP until an optimal solution of the MP is obtained. In general, it is achieved by generating a relatively small subset of the columns.

In the standard column generation technique, an unstable behaviour is caused by the use of optimal dual solutions of the RMPs [32, 43]. To overcome this drawback, variations of the standard technique relying on interior points of the dual feasible set of the RMP have been proposed (see Section 2.1). Here we focus on the primal-dual column generation technique [25], in which an interior point method is used to obtain non-optimal solutions that are well-centred in the dual feasible set of the corresponding RMP. Promising computational results have been reported using this technique [25, 34], but it has never been tested on applications in which the MP formulation comes from an integer programming context. Furthermore, no theoretical analysis that guarantees the convergence of the primal-dual approach has been presented. The aim of this paper is to close this gap. First, we review the primal-dual column generation technique and show that it converges to an optimal solution of the master problem even though non-optimal solutions are used in the course of the algorithm. Then, we present extensive computational results for linear-relaxed formulations obtained from the decomposition of three classes of problems which are well-known in the column generation literature: 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 very degenerate restricted master problems, a property that usually causes instability in the standard column generation [5, 30, 11]. We compare the performance of the primal-dual column generation technique against the standard column generation and the analytic centre cutting planning method.

The contributions of this paper are twofold: (i) presenting new theoretical as well as computational results of a naturally stable column generation technique; (ii) showing how primal-dual interior point methods can be efficiently combined with the column generation technique for solving relaxations of integer programming problems. The motivation for the latter lies in the fact that this variation of interior point methods has become very powerful when solving linear programming problems, but only a few attempts have been made to use it within a column generation procedure.

Notice that here we are not concerned with obtaining optimal integer solutions, which would require the development of a branch-and-price framework for each application. Instead, we want to analyse the behaviour of the primal-dual column generation strategy, when applied to a given node of the branch-and-price tree. By improving the performance of the column generation procedure, we are likely to improve the overall performance on solving the integer problem to optimality. The use of the primal-dual column generation technique within a branch-and-price framework will be addressed in a companion paper.

The structure of the remaining sections is the following. In Section 2, we present the main concepts in column generation and establish the notation used throughout the paper. The primal-dual column generation technique and new theoretical developments are discussed in Section 3. In Section 4, a computational study comparing the primal-dual approach to other two column generation techniques is presented. The conclusions and directions for further studies are presented in Section 5.

2 Column generation technique

Consider a master problem (MP) represented as the following linear programming problem

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

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

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

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 to obtain an optimal solution of the 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 also 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 $j \in N$. Hence, we use the minimum among them, obtained by solving the subproblem

$$z_{SP}(\overline{u}) := \min\{0; c_j - \overline{u}^T a_j | a_j \in \mathcal{A}\}. \tag{2.3}$$

In some applications, the subproblem (2.3) can be partitioned into several independent subproblems that provide different types of columns. In this case, $z_{SP}(\overline{u})$ corresponds to the minimum reduced costs among all the subproblems.

The value $z_{SP}(\overline{u})$ is called the *value of the oracle.* If $z_{SP}(\overline{u}) = 0$, we can ensure that there is no negative reduced cost and, hence, an optimal solution of the MP has been obtained. Otherwise, a 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. 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 $z_{SP}(\overline{u})$ and, hence, we have

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

The value of κ is promptly available when the Dantzig-Wolfe decomposition is applied to obtain the column generation scheme.

The column generation terminates when both bounds in (2.5) are the same, *i.e.*, $z_{SP}(\overline{u}) = 0$. We refer to each call to the oracle as an *outer* iteration and consider the column generation scheme to be efficient if it keeps the number of outer iterations small. Every RMP is then solved by an appropriate linear programming technique (infeasible primal-dual interior point method in our case) and the iterations in this process are called *inner* iterations.

2.1 Column generation strategies

As already mentioned in the previous section, the standard column generation is adversely affected by the use of optimal dual solutions. However, solving every RMP to optimality is not needed in a column generation procedure and, hence, variations of the standard technique avoid this strategy. In general, they rely on interior points of the dual feasible set of the RMP. For instance, the stabilization techniques [33, 13, 7] choose a dual point called stability centre and add penalization terms to the dual objective function of the RMP to keep the dual solutions close to this centre. The modified RMP is solved to optimality, however now the dual solutions do not oscillate greatly from one outer iteration to another, because of the penalties added to the problem. For performance comparisons involving stabilized approaches and the standard column generation, see [7, 39, 6].

An interior point column generation based on the simplex method is proposed in [39]. At each outer iteration, a dual solution in the interior of the dual space is obtained by solving the dual problem of the RMP several times, with different objective functions that are randomly generated. 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 considerably reduced in relation to the standard as well as stabilized column generation. However, for applications with large-scale RMPs, the need of solving these problems several times for different objective functions adversely affects the efficiency of the approach.

Other column generation approaches obtain interior points of the dual feasible set without (directly) modifying the RMP. In [35] and [36], 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. For 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 possibly time-consuming problem (NP-hard in many cases).

The analytic centre cutting plane method (ACCPM) [18, 1, 19] is an interior point approach that relies on central prices. 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 prevents the unstable behaviour between consecutive dual points 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 [37]). The use of stabilization terms within the analytic centre cutting plane method has been successful, as showed in [2]. A thorough comparison of our approach against the implementation of ACCPM [2] will be provided in Section 4.

Another interior point approach is the primal-dual column generation technique. Proposed in [25], it relies on an infeasible primal-dual interior point method to find a non-optimal solution of the RMP, such that the distance to optimality is defined in function of the relative gap in the column generation procedure. In the first outer iterations, each RMP is solved with a loose tolerance, and this tolerance is dynamically reduced throughout the iterations as the relative 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 by [34] 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 the column generation technique is used in the solution of linear relaxations that typically arise from Dantzig-Wolfe reformulations.

Notice that the standard column generation and the analytic centre approaches are extremal strategies, as they are based on optimal solutions. In fact, the analytic centre of a feasible set corresponds to the optimal solution of a modified dual problem associated to the RMP. From this point of view, the idea of the primal-dual column generation technique is 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 central trajectory in the dual feasible set. The contribution of using non-optimal solutions is twofold. First, a smaller number of inner iterations is usually needed to solve each RMP and, hence, the running times per outer iteration is reduced. Second, a more stable column generation strategy is likely to be obtained, so that smaller number of outer iterations as well as less total CPU time are usually required.

3 Primal-dual column generation

Proposed in [25], the primal-dual column generation method (PDCGM) is based on non-optimal solutions of the RMPs. A primal-dual interior point method is used 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 theoretical development of the method is presented in this section.

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 standard 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 $0 < (c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|)$, 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 [46, 22].

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 $z_{SP}(\tilde{u}) = 0$, if no columns could be generated from the proposed query point, or a value $z_{SP}(\tilde{u}) < 0$, together with one or more columns to be added to the RMP. Consider the value $\kappa > 0$ defined as (2.4). As already mentioned before, a suitable value for κ is usually promptly available in a column generation scheme. According to Proposition 3.2, a lower bound of the optimal value of the MP can still be obtained.

Proposition 3.2 *Let $\tilde{z}_{SP} := z_{SP}(\tilde{u})$ 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) 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

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

where $\tilde{z}_{SP} := z_{SP}(\tilde{u})$, as defined in Proposition 3.2. At the end of every outer iteration, we recompute the relative gap, and the tolerance ε is updated as

$$\varepsilon^k := \min\{\varepsilon_{\max},\ gap^{k-1}/D\}, \tag{3.2}$$

where $D > 1$ is the *degree of optimality* that relates the tolerance ε^k to the relative gap at iteration $k-1$. 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 standard 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 $b^T\tilde{u}$. Proposition 3.3 shows that the gap is still reduced in this case, and the progress of the algorithm is guaranteed.

Proposition 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 Proposition 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 standard 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 $\kappa$, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
2. **set** LB $= -\infty$, UB $= \infty$, $gap = \infty$, $\varepsilon = 0.5$;
3. **while** $(gap \geq \delta)$ **do**
4.     find a well-centred $\varepsilon$-optimal solution $(\tilde{\lambda}, \tilde{u})$ of the RMP;
5.     UB $= \min($UB$, \tilde{z}_{RMP})$;
6.     call the oracle with the query point $\tilde{u}$;
7.     LB $= \max($LB$, \kappa\tilde{z}_{SP} + b^T\tilde{u})$;
8.     $gap = ($UB $-$ LB$)/(1 + |$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. Even though the optimality tolerance ε decreases geometrically in the algorithm, there is a special case in which the subproblem value is zero, which would cause the method to stall. Fortunately, by using Proposition 3.3 we can guarantee the method still converges successfully. The proof of convergence is given in Proposition 3.4.

Proposition 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 Proposition 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 Proposition 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. □

Having presented a proof of convergence for 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 [20]. 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 [21, 9].

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 [21, 23, 24]). 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 are always 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 Computational experiments

In this section, we present the results of computational experiments using three classes of problems which are well known in the column generation literature. 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). For each application, we have implemented three different column generation strategies. The descriptions of each strategy are the following:

- Standard column generation (SCG): each RMP is solved to optimality by the simplex method available in the commercial solver CPLEX [27]. The solver is used as a black-box by invoking the function `lpopt()`, and all parameters are left at their default values. Note that by using this function we are not forcing CPLEX to use any particular method (primal, dual or barrier) and therefore, we rely on the internal rules of CPLEX on this regard. Nevertheless, we have made preliminary experiments in which we forced each of the `lpopt()` methods in the SCG. The primal and the dual simplex methods resulted in a comparable behaviour in terms of CPU time, being the primal slightly better than the dual method as the optimal basis remains primal feasible from one outer iteration to another. The overall performance using the barrier method was inferior than the other two methods, which shows that an appropriate use of an interior point method is essential for its success in the column generation context.

- Primal-dual column generation (PDCGM): the suboptimal solutions of each RMP are obtained by using the interior point solver HOPDM [20], 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 [8], a state-of-the-art implementation of the analytic centre strategy with additional stabilization terms [2].

For each application and for every aforementioned column generation strategy, the subproblems are solved using the same source-code. Also, the SCG and the PDCGM are initialized with the same columns and, hence, have the same initial RMP. The ACCPM requires an initial dual point to start from, instead of a set of initial columns. After preliminary tests, we have chosen initial dual points that led to a better performance of the method on average. We have used different initial dual points for each application, as will be specified later. To run the tests we have used a computer with processor Intel Core 2 Duo 2.26 Ghz, 4 GB RAM, and Linux operating system. For each of the strategies, we stop the column generation procedure when the relative gap becomes smaller than the default accuracy $\delta = 10^{-6}$.

The purpose of comparing the PDCGM against the SCG is to give an idea of how much it can be gained in overall performance in relation to the standard approach, *i.e.*, without any stabilization. Undoubtedly, it would be interesting to consider stabilized versions of the standard column generation in the computational experiments presented here. However, the lack of publicly available codes of stabilized versions discouraged us to include them at this stage of our study. For the interested reader, available comparisons between standard and stabilized column generation are available in the literature for the same applications [39, 7, 6]. The ACCPM was included in our experiments for being a strategy that also relies on an interior point method (although essentially different). After extensive testing we chose what seems to be the best possible starting point/parameters setting for OBOE/COIN.

In the applications addressed in this paper, the column generation schemes are obtained by applying Dantzig-Wolfe decomposition (DWD) to the corresponding integer programming formulations [10, 42]. In each application, the decomposition leads to an integer MP and also an integer (pricing) subproblem. Here, we relax the integrality of the variables in the integer MP and then solve it by column generation, which gives a lower bound of the optimal value of the original formulation. To obtain an integer solution, it would be necessary to combine the column generation with a branch-and-bound search, which is called a branch-and-price method [4, 32]. However, this combination is out of the scope of this paper, as we are concerned with the behaviour of the column generation strategies.

4.1 Cutting stock problem

The one-dimensional CSP consists in determining the smallest number of rolls of fixed width W that have to be cut in order to satisfy the demands of m pieces which may have different widths [17]. The coefficient matrix of its standard integer programming formulation has a special structure which is well-suited to the application of the DWD [5]. The oracle associated to the decomposition is given by a set of n subproblems, where n is an upper bound for the total number of rolls. Since the stock pieces have all the same width, these subproblems are identical and, hence, an aggregated master problem can be used instead. The oracle is then given by an integer knapsack problem. If the k-best solutions of the knapsack problem are available, for a given $k > 0$, then up to k (different) columns can be generated at each call to the oracle (see [5, 7] for further details about the decomposition). The number of rows in any restricted master problem is equal to m.

To analyse the performance of the different column generation strategies addressed here, we have initially selected 262 instances from the literature in one-dimensional CSP (`http://www.math.tu-dresden.de/~capad/`). Additionally, we have extended our comparisons by using 180 instances from the so-called triplet and uniform sets proposed in [14]. In all these cases, 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 and after testing with different values, we have used the initial guess $u^0 = 0.5e$ which has provided the best results for this strategy. The knapsack problem is solved using a branch-and-bound method described in [31], the implementation of which was provided by the author.

	SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
		Time (in s)			Time (in s)			Time (in s)				
Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
S	571.8	2.0	2.6	368.9	1.9	4.8	466.2	3.0	10.5	$< 0.01(< 0.01)$	$0.01(< 0.01)$	0.02(0.01)
L	881.3	153.7	155.8	591.4	35.5	44.5	734.0	143.9	182.4	0.18(0.17)	0.08(0.06)	0.25(0.20)
All	671.0	50.6	51.7	440.3	12.7	17.5	552.1	48.2	65.6	0.08(0.07)	0.04(0.03)	0.12(0.09)

Table 1: CSP - Average results on the 262 instances adding one column at a time.

Adding one column to the RMP In the first set of numerical experiments we consider that only one column is generated by the oracle at each iteration. We have classified the instances into two sets according to m, the number of pieces. In class S (set of small instances) we have included 178 instances with dimensions ranging between 15 and 199 pieces while in class L (set of large instances) we have 84 instances ranging from 200 to 555 pieces. Table 1 presents for each class and strategy: the average number of outer iterations (Iter), the average CPU time spent in the oracle (Oracle) and the average CPU time required for the column generation procedure (Total). The last row (All) presents the average results considering the 262 instances. Additionally, the last three columns show the average total and oracle times per iteration (in seconds). The oracle times are shown in parentheses. For the set of small instances, the SCG is the best overall strategy if we consider the total CPU time, while the PDCGM has the smallest number of iterations on average. For the set of large instances, the PDCGM is on average 3.5 times faster than the SCG and 4.1 times faster than the ACCPM. If we consider the average over all the instances, the PDCGM requires fewer outer iterations and less CPU time when compared with both, the SCG and the ACCPM.

Observe that the RMPs solved at each outer iteration are actually small/medium size linear programming problems. The number of columns in the last RMP is approximately the number of initial columns plus the number of outer iterations. Note that for the SCG the time spent in solving the RMPs is very small in relation to the time required to solve the subproblems, regardless the size of the instances. It happens because the simplex method available in the CPLEX solver is very efficient on solving/reoptimizing these linear programming problems. For the PDCGM and the ACCPM, the proportion of the total CPU time required to solve the RMP and the oracle varies according to the size of the instances.

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 carry out a second set of experiments in which we have tested these strategies for three different values of k: 10, 50 and 100.

In Table 2, we present the results obtained by adding more than one column at each iteration. For the set of small instances, the SCG is more efficient than the PDCGM and the ACCPM, regardless the number of columns added at each iteration. However, when the set of large instances is considered, the PDCGM is on average more efficient than the SCG and the ACCPM in terms of both outer iterations and CPU time. For instance, if we consider $k = 100$, the PDCGM is 2.2 times faster than the SCG and 16.7 times faster than the ACCPM. Similar

		SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
k	Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
10	*S*	149.7	0.9	1.2	102.2	0.8	2.1	253.1	2.5	26.1	0.01(0.01)	0.02(0.01)	0.10(0.01)
	L	251.4	75.8	77.0	158.4	15.1	18.3	368.3	82.6	148.7	0.31(0.30)	0.12(0.10)	0.40(0.22)
	All	182.3	24.9	25.5	120.2	5.4	7.3	290.0	28.2	65.4	0.14(0.14)	0.06(0.04)	0.23(0.10)
50	*S*	70.9	1.8	2.1	63.2	2.0	3.8	276.8	10.7	106.3	0.03(0.03)	0.06(0.03)	0.38(0.04)
	L	133.7	56.6	58.2	97.1	18.8	23.1	400.2	45.5	277.6	0.44(0.42)	0.24(0.19)	0.69(0.11)
	All	91.0	19.4	20.1	74.1	7.4	10.0	316.4	21.9	161.2	0.22(0.21)	0.13(0.10)	0.51(0.07)
100	*S*	53.7	3.8	4.2	53.9	4.6	7.3	308.4	31.2	221.8	0.08(0.07)	0.14(0.09)	0.72(0.10)
	L	101.0	66.3	67.8	82.3	25.4	31.5	449.4	96.4	525.2	0.67(0.66)	0.38(0.31)	1.17(0.21)
	All	68.8	23.9	24.6	63.0	11.3	15.1	353.6	52.1	319.1	0.36(0.35)	0.24(0.18)	0.90(0.15)

Table 2: CSP - Average results on the 262 instances adding up to k columns at a time.

	SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
		Time (in s)			Time (in s)			Time (in s)				
Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
$S(u,t)$	31.0	0.9	1.0	34.0	1.7	2.2	143.3	4.8	13.7	0.03(0.03)	0.06(0.05)	0.10(0.03)
$L(u,t)$	150.6	259.4	263.5	110.2	396.0	405.1	383.3	443.5	1064.3	1.75(1.72)	3.68(3.59)	2.78(1.16)
$V(u,t)$	442.7	142.1	190.5	255.3	81.4	145.7	-	-	-	0.43(0.32)	0.57(0.32)	-(-)

Table 3: CSP - Average results on 160 instances from triplet and uniform problem sets adding up to 100 columns at a time.

results are observed when all instances are considered. Again for $k = 100$, the PDCGM is 1.6 times faster than the SCG and 21.1 times faster than the ACCPM on average. The results indicate that the best overall strategy to solve the 262 instances is the PDCGM with $k = 10$, which is on average 2.8 and 8.9 times faster than the best result found with the SCG ($k = 50$) and the ACCPM ($k = 10$), respectively. Clearly, the behaviour of the ACCPM is adversely affected by the number of columns added at a time, as the number of iterations and the CPU time required for solving the RMPs are considerably increased for larger values of k. The main reason for this behaviour is that the localization set may be drastically changed from one outer iteration to another if many columns are added. Hence, finding the new analytic centre can be very expensive in this case.

From Tables 1 and 2 we observe that the average CPU time per iteration spent in the oracle is smaller for the PDCGM and the ACCPM than for the SCG, which shows the importance of using well-centred, stable dual solutions. Furthermore, the PDCGM is able to solve the RMPs more efficiently than the ACCPM and, hence, better total average CPU times per iteration are achieved.

We have also compared the performances of the column generation strategies by using 160 instances from the triplet and uniform problem sets. The results when 100 columns are added at each iteration are presented in Table 3. We have grouped the instances in three classes. The class of small instances, $S(u,t)$, contains 80 instances with m varying from 60 to 120. The $L(u,t)$ class groups 120 instances with m between 249 and 501. Finally, the last row, $V(u,t)$, contains 20 instances with $m = 1000$. Note that for this class of instances, the ACCPM was not able to solve all the instances so we have omitted results for the ACCPM in the last row. The columns in Table 3 have the same meaning as for Tables 1 and 2. For these sets of instances, the SCG performs better than the PDCGM and the ACCPM in classes $S(u,t)$ and $L(u,t)$, however for very large instances, with $m = 1000$, PDCGM outperforms the other two strategies in both, CPU time and the number of outer iterations.

		SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
Instance	m	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
BPP_U09498	1005	548	12760.2	12946.8	293	5545.0	5678.0	762	10054.0	21253.6	23.6(23.3)	19.4(18.9)	27.9(13.2)
BPP_U09513	975	518	9741.1	9903.8	267	4169.3	4276.7	779	7404.4	19362.0	19.1(18.8)	16(15.6)	24.9(9.5)
BPP_U09528	945	541	9011.3	9173.2	276	4810.9	4923.6	740	6586.1	15919.8	17(16.7)	17.8(17.4)	21.5(8.9)
BPP_U09543	915	506	7676.4	7797.8	263	3624.1	3723.7	723	5254.7	13448.9	15.4(15.2)	14.2(13.8)	18.6(7.3)
BPP_U09558	885	482	5479.0	5585.0	265	2631.4	2730.4	683	4222.4	10860.5	11.6(11.4)	10.3(9.9)	15.9(6.2)
BPP_U09573	855	473	4693.7	4771.1	230	1980.2	2054.3	672	3732.1	9793.7	10.1(9.9)	8.9(8.6)	14.6(5.6)
BPP_U09588	825	467	4876.0	4950.3	247	1573.9	1649.4	658	3983.1	9376.4	10.6(10.4)	6.7(6.4)	14.2(6.1)
BPP_U09603	795	465	3893.9	3961.7	237	1597.8	1668.3	627	3055.2	7503.6	8.5(8.4)	7(6.7)	12.0(4.9)
BPP_U09618	765	424	2773.2	2830.4	203	1041.9	1091.6	617	2156.1	6466.7	6.7(6.5)	5.4(5.1)	10.5(3.5)
BPP_U09633	735	432	2832.7	2878.1	217	912.4	969.0	595	1750.7	5307.6	6.7(6.6)	4.5(4.2)	8.9(2.9)
BPP_U09648	705	424	2611.3	2659.5	209	807.9	856.8	582	1403.0	4466.2	6.3(6.2)	4.1(3.9)	7.7(2.4)
BPP_U09663	675	381	2155.8	2187.4	202	613.0	654.0	534	1073.7	3324.9	5.7(5.7)	3.2(3)	6.2(2.0)
BPP_U09678	645	376	1745.3	1774.6	173	387.1	417.5	542	1042.8	3395.1	4.7(4.6)	2.4(2.2)	6.3(1.9)
BPP_U09693	615	384	1323.6	1347.2	165	400.6	426.8	520	875.9	2773.2	3.5(3.4)	2.6(2.4)	5.3(1.7)

Table 4: CSP - Results on 14 large instances adding up to 100 columns at a time.

Finally, we have further compared the performance of the three approaches, using 14 instances with numbers of items (m) varying from 615 to 1005, which leads to larger restricted master problems. Table 4 shows the results of this experiment. In all cases, the PDCGM is faster and requires less iterations than the SCG and the ACCPM, which supports the conclusion that the relative performance of the PDCGM is improved as the instances become larger.

4.2 Vehicle routing problem with time windows

Consider a set of vehicles available in a depot to serve n customers with known demands. A vehicle can serve more than one customer in a route, as long as its maximum capacity is not exceeded. Each customer must be served once within a given time window. Besides, a service time is assigned for each customer. Late arrivals are not allowed and if a vehicle arrives earlier to a customer it must wait until the window is open. We assume all the vehicles are identical and are initially at the same depot, and every route must start and finish at this depot. The objective is to design a set of minimum cost routes in order to serve all the customers. The column generation technique has been successfully used in the solution of the VRPTW, after applying DWD to the standard integer programming formulation [12, 30]. The subproblem obtained in this case corresponds to an elementary shortest path problem with resource constraints. Although exact algorithms are available in the literature (see [28] for a survey), solving this subproblem to optimality may require a relatively large CPU time, especially when the time windows are wide. As a consequence, a relaxed version has been used in practice, in which non-elementary paths are allowed (*i.e.*, paths that visit the same customer more than once). Even though 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 three implementations. As it will be observed in the following results, a large amount of the total CPU time is spent on solving the subproblems. We believe that using an exact subproblem solver will lead to similar results, although more tests in this direction may be required to support this belief. Similarly to CSP, an aggregated master problem is used, as we consider identical vehicles. The restricted master problems have the set covering structure and the number of rows is equal to n.

We have selected 87 VRPTW instances from the literature (`http://www2.imm.dtu.dk/`

	SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
		Time (in s)			Time (in s)			Time (in s)				
Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
S	99.9	0.8	0.8	48.7	0.4	0.5	106.6	0.4	0.6	0.01(0.01)	0.01(0.01)	0.01($<$ 0.01)
M	279.6	20.0	20.1	101.3	6.1	6.3	162.4	7.5	7.8	0.07(0.07)	0.06(0.06)	0.05(0.05)
L	797.8	469.7	470.4	213.7	127.8	128.6	292.2	163.4	164.8	0.59(0.59)	0.60(0.60)	0.56(0.56)
All	392.4	163.5	163.8	121.3	44.8	45.1	187.1	57.1	57.8	0.42(0.42)	0.37(0.37)	0.31(0.31)

Table 5: VRPTW - Average results on 87 instances adding one column at a time.

`~jla/solomon.html`), which were originally proposed in [40]. We have divided them in small ($n = 25$), medium ($n = 50$) and large ($n = 100$) classes. Each class has 29 instances. 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 $u^0 = 100.0e$ which after testing various settings has proven to be the choice which gives the best overall results for this problem. The subproblem is solved by our own implementation of the bounded bidirectional dynamic programming algorithm proposed in [38], with state-space relaxation and identification of unreachable nodes [15].

Adding one column to the RMP In Table 5 we compare the performance of the three strategies when only one column is added to the RMP at each iteration. For each class and strategy we present: the number of outer iterations (Iter), the average CPU time to solve the subproblems (Oracle) and the average total CPU time required for the column generation (Total). The last row (All) shows the average results considering the 87 instances. In the last three columns, the average total and oracle time per iteration is presented. In all the classes, the PDCGM shows the best average performance in the number of iterations and total CPU time compared with the other two strategies. When the size of the instances increases, the difference between the SCG and the other two strategies increases as well, with the SCG being the one which shows the worst overall performance. Considering the 87 instances, the PDCGM is on average 3.7 and 1.3 times faster than the SCG and the ACCPM, respectively.

Notice that, differently from what was observed on the CSP results, the CPU time required for solving the RMPs is very small not only for the SCG, but also for the PDCGM and the ACCPM. In the VRPTW, the RMPs have the set covering structure, which corresponds to a very sparse coefficient matrix with only binary components, a property that is well exploited by the solvers.

Adding k-best columns to the RMP Since the subproblem solver is able to provide the k-best solutions at each iteration, we carried out a second set of experiments. For each column generation method, we have solved each instance using k equal to 10, 50, 100, 200 and 300. In Table 6 we show the results of these experiments where column k denotes the maximum number of columns added at each iteration to the RMP.

For class S, the SCG and the PDCGM have a similar overall performance. Now, if we take into account classes M and L, the PDCGM seems to be consistently more efficient than the other two approaches in both, number of outer iterations and total CPU time, for any k. The same conclusion is obtained considering all the 87 instances. For all the strategies and values of k, the PDCGM with $k = 200$ is the most efficient setting on average, as it is 1.6 and 4.5

		SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
k	Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
10	*S*	26.2	0.3	0.3	22.3	0.2	0.2	93.6	0.4	0.5	0.01(0.01)	0.01(0.01)	0.01(< 0.01)
	M	66.7	6.1	6.2	37.7	2.4	2.6	122.0	5.4	5.7	0.09(0.09)	0.07(0.06)	0.05(0.04)
	L	188.2	113.5	114.1	72.6	41.0	41.6	170.6	90.9	92.1	0.61(0.60)	0.57(0.56)	0.54(0.53)
	All	93.7	40.0	40.2	44.2	14.5	14.8	128.7	32.2	32.8	0.43(0.43)	0.33(0.33)	0.25(0.25)
50	*S*	14.4	0.2	0.2	18.1	0.1	0.2	92.2	0.4	0.5	0.01(0.01)	0.01(0.01)	0.01(< 0.01)
	M	33.1	3.5	3.5	26.4	1.6	1.8	120.0	5.3	5.7	0.11(0.11)	0.07(0.06)	0.05(0.04)
	L	88.0	54.9	55.5	48.6	26.0	27.0	165.2	85.8	87.8	0.63(0.62)	0.56(0.53)	0.53(0.52)
	All	45.2	19.5	19.7	31.0	9.3	9.7	125.8	30.5	31.3	0.44(0.43)	0.31(0.30)	0.25(0.24)
100	*S*	12.2	0.2	0.2	16.7	0.1	0.2	92.3	0.4	0.6	0.02(0.02)	0.01(0.01)	0.01(< 0.01)
	M	26.0	2.9	3.0	23.2	1.4	1.7	119.7	5.4	5.8	0.12(0.11)	0.07(0.06)	0.05(0.05)
	L	65.4	41.7	42.4	37.9	20.3	21.5	166.0	84.5	87.5	0.65(0.64)	0.57(0.54)	0.53(0.51)
	All	34.5	14.9	15.2	25.9	7.3	7.8	126.0	30.1	31.3	0.44(0.43)	0.3(0.28)	0.25(0.24)
200	*S*	9.8	0.1	0.2	16.1	0.1	0.3	92.4	0.4	0.6	0.02(0.01)	0.02(0.01)	0.01(< 0.01)
	M	20.8	2.3	2.4	21.1	1.2	1.7	120.9	5.3	6.0	0.12(0.11)	0.08(0.06)	0.05(0.04)
	L	50.1	33.1	33.9	32.4	16.9	18.7	167.4	82.1	89.2	0.68(0.66)	0.58(0.52)	0.53(0.49)
	All	26.9	11.9	12.1	23.2	6.1	6.9	126.9	29.3	31.9	0.45(0.44)	0.30(0.26)	0.25(0.23)
300	*S*	9.4	0.1	0.1	15.8	0.1	0.3	93.3	0.4	0.6	0.01(0.01)	0.02(0.01)	0.01(< 0.01)
	M	18.0	2.1	2.2	20.4	1.2	1.8	121.1	5.2	6.1	0.12(0.12)	0.09(0.06)	0.05(0.04)
	L	42.6	28.7	29.7	31.5	16.2	18.8	168.7	79.4	89.9	0.70(0.67)	0.60(0.51)	0.53(0.47)
	All	23.3	10.3	10.7	22.6	5.8	7.0	127.7	28.3	32.2	0.46(0.44)	0.31(0.26)	0.25(0.22)

Table 6: VRPTW - Average results on 87 instances adding at most k columns at a time.

times faster than the best results obtained with the SCG ($k = 300$) and the ACCPM ($k = 100$), respectively.

Similarly to the results obtained for the CSP, the well-centred dual points provided by the PDCGM and the ACCPM lead to smaller average oracle CPU times per iteration when compared to the SCG. However, the ACCPM achieved now the best results regarding the total CPU times per iteration, as it can efficiently solve the RMPs in this case, even if up to 300 columns are added at each call to the oracle. The only drawback of this strategy was the (relatively) large number of outer iterations.

Additionally, we have tested the three described column generation strategies in more challenging instances with 200, 400 and 600 customers, which were proposed in [26]. Table 7 shows the results of this second round of experiments, adding 300 columns per iteration. Column n denotes the number of customers per instance while the remaining columns have the same meaning as in Table 5. For all these instances, the PDCGM requires less CPU time and fewer iterations when compared with the SCG and the ACCPM. For the most difficult instance the PDCGM is 2.1 and 6.4 times faster than the SCG and the ACCPM, respectively. In terms of time per iteration, the three strategies behave similarly.

		SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
Instance	n	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
R1_2_1	200	57	36.4	42.7	45	26.0	34.2	423	191.5	201.9	0.7(0.6)	0.8(0.6)	0.5(0.5)
C1_2_1	200	85	32.5	41.0	29	12.6	15.2	169	71.6	81.6	0.5(0.4)	0.5(0.4)	0.5(0.4)
RC1_2_1	200	67	105.3	109.9	57	76.8	88.4	385	566.6	607.1	1.6(1.6)	1.6(1.3)	1.6(1.5)
R1_4_1	400	131	793.3	865.2	84	596.1	640.5	636	2994.5	3075.6	6.6(6.1)	7.6(7.1)	4.8(4.7)
C1_4_1	400	137	453.2	551.9	53	171.4	185.7	272	885.8	908.6	4.0(3.3)	3.5(3.2)	3.3(3.3)
RC1_4_1	400	189	2706.0	2788.8	113	1359.8	1436.1	521	6547.6	6649.4	14.8(14.3)	12.7(12)	12.8(12.6)
R1_6_1	600	222	7226.1	7558.4	118	4142.1	4259.9	897	25599.4	25870.2	34.0(32.6)	36.1(35.1)	28.8(28.5)
C1_6_1	600	183	1920.8	2334.7	48	495.7	510.2	482	5114.7	5172.9	12.8(10.5)	10.6(10.3)	10.7(10.6)
RC1_6_1	600	258	18701.4	18972.3	150	8676.8	8844.3	923	56177.4	56683.3	73.5(72.5)	59.0(57.8)	61.4(60.9)

Table 7: VRPTW - Results on 6 large instances adding 300 columns at a time.

4.3 Capacitated Lot-Sizing Problem with Setup Times

Consider m items which must be processed by a single machine in n time periods. The objective is to minimize the total cost of producing, holding and setting up the machine in order to satisfy the demands of each item at each time period. Processing and setup times are associated to the manufacturing of each item and the machine has a limited capacity. This problem is known as the capacitated lot sizing problem with setup times (CLSPST) [41, 29]. A detailed description of the decomposition used here can be found in [7]. Each subproblem is a single-item lot sizing problem with modified production and setup costs, and without capacity constraints. Hence, it can be solved by the Wagner-Whitin algorithm [45]. Unlike the other two applications, in the CLSPST m different subproblems are solved in each call to the oracle and a disaggregated master problem is used. Up to m columns may be added to the RMP at each outer iteration, one from each subproblem.

We have selected 751 instances proposed in [41] to test the aforementioned column generation strategies in this application. The SCG and the 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, after several settings, we have chosen $u^0 = 10.0e$ as the initial dual point. The subproblems are solved using our own implementation of the Wagner-Whitin algorithm [45].

For each column generation strategy, we found that the 751 instances were solved in less than 100 seconds. The majority of them were solved in less than 0.1 seconds. From these results, no meaningful comparisons and conclusions can be derived, so we have modified the instances in order to challenge the column generation approaches. For each instance and for each product we have replicated their demands 5 times and divided the capacity, processing time, setup 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 therefore, all instances remain feasible. In Table 8, we show a summary of our findings using the modified instances. We have grouped the instances in 7 different classes. Small instances are included in classes E, F and W while classes $G, X1, X2$ and $X3$ contain larger instances. For each class and strategy we present: the number of outer iterations (Iter), the average CPU time to solve the subproblems (Oracle) and the average total CPU time required for the column generation (Total). The last row (All) shows the average results considering the 751 modified instances. Additionally to our usual notation, we have included the number of instances per class (Inst). From Table 8, we can observe that the strategies have different performances for the small instances classes and on average each strategy requires less than 2 seconds to solve an instance from these classes. If we consider the total CPU time, the SCG is slightly better for classes E and F, and the ACCPM outperforms the other two strategies only in class W. If we look at the oracle times, we will observe that for small instances the ACCPM and the PDCGM outperform the SCG due to the reduction in the number of outer iterations. Now, if we observe the performance in classes containing larger instances (*i.e.*, G, $X1$, $X2$ and $X3$), the PDCGM outperforms the other two strategies on average. Furthermore, for the 751 instances (All), the PDCGM reduces the average number of outer iterations and total CPU time when compared with the ACCPM and the SCG.

		SCG			PDCGM			ACCPM[(1)]			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
Class	Inst	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
E	58	38.1	0.7	0.7	29.7	0.5	0.9	38.3	0.7	0.8	0.02(0.02)	0.03(0.02)	0.02(0.02)
F	70	33.4	0.6	0.6	28.0	0.5	0.8	40.4	0.7	0.9	0.02(0.02)	0.03(0.02)	0.02(0.02)
W	12	66.4	1.2	1.2	55.3	1.0	1.8	48.6	0.8	1.1	0.02(0.02)	0.03(0.02)	0.02(0.02)
G	71	44.8	6.6	6.6	32.4	3.9	4.7	43.2	5.2	5.6	0.15(0.15)	0.15(0.12)	0.13(0.12)
$X1$	180	47.5	4.2	4.2	28.8	2.4	3.0	35.2	3.0	3.3	0.09(0.09)	0.10(0.08)	0.09(0.09)
$X2$	180	42.6	7.4	7.5	20.5	3.5	3.9	27.4	4.6	5.0	0.18(0.17)	0.19(0.17)	0.18(0.17)
$X3$	180	48.9	12.7	12.8	18.7	4.7	5.2	24.3	6.1	6.7	0.26(0.26)	0.28(0.25)	0.28(0.25)
All	751	44.7	6.6	6.6	25.1	3.0	3.5	32.4	3.9	4.3	0.15(0.15)	0.14(0.12)	0.13(0.12)

[(1)] A subset of 7 instances could not be solved by the 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 8: CLSPST - Average results on 751 instances adding one column per subproblem at a time.

	SCG			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
		Time (in s)			Time (in s)			Time (in s)				
r	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCG	PDCGM	ACCPM
5	27.5	4.7	4.7	11.5	1.5	1.6	22.5	3.1	3.2	0.17(0.17)	0.14(0.13)	0.14(0.14)
10	32.0	62.7	62.7	15.6	20.4	21.0	29.5	49.1	49.5	1.96(1.96)	1.35(1.31)	1.68(1.66)
15	38.4	308.8	308.8	20.0	103.8	106.2	36.4	273.2	274.3	8.04(8.04)	5.31(5.19)	7.54(7.51)
20	45.5	975.6	975.8	25.9	350.5	358.4	42.4	938.7	941.0	21.45(21.44)	13.84(13.53)	22.19(22.14)
All	35.8	337.9	338.0	18.3	119.0	121.8	32.7	316.0	317.0	9.44(9.44)	6.66(6.50)	9.69(9.66)

Table 9: CLSPST - Average results on 11 modified instances adding one column per subproblem at a time.

In addition to the previous experiment, we have considered a set of more challenging instances. We have taken 3 instances from [41], which were used in [11] as a comparison set, to test the three column generation strategies. Additionally, we have selected 8 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 9, where column r denotes the factor used to replicate the selected instances. From the results, we see that for every choice of r, the PDCGM requires fewer outer iterations and less CPU time on average, when compared with the ACCPM and the SCG. Considering the 44 instances (11 instances and 4 values for r), the PDCGM is 2.8 and 2.6 times faster than the SCG and the ACCPM, respectively.

If we consider the average CPU time per iteration for the CLSPST modified instances, the PDCGM is the most efficient among the studied strategies, while the SCG and the ACCPM have very similar times per iteration. This observation allows us to think that for some applications taking optimality and stabilization strategies as separated objectives may not originate any saving. However, if one can combine both objectives the resulting method can deliver important savings in terms of CPU time and iterations.

5 Conclusions

In this paper we have presented new developments in theory and applications of the primal-dual column generation method (PDCGM). The method relies on a primal-dual interior point method to obtain non-optimal and well-centred solutions of the restricted master problems, leading to

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

a more stable approach in relation to the standard column generation technique. Theoretical support is given to show that the PDCGM converges to an optimum of the master problem, even though non-optimal dual solutions are used. Also, computational experiments show that the method is competitive when compared against the standard column generation method (SCG) and the analytic centre cutting plane method (ACCPM). The experiments were based on linear relaxations of integer master problems obtained from applying the Dantzig-Wolfe decomposition to integer programming formulations of three well-known applications, namely the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW) and the capacitated lot sizing problem with setup times (CLSPST). Different types of master problem formulations are used on these applications: an aggregated master problem in the CSP, an aggregated master problem with a set covering structure in the VRPTW, and a disaggregated master problem in the CLSPST. Additionally, we have tested the addition of different numbers of columns at each outer iteration, which typically affects the behaviour of the methods.

By analysing the computational results, we conclude that the PDCGM achieves the best overall performance when compared to the SCG and the ACCPM. Although the SCG is usually the most efficient for the small instances, we have observed that the relative performance of the PDCGM improves when larger instances are considered. The comparison of the PDCGM against the SCG gives and idea of how much can be gained by using non-optimal and well-centred dual solutions provided by a primal-dual interior point method. One important characteristic of the PDCGM is that no specific tuning was necessary for each application, while the success of using a stabilization technique for the SCG and the ACCPM sometimes strongly depends on the appropriate choice of parameters for a specific application. The natural stabilization available in the PDCGM due to the use of well-centred interior point solutions is a very attractive feature of this column generation approach.

Several avenues are available for further studies involving the primal-dual column generation technique. One of them is to compare the performance of the PDCGM with advanced column generation methods such as generalized bundle methods and the volume algorithm [16, 3]. Another, is to combine this technique with a branch-and-bound algorithm to obtain a branch-and-price framework that is able to solve the original integer programming problems. Furthermore, since the 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 making the instances of the CLSPST available to us, to Aline A. S. Leão for providing us with the knapsack solver used in the computational experiments, and to Dr. Raf Jans for facilitating his results of the CLSPST instances for validation purposes. Also, the authors would like to thank the anonymous referees for their valuable contributions to this paper.

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] F. Babonneau, C. Beltran, A. Haurie, C. Tadonki, and J.-P. Vial. Proximal-ACCPM: A versatile oracle based optimisation method. In E. J. Kontoghiorghes, C. Gatu, H. Amman, B. Rustem, C. Deissenberg, A. Farley, M. Gilli, D. Kendrick, D. Luenberger, R. Maes, I. Maros, J. Mulvey, A. Nagurney, S. Nielsen, L. Pau, E. Tse, and A. Whinston, editors, *Optimisation, Econometric and Financial Analysis*, volume 9 of *Advances in Computational Management Science*, pages 67–89. Springer Berlin Heidelberg, 2007.

[3] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87:385–399, 2000.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] COIN-OR. *OBOE: the Oracle Based Optimization Engine*, 2010. Available at http://projects.coin-or.org/OBOE [Accessed 2 October 2010].

[9] M. Colombo and J. Gondzio. Further development of multiple centrality correctors for interior point methods. *Computational Optimization and Applications*, 41(3):277–305, 2008.

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

[11] 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.

[12] 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.

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

[14] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[15] 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.

[16] A. Frangioni. Generalized bundle methods. *SIAM Journal on Optimization*, 13:117–156, May 2002.

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

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

[19] 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.

[20] 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.

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

[22] J. Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

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

[24] 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.

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

[26] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005.

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

[28] 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.

[29] R. Jans and Z. Degraeve. Improved lower bounds for the capacitated lot sizing problem with setup times. *Operations Research Letters*, 32(2):185 – 195, 2004.

[30] 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.

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

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

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

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

[35] 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.

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

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

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

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

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

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

[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] F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer US, 2005.

[44] F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer Berlin Heidelberg, 2010.

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

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