New developments in the primal-dual column generation technique

Gondzio, J, Gonzalez-Brevis, P & Munari, P 2013, 'New developments in the primal-dual column generation technique' European Journal of Operational Research, vol 224, no. 1, pp. 41-51.

New developments in the primal-dual column generation technique

Jacek Gondzio[a], Pablo González-Brevis[a,1,*], Pedro Munari[b,2]

[a]*School of Mathematics, The University of Edinburgh, James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom*
[b]*Instituto de Ciências Matemáticas e de Computação, University of São Paulo, Av. Trabalhador São-carlense, 400 - Centro, Cx. Postal 668, CEP 13560-970, São Carlos-SP, Brazil*

Abstract

The optimal solutions of the restricted master problems used by a standard column generation technique typically leads to an unstable behaviour 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, extensive computational experiments are presented using column generation formulations that arise in the context of integer programming. Three classical applications are used in the experiments: 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 [1, 2]. 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 that the columns in the coefficient matrix of this problem can be generated by following a known

*Telephone: +44 (0)131 650 5083, Fax: +44 (0)131 650 6553
Email addresses: J.Gondzio@ed.ac.uk (Jacek Gondzio), P.Gonzalez-Brevis@sms.ed.ac.uk (Pablo González-Brevis), munari@icmc.usp.br (Pedro Munari)

[1]Partially supported by Beca Presidente de la República and Facultad de Ingeniería, Universidad del Desarrollo, Chile.
[2]Supported by CAPES and FAPESP, Brazil.

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 [1, 3]. 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 [4], 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 [4, 5], 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 three classes of problems which are well-known 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 very degenerate restricted master problems, a property that usually causes instability in the standard column generation [6, 7, 8]. 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 on solving linear programming problems, but only a few attempts have been made on using 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, we describe the Dantzig-Wolfe decomposition, since it is applied in the problems considered in this paper. In Section 5, 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 6.

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 $\mathcal{A} \neq \emptyset$, 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

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

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 minimal 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. 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 Dantzig-Wolfe decomposition 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 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 [9, 10, 11] 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. Although good computational results have been reported for these techniques, they are too dependent of appropriate choices of the stability centre and penalization terms, which can be a difficult task in practice, specially for a variety of different problems. For performance comparisons involving stabilized approaches and the standard column generation, see [12, 11, 13].

An interior point column generation based on the simplex method is proposed in [12]. 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 for 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 [14] and [15], 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) [16, 17, 18] 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 avoid 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 [19]). The use of stabilization terms within the analytic centre cutting plane method has been successful, as showed in [20]. An implementation of this approach will be used in our computational experiments.

Another interior point approach is the primal-dual column generation technique. Proposed in [4], 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 [5] 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 formulations are widely employed.

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 [4], 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 $(c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|)$, 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 [4, 21].

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 Proposition 3.2, a lower bound of the optimal value of the MP can still be obtained.

Proposition 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 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 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. 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 [22]. 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 [23, 24].

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 [23, 25, 26]). 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 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 [27]. 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 [28, 29]). In this context, the focus was to provide stronger bounds when solving linear relaxations in order to speed up a branch-and-bound search.

Similar to the continuous case, an extended DWD can be 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 DWD to the integer programming given above, we consider here the convexification approach, although an alternative approach could have been used (see [29]). Consider the convex hull of the set $\mathcal{X}$, denoted by $C = conv(\mathcal{X})$. Assume that we know the sets of all extreme points p_q and extreme rays p_r that fully represent C. Hence, we can write any $x \in C$ as

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

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

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

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

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

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

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

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

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 $a_j = A p_j$, $\forall j \in Q$ and $\forall j \in \mathcal{R}$, a *relaxation* of the problem (4.2) is given by:

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

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

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

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

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, $q \in Q$, and every extreme ray p_r, $r \in \mathcal{R}$, we consider only a subset of them, for some $\overline{Q} \subseteq Q$ and $\overline{\mathcal{R}} \subseteq \mathcal{R}$. The dual problem associated to (4.3) has the same form as the resulting problem after applying Lagrangian relaxation for integer programming in the original problem (4.1) [30, 1].

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

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

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

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

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

where the extreme points of the subset $\mathcal{X}_k$ are represented by each p_q with $q \in Q_k$. 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 $k = 1, \ldots, K$:

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

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:

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

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

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

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

Although similar to the DW-MP, there is now only one subproblem associated to the aggregated master problem, which is given by any z^k_{SP}, 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

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

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.

5. 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 [32]. The solver is used as a black-box and its default parameters are used in the experiments.

- Primal-dual column generation (PDCGM): the suboptimal solutions of each RMP are obtained by using the interior point solver HOPDM [22], which is able to efficiently provide well-centred dual points and uses warmstarting strategies proposed in [23, 25, 26].

- 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 [33], a state-of-the-art implementation of the analytic centre strategy with additional stabilization terms [20].

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. The default accuracy, δ, has been set to 10^{-6}.

The standard column generation is known to be very unstable on solving the master problem formulations of the selected problems. Undoubtedly, it would be interesting to consider stabilized versions of the standard column generation in the computational experiments. However, the choice of proper stabilization parameters is known to be problem-dependant and makes it impractical for the variety of problems considered in this study. Nevertheless, comparisons involving stabilized and standard column generation procedures are available in the literature (see Section 5.4). 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. 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.

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 $(x_{11}, x_{12}, \ldots, x_{ij}, \ldots, x_{n(n-1)}, x_{nn})$.

5.1. 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 $j \in M = \{1, 2, \ldots, m\}$ [6]. 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. An integer programming formulation 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) ensure that the sum of the widths of all pieces cut from a roll does not exceed its width W.

5.1.1. 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 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, 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 $p \in P_i$. Using this notation, we have the following 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 associated to the master problem (5.2) is given by a set of n subproblems. Hence, for every $i \in N$, we have the following subproblem:

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

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

where $\overline{u}$ and $\overline{v}$ represent an arbitrary dual solution. Since the stock pieces are identical (*i.e.*, have the same width), 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 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 $\overline{v}_i$. To solve the subproblem we first solve a knapsack

problem given by:

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

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

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

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.

5.1.2. Computational results

To analyse the performance of the column generation strategies addressed here when solving problem (5.4), we have selected 262 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 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 [34], the implementation of which was provided by the author.

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 in 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. 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 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.

	SCG			PDCGM			ACCPM		
		Time (in s)			Time (in s)			Time (in s)	
Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total
S	571.8	2.0	2.6	368.9	1.9	4.8	466.2	3.0	10.5
L	881.3	153.7	155.8	591.4	35.5	44.5	734.0	143.9	182.4
All	671.0	50.6	51.7	440.3	12.7	17.5	552.1	48.2	65.6

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

		SCG			PDCGM			ACCPM		
			Time (in s)			Time (in s)			Time (in s)	
k	Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total
10	*S*	149.7	0.9	1.2	102.2	0.8	2.1	253.1	2.5	26.1
	L	251.4	75.8	77.0	158.4	15.1	18.3	368.3	82.6	148.7
	All	182.3	24.9	25.5	120.2	5.4	7.3	290.0	28.2	65.4
50	*S*	70.9	1.8	2.1	63.2	2.0	3.8	276.8	10.7	106.3
	L	133.7	56.6	58.2	97.1	18.8	23.1	400.2	45.5	277.6
	All	91.0	19.4	20.1	74.1	7.4	10.0	316.4	21.9	161.2
100	*S*	53.7	3.8	4.2	53.9	4.6	7.3	308.4	31.2	221.8
	L	101.0	66.3	67.8	82.3	25.4	31.5	449.4	96.4	525.2
	All	68.8	23.9	24.6	63.0	11.3	15.1	353.6	52.1	319.1

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

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, where k means the target and maximum number of columns that may be added at each iteration. For the set of small instances, the SCG is more efficient than 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 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 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.

5.2. Vehicle routing problem with time windows

Consider a set of vehicles $V = \{1, 2, \ldots, v\}$ available to serve a set of customers $C = \{1, 2, \ldots, n\}$ with demands d_i, for every $i \in C$. 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 must wait until the window is open (a_i). 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.

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 serve customer $i \in C$ is represented by the variable s_{ik}. We have 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) together with (5.6i) guarantee that each customer must be visited by only one vehicle. Constraints (5.6c) ensure that a vehicle cannot exceed its capacity. Both constraints together assure that the demand of each client has to be satisfied by only one vehicle. Moreover, constraints (5.6d) and (5.6f) ensure 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) link 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) assure 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 $(0, n+1)$.

5.2.1. 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 we did 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 the developments in Section 4, we obtain the following 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 (*i.e.*, have the same capacity), the subset $\mathcal{X}_k$ is the same for every $k \in V$ and hence the oracle will generate v identical columns. Similar to Section 5.1.1, 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). 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 [35] 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 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

	SCG			PDCGM			ACCPM		
		Time (in s)			Time (in s)			Time (in s)	
Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total
S	99.9	0.8	0.8	48.7	0.4	0.5	106.6	0.4	0.6
M	279.6	20.0	20.1	101.3	6.1	6.3	162.4	7.5	7.8
L	797.8	469.7	470.4	213.7	127.8	128.6	292.2	163.4	164.8
All	392.4	163.5	163.8	121.3	44.8	45.1	187.1	57.1	57.8

Table 3: VRPTW - Average results on 87 instances adding one column at a time.

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.

5.2.2. 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 [36]. 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 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 [37], with state-space relaxation and identification of unreachable nodes [38]. We have divided the instances in small ($n = 25$), medium ($n = 50$) and large ($n = 100$) classes. Each class has 29 instances.

Adding one column to the RMP. In Table 3 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 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, here the CPU time required for solving the RMPs is very small for the PDCGM and the ACCPM as well. In the VRPTW, the RMPs have the set partitioning 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 4 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

k	Class	SCG			PDCGM			ACCPM		
			Time (in s)			Time (in s)			Time (in s)	
		Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total
10	*S*	26.2	0.3	0.3	22.3	0.2	0.2	93.6	0.4	0.5
	M	66.7	6.1	6.2	37.7	2.4	2.6	122.0	5.4	5.7
	L	188.2	113.5	114.1	72.6	41.0	41.6	170.6	90.9	92.1
	All	93.7	40.0	40.2	44.2	14.5	14.8	128.7	32.2	32.8
50	*S*	14.4	0.2	0.2	18.1	0.1	0.2	92.2	0.4	0.5
	M	33.1	3.5	3.5	26.4	1.6	1.8	120.0	5.3	5.7
	L	88.0	54.9	55.5	48.6	26.0	27.0	165.2	85.8	87.8
	All	45.2	19.5	19.7	31.0	9.3	9.7	125.8	30.5	31.3
100	*S*	12.2	0.2	0.2	16.7	0.1	0.2	92.3	0.4	0.6
	M	26.0	2.9	3.0	23.2	1.4	1.7	119.7	5.4	5.8
	L	65.4	41.7	42.4	37.9	20.3	21.5	166.0	84.5	87.5
	All	34.5	14.9	15.2	25.9	7.3	7.8	126.0	30.1	31.3
200	*S*	9.8	0.1	0.2	16.1	0.1	0.3	92.4	0.4	0.6
	M	20.8	2.3	2.4	21.1	1.2	1.7	120.9	5.3	6.0
	L	50.1	33.1	33.9	32.4	16.9	18.7	167.4	82.1	89.2
	All	26.9	11.9	12.1	23.2	6.1	6.9	126.9	29.3	31.9
300	*S*	9.4	0.1	0.1	15.8	0.1	0.3	93.3	0.4	0.6
	M	18.0	2.1	2.2	20.4	1.2	1.8	121.1	5.2	6.1
	L	42.6	28.7	29.7	31.5	16.2	18.8	168.7	79.4	89.9
	All	23.3	10.3	10.7	22.6	5.8	7.0	127.7	28.3	32.2

Table 4: VRPTW - Average results on 87 instances adding at most k columns at a time.

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

5.3. Capacitated Lot-Sizing Problem with Setup Times

Consider a set of time periods $N = \{1, \ldots, n\}$ and a set of items $M = \{1, \ldots, m\}$ which 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 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). Following [39] this problem can be formulated as:

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

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

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

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

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

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

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

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 assumed 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) ensure that for every time period $t \in N$, the capacity required for the production plan should not exceed the available capacity of the machine in that period. Constraints (5.9c) are the inventory equations which guarantee 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. Note that backlogging is not allowed. Constraints (5.9d) assure 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) guarantee that the level of production and stock at each period t for each item j are non-negative.

5.3.1. Dantzig-Wolfe decomposition

As with the CSP and the 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, 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 every $j \in M$. 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:

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

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

for each $j \in M$. Observe that each subproblem is a single-item lot sizing problem with modified production and setup costs, and without capacity constraint. Hence, it can be solved by the Wagner-Whitin algorithm [40]. 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 for (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.

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

[(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 5: CLSPST - Average results on 751 instances adding one column per subproblem at a time.

5.3.2. Computational results

We have selected 751 instances proposed in [39] to test the aforementioned column generation strategies. 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 [40].

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 j 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 5, 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 5, 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, the PDCGM reduces the average number of outer iterations and total CPU time when compared with the ACCPM and the SCG.

In addition to the previous experiment, we have considered a set of more challenging instances. We have taken 3 instances from [39], which were used in [8] as a comparison set, to test the three column generation strategies. Additionally, we have selected 8 additional instances

	SCG			PDCGM			ACCPM		
		Time (in s)			Time (in s)			Time (in s)	
r	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total
5	27.5	4.7	4.7	11.5	1.5	1.6	22.5	3.1	3.2
10	32.0	62.7	62.7	15.6	20.4	21.0	29.5	49.1	49.5
15	38.4	308.8	308.8	20.0	103.8	106.2	36.4	273.2	274.3
20	45.5	975.6	975.8	25.9	350.5	358.4	42.4	938.7	941.0
All	35.8	337.9	338.0	18.3	119.0	121.8	32.7	316.0	317.0

Table 6: CLSPST - Average results on 11 modified instances adding one column per subproblem at a time.

from the sets of larger classes, X2 and X3. This small set of 11 instances[3] 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. 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.

5.4. Additional computational results from the literature

In [11], the authors present a comprehensive computational study comparing the standard column generation against the bundle method, a stabilized cutting plane method which uses quadratic stabilization terms. A set of five different applications were used in the computational experiments, including the CSP and the CLSPST (MILS problem in their paper). Regarding the CSP, the results indicate that using the bundle stabilization may slightly reduce the number of iterations at the cost of worsening CPU times. In the results obtained for the CLSPST, we see that the bundle method behaves poorly in terms of both, CPU times and number of iterations, when compared with the standard column generation.

Rousseau et al. [12] propose an interior point column generation technique in which the dual solutions are convex combinations of extreme dual points of the RMP that are obtained by randomly modifying the dual objective function. To analyse the computational performance of their approach, they have used the set of large VRPTW instances described in Section 5.2.2 (*i.e.*, Solomon's instances with $n = 100$). Only the results of 22 out of 29 instances were presented by the authors. Their comparison involved the implementations of the standard column generation as well as a stabilized version called BoxPen technique [10]. Since a different subproblem solver has been used in their computational experiments, it would not be appropriate to make a straightforward comparison of the figures presented in their tables with those presented in Tables 3 and 4 of Section 5.2.2. Hence, we have considered the gain obtained by each approach in relation to the standard column generation. According to their results, a well-tuned implementation of the BoxPen stabilization reduces the number of outer iterations by 16%, on average, while the total difference regarding CPU times is negligible. The interior point column generation technique proposed by the authors showed a better performance than the BoxPen stabilization, being 1.38 times faster than the standard column generation technique. For the same set of instances, PDCGM can be around 2 times faster than the SCG on average ($k = 200$).

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

6. 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 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 column generation procedure is applied in the context of integer programming. These experiments were based on column generation formulations of three classes of problems that are well-known in the literature, namely 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 lead to different types of master problem formulations: an aggregated master problem in the CSP, an aggregated master problem with a set partitioning 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. Hence, it was possible to verify the performance of the primal-dual column generation technique on a wide range of situations.

By analysing the computational results, we conclude that the PDCGM achieves the best overall performance when compared to the standard column generation method (SCG) and the analytic centre cutting plane method (ACCPM). Furthermore, 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 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] M. E. Lübbecke, J. Desrosiers, Selected topics in column generation, Operations Research 53 (6) (2005) 1007–1023.

[2] F. Vanderbeck, 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, L. A. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, 2010, pp. 431–502.
[3] F. Vanderbeck, Implementing mixed integer column generation, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer US, 2005, pp. 331–358.
[4] J. Gondzio, R. Sarkissian, Column generation with a primal-dual method, Technical Report 96.6, Logilab (1996).
[5] R. K. Martinson, J. Tind, An interior point method in Dantzig-Wolfe decomposition, Computers and Operation Research 26 (1999) 1195–1216.
[6] H. Ben Amor, J. Valério de Carvalho, Cutting stock problems, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer US, 2005, pp. 131–161.
[7] B. Kallehauge, J. Larsen, O. B. Madsen, M. M. Solomon, Vehicle routing problem with time windows, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer US, 2005, pp. 67–98.
[8] Z. Degraeve, 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) (2007) 909–920.
[9] R. E. Marsten, W. W. Hogan, J. W. Blankenship, The boxstep method for large-scale optimization, Operations Research 23 (3) (1975) 389–405.
[10] O. du Merle, D. Villeneuve, J. Desrosiers, P. Hansen, Stabilized column generation, Discrete Mathematics 194 (1-3) (1999) 229–237.
[11] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, F. Vanderbeck, Comparison of bundle and classical column generation, Mathematical Programming 113 (2008) 299–344.
[12] L.-M. Rousseau, M. Gendreau, D. Feillet, Interior point stabilization for column generation, Operations Research Letters 35 (5) (2007) 660–668.
[13] H. M. T. Ben Amor, J. Desrosiers, A. Frangioni, On the choice of explicit stabilizing terms in column generation, Discrete Applied Mathematics 157 (6) (2009) 1167–1184.
[14] J. Mitchell, B. Borchers, Solving real-world linear ordering problems using a primal-dual interior point cutting plane method, Annals of Operations Research 62 (1996) 253–276.
[15] J. E. Mitchell, Computational experience with an interior point cutting plane algorithm, SIAM Journal of Optimization 10 (4) (2000) 1212–1227.
[16] J. L. Goffin, A. Haurie, J. P. Vial, Decomposition and nondifferentiable optimization with the projective algorithm, Management Science 38 (2) (1992) 284–302.
[17] D. S. Atkinson, P. M. Vaidya, A cutting plane algorithm for convex programming that uses analytic centers, Mathematical Programming 69 (1995) 1–43.
[18] J.-L. Goffin, J.-P. Vial, Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method, Optimization Methods and Software 17 (2002) 805–868.
[19] J. E. Mitchell, Polynomial interior point cutting plane methods, Optimization Methods and Software 18 (5) (2003) 507–534.
[20] F. Babonneau, C. Beltran, A. Haurie, C. Tadonki, 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, A. Whinston (Eds.), Optimisation, Econometric and Financial Analysis, Vol. 9 of Advances in Computational Management Science, Springer Berlin Heidelberg, 2007, pp. 67–89.
[21] S. J. Wright, Primal-Dual Interior-Point Methods, SIAM, 1997.
[22] J. Gondzio, HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method, European Journal of Operational Research 85 (1995) 221–225.
[23] J. Gondzio, Warm start of the primal-dual method applied in the cutting-plane scheme, Mathematical Programming 83 (1998) 125–143.
[24] M. Colombo, J. Gondzio, Further development of multiple centrality correctors for interior point methods, Comput. Optim. Appl. 41 (3) (2008) 277–305.
[25] J. Gondzio, A. Grothey, Reoptimization with the primal-dual interior point method, SIAM Journal on Optimization 13 (3) (2003) 842–864.
[26] J. Gondzio, A. Grothey, A new unblocking technique to warmstart interior point methods based on sensitivity analysis, SIAM Journal on Optimization 19 (3) (2008) 1184–1210.
[27] G. B. Dantzig, P. Wolfe, Decomposition principle for linear programs, Operations Research 8 (1) (1960) 101–111.
[28] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance, Branch-and-price: Column generation for solving huge integer programs, Operations Research 46 (3) (1998) 316–329.
[29] F. Vanderbeck, On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm, Operations Research 48 (1) (2000) 111–128.
[30] A. M. Geoffrion, Lagrangean relaxation for integer programming, Mathematical Programming Studies (1974) 82–114.

[31] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience, 1988.
[32] IBM ILOG CPLEX v.12.1, Using the CPLEX Callable Library (2010).
[33] COIN-OR, OBOE: the Oracle Based Optimization Engine, Available at http://projects.coin-or.org/OBOE [Accessed 2 October 2010] (2010).
[34] A. A. S. Leão, Geração de colunas para problemas de corte em duas fases, Master's thesis, ICMC - University of Sao Paulo, Brazil (2009).
[35] S. Irnich, G. Desaulniers, Shortest path problems with resource constraints, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer US, 2005, pp. 33–65.
[36] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35 (2) (1987) pp. 254–265.
[37] G. Righini, M. Salani, New dynamic programming algorithms for the resource constrained elementary shortest path problem, Networks 51 (3) (2008) 155–170.
[38] D. Feillet, P. Dejax, M. Gendreau, C. Gueguen, An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle-routing problems, Networks 44 (2004) 216–229.
[39] W. W. Trigeiro, L. J. Thomas, J. O. McClain, Capacitated lot sizing with setup times, Management Science 35 (3) (1989) 353–366.
[40] H. M. Wagner, T. M. Whitin, Dynamic version of the economic lot size model, Management Science 5 (1) (1958) 89–96.